daiteq

Technical Report

2021-IETR021ESAIP013-V2.0

ESA 4000122242/17/NL/LF

Deliverable: CCN2/D1+D2
Version: V2.0
Status: draft
Date: 2021/11/08

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 6 provide an overview of the benchmarks and tools used together with a definition and description of the evaluated NOEL-V and LEON configurations.
- Section 8 presents performance results for 4-core NOEL-V systems for each of the defined configurations.
- Section 9 presents performance results for LEON-based systems.
- Section 10 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 11 provides final conclusions of the performed experiments.

Each of Sections 8, 9 and 10 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 8 and 9 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 10 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 8 and 9.

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 [RTEMSC], 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 1. The individual processor configurations used are listed in Table 2 for the NOEL-V systems and Table 5 for the LEON-based systems.

Table 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 2: 64-bit NOEL-V configurations and BSPs used for initial benchmarking (GRLIB 2020.4).

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

Table 3: 64-bit NOEL-V configurations and BSPs used in implementation experiments and for final benchmarking (GRLIB 2021.2 incl. daiFPUrv+L2Cache).

ID	Altern.	Platform	BSP	CFG	FPU	#CORES
.	ID	ID	RTEMS	ID	.	TOTAL
NV1D1	.	P5	noel64im	CFG1	daiFPUrv	3
NV1D2	.	P5	noel64imafd	CFG1	daiFPUrv	3
NV1D3	.	P5	noel64ima_smp	CFG1	daiFPUrv	3
NV1D4	.	P5	noel64imafd_smp	CFG1	daiFPUrv	3
NV1N1	.	P5	noel64im	CFG1	nanofpunv	3
NV1N2	.	P5	noel64imafd	CFG1	nanofpunv	3
NV1N3	.	P5	noel64ima_smp	CFG1	nanofpunv	3
NV1N4	.	P5	noel64imafd_smp	CFG1	nanofpunv	3
NV2D1	.	P5	noel64im	CFG2	daiFPUrv	4
NV2D2	.	P5	noel64imafd	CFG2	daiFPUrv	4
NV2D3	.	P5	noel64ima_smp	CFG2	daiFPUrv	4
NV2D4	.	P5	noel64imafd_smp	CFG2	daiFPUrv	4
NV2N1	.	P5	noel64im	CFG2	nanofpunv	4
NV2N2	.	P5	noel64imafd	CFG2	nanofpunv	4
NV2N3	.	P5	noel64ima_smp	CFG2	nanofpunv	4
NV2N4	.	P5	noel64imafd_smp	CFG2	nanofpunv	4
NV46X1	.	P5	noel64im	CFG4	N/A	6
NV46X3	.	P5	noel64ima_smp	CFG4	N/A	6
NV47X1	.	P5	noel64im	CFG4	N/A	7
NV47X3	.	P5	noel64ima_smp	CFG4	N/A	7

Table 4: 32-bit NOEL-V configurations and BSPs used in implementation experiments (GRLIB 2021.2 w/ daiFPUrv).

ID	Altern.	Platform	BSP	CFG	FPU	#CORES
.	ID	ID	RTEMS	ID	.	TOTAL
NVS1D1	.	P5	noel32im	CFG1	daiFPUrv	3
NVS1D2	.	P5	noel32imafd	CFG1	daiFPUrv	3
NVS1D3	.	P5	noel32ima_smp	CFG1	daiFPUrv	3
NVS1D4	.	P5	noel32imafd_smp	CFG1	daiFPUrv	3
NVS1N1	.	P5	noel32im	CFG1	nanofpunv	4
NVS1N2	.	P5	noel32imafd	CFG1	nanofpunv	4
NVS1N3	.	P5	noel32ima_smp	CFG1	nanofpunv	4
NVS1N4	.	P5	noel32imafd_smp	CFG1	nanofpunv	4
NVS2D1	.	P5	noel32im	CFG2	daiFPUrv	4
NVS2D2	.	P5	noel32imafd	CFG2	daiFPUrv	4
NVS2D3	.	P5	noel32ima_smp	CFG2	daiFPUrv	4
NVS2D4	.	P5	noel32imafd_smp	CFG2	daiFPUrv	4
NVS2N1	.	P5	noel32im	CFG2	nanofpunv	4
NVS2N2	.	P5	noel32imafd	CFG2	nanofpunv	4
NVS2N3	.	P5	noel32ima_smp	CFG2	nanofpunv	4
NVS2N4	.	P5	noel32imafd_smp	CFG2	nanofpunv	4
NVS48X1	.	P5	noel32im	CFG4	N/A	8
NVS48X2	.	P5	noel32ima_smp	CFG4	N/A	8

Table 5: LEON setups used for benchmarking[1] .

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

[1] 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 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[1] web works fine with GRLIB version 2020.4
- The NOEL-V distribution *noel-buildroot-2020.08-1.0*[Page 13, 1] 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.

[1] available for download from the Cobham Gaisler web

5 NOEL-V configurations in GRLIB 2020.4

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 7. 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[1]
- FPU instructions (except load and store) if `fpu_lane` is configured to 1[2]

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[Page 13, 1]
- FPU instructions (except load and store) if `fpu_lane` is configured to 0[?]

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 6 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 6: 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

continues on next page

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

Table 6 – 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 7. 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 8.

Table 7: 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**. Values in brackets denote features that are not configured.

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[3]	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							

continues on next page

Table 7 – 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**
bhtentries	128	128	**128**	**64**	**64**	**32**	32
bhtlength	5	5	5	5	**5**	**2**	2
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 7 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 8 shows resources needed to implement each of the NOEL-V configurations shown in Table 7, 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.t

None of the designs was configured with L2Cache.

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 7 and instantiates **cpucorenv**.

The module *cpucorenv* instantiates

- **iunv** - the actual RISC-V processor pipeline

[3] 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.

- mul64 and div64 - integer multiplier and divider when `ext_m` is set to *1*
- cctrlnv - cache controller
- bhtnv - branch history table
- 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*[?]

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 8: 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 (GRLIB 2020.4)

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[4] . The results are summed in Table 9.

Table 9: 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

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

6 NOEL-V configurations in GRLIB 2021.2

The configurations in the GRLIB 2021.2 have been slightly changed as shown in Table 10. The changes can be summarised as follows:

- support for the compressed instructions (C-extension)
- support for hypervisor (H-extensions)
- introduction of TLB for the hypervisor mode
- *btbentries* decreased to *16* in the *HPP* configuration (*CFG_CFG=0*).

In addition, some of the designs in GRLIB 2021.2 were configured with L2Cache to evaluate the influence of L2Cache on multi-core performance.

Table 10: NOEL-V configurations as defined in *GRLIB 2021.2*. *CFG_CFG* is defined in *config.vhd* inside the design directory. Values that change are shown in **bold**. Values in brackets denote features that are not configured.

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							
sin-gle_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	1	1	1	1	**1**	**0**	0
ext_h	1	1	1	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	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

continues on next page

Table 10 – 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**
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)
htlbnum	8	8	8	(1)	(1)	(1)	(1)
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
predictor	2	2	2	2	2	2	2
Branch target prediction							
btbentries	16	16	16	16	**16**	**8**	8
btbsets	2	2	2	2	2	2	2

Besides the architectural improvements, GRLIB 2021.2 also supports configuration of NOEL-V targets with the *make xconfig* tool. The official configurations that can be selected with the *make xconfig* tool are listed in Table 11.

Table 11: NOEL-V - Mapping between configuration names used in *make xconfig*, in the NOELVSYS datasheet (*grip.pdf*) and *CFG_CFG* in the configuration file (*config.vhd*).

selection in xconfig	Abbrev.	CFG_CFG	Dual issue	Multi-core	FPU
High performance cfg	HPP	0	Y	Y	Y
General purpose cfg(dual issue)	GPP-DI	1	Y	Y	Y
General purpose cfg(dual issue)	GPP-SI	2	N	Y	Y
Minimal cfg	MIN	4	N	Y	N
Tiny cfg	TINY	5	N	N	N

Since in GRLIB 2021.2 *HPP* is the same as *GPP-DI*[1] , the high-performance configuration *HPP* was not evaluated in the implementation and performance tests.

[1] this will probably change in the future

6.1 Selecting RV32 or RV64

IMPORTANT!!! Selection between the RV32 and RV64 instruction sets is **NOT** done with the constant *CFG_NOELV_XLEN* that is defined in *config.vhd* in the current design directory.

Selecting between RV32 and RV64 is a bit tricky. The safest way is to select the corresponding configuration with the *make xconfig* tool.

In some situations it may be desirable to change the instruction set directly in a configuration file to guarantee that the rest of the parameters remain the same, e.g. when cloning designs. The GRLIB 2021.2 package defines the constant *XLEN*, *NV_XLEN* and *CFG_XLEN* at different places: the designer should check the *config.vhd*, *Makefile* and *.config* files in the design directory as well as the files *lib/gaisler/noelv/pkg/noelv.vhd*, *lib/gaisler/noelv/core/cpucorenv.vhd*. The designer should be well aware that in GRLIB 2021.2 the selection is made by compiling the correct NOELV configuration package; there is one package for each instruction set:

The mechanism is not as straightforward as one may think. The key is the file *.config* in the design directory. The best procedure is to select the proper instruction set there by selecting one from the following assignments

```
CONFIG_NOELV_RV64=y
CONFIG_NOELV_RV32=y
```

and then rebuild the scripts by

```
$ chmod 000 xilinx_lib
$ make clean
$ make scripts
$ chmod 700 xilinx_lib
$ make map_xilinx_7series_lib
```

and eventually run the simulation and synthesis

```
$ make vsim
$ make vivado
```

6.2 Maximum number of NOEL-V cores and the maximal operating frequency (GRLIB 2021.2)

The foreseen high-performance NOEL-V configurations are RV64 that use an external DDR memory controller and a L2Cache to improve multi-core performance. Using the internal AHBRAM memory is an option for applications that would fit in the 1MB space, but unfortunately the CoreMark-Pro and FPMark workloads that are used to evaluate the NOEL-V performance require significantly more space, and would not fit in AHBRAM.

The implementation experiments shown below answer the following questions:

- What is the maximal single-core frequency for each NOEL-V configuration?
- How many cores do fit in the XCKU040 device with the smallest memory option available[2] ?
- What is the highest operating frequency for the highest number of cores when using AHBRAM?

[2] (i.e. AHBRAM, since configurations with the MIG and MIG+L2Cache options consume more resources than AHBRAM)

- What is the highest operating frequency for the highest number of cores when using MIG with L2Cache?

This implies the order of steps that were performed for each NOEL-V configuration shown in Table 11:

1. Start with NOEL-V systems configured with *AHBRAM* without *L2Cache*. Set the number of cores to 1.
2. Determine the maximum operating frequency for single-core configurations, i.e. find the values of *CFG_CLKMUL* and *CFG_CLKDIV* that result in the smallest possible value of the worst negative slack (WNS).
3. Keeping the frequency, set the number of cores to 4, and check if the design fits in the device.
4. If the design does not fit in the device, decrease the number of cores by 1 and check again. Repeat until the design fits in the device. This will likely yield a negative WNS value.
5. Decrease the operating frequency until the WNS value is positive.
6. Change the memory from *AHBRAM* to *MIG*, this usually worsens the WNS.
7. Decrease the operating frequency until the WNS value is positive.
8. Enable *L2Cache*
9. Decrease the operating frequency until the WNS value is positive.
10. Try increasing the number of cores, but do not decrease the operating frequency too much not to hamper the increased number of cores by a significant drop of the operating frequency.

IMPORTANT!!! For configurations that use *MIG* the values of *CFG_CLKMUL* and *CFG_CLKDIV* in *config.vhd* no longer configure the operating frequency; the configuration has to be done manually inside the MIG IP core definition in the *<design>/vivado/mig.xci* file.

The implementation experiments evaluated only the non-fault tolerant configurations only as the fault-tolerant option was not available.

Table 12 shows values for the clock multiplier (*CFG_CLKMUL*) and divider (*CFG_CLKDIV*) for system frequencies used in the implementation experiments.

Table 12: NOEL-V - CFG_CLKMUL and CFG_CLKDIV values for configurations with AHBRAM, KCU105, on-board oscillator frequency 300MHz.

CLKMUL	CLKDIV	FREQ [MHz]
4	14	85.714
3	10	90.000
4	13	92.308
4	12	100.000
4	11	109.091
3	8	112.500
4	10	120.000
3	7	128.571
4	9	133.333
4	8	150.000
4	7	171.429

The following section lists all major configurations of the NOEL-V system, including the data word width, floating-point option, and memory controller configuration, together with the targeted frequency and the achieved worst negative slack (WNS) as reported by the Xilinx Vivado 2020.1 tool when targetting the platform P5 (Table 1).

6.2.1 RV64

The following tables summarize results for 64-bit NOEL-V configurations.

Table 13: NOEL-V - implementation results for 64-bit configurations (RV64). The second row in the table header states the name of the configuration constants in config.vhd without the prefix *CFG_*. *CFG_FPUCONF* was introduced as part of the daiFPUrv extensions to GRLIB 2021.2. *TOO BIG(1)* in the last columns means that the design failed on available resources during synthesis. *TOO BIG(2)* denotes configurations that failed on available resources during placement ("combined CLB capacity"). *(3)* in the last column denotes implementations with a negative WNS that nevertheless executed all right in hardware.

CFG	RV	CORES	L2	MIG	FPU	FREQ [MHz]	WNS [ns]
CFG	XLEN	NCPU	L2_EN	MIG_7SERIES	FPUCONF	CLKMUL, CLKDIV	Vivado
CFG_CFG=1, daiFPUrv, ahbram							
1	64	1	N	N	daiFPUrv	120	0.072
1	64	1	N	N	daiFPUrv	128.571	-0.223
1	64	4	N	N	daiFPUrv	100	-0.692
1	64	4	N	N	daiFPUrv	90	0.253
1	64	5	N	N	daiFPUrv	90	TOO BIG(1)
CFG_CFG=1, daiFPUrv, MIG							
1	64	4	N	Y	daiFPUrv	90	TOO BIG(1)
1	64	3	N	Y	daiFPUrv	100	-0.116
CFG_CFG=1, daiFPUrv, MIG, L2Cache							
1	64	3	Y	Y	daiFPUrv	100	-0.363
1	64	3	Y	Y	daiFPUrv	100	-0.450
1	64	3	Y	Y	daiFPUrv	92.308	-0.198
1	64	3	Y	Y	daiFPUrv	90	-0.240(3)
1	64	3	Y	Y	daiFPUrv	85.714	0.032
CFG_CFG=1, nanofpunv, ahbram							
1	64	1	N	N	nanofpunv	120	0.010
1	64	1	N	N	nanofpunv	128.571	-0.899
1	64	4	N	N	nanofpunv	100	-1.001
1	64	4	N	N	nanofpunv	90	0.005
1	64	5	N	N	nanofpunv	90	TOO BIG(1)
CFG_CFG=1, nanofpunv, MIG							
1	64	4	N	Y	nanofpunv	90	-0.855
1	64	3	N	Y	nanofpunv	100	0.000
CFG_CFG=1, nanofpunv, MIG, L2Cache							

continues on next page

Table 13 – continued from previous page

CFG	RV	CORES	L2	MIG	FPU	FREQ [MHz]	WNS [ns]
CFG	XLEN	NCPU	L2_EN	MIG_7SERIES	FPUCONF	CLKMUL, CLKDIV	Vivado
1	64	3	Y	Y	nanofpunv	100	-0.209
1	64	3	Y	Y	nanofpunv	92.308	0.029
CFG_CFG=2, daiFPUrv, ahbram							
2	64	1	N	N	daiFPUrv	120	0.376
2	64	1	N	N	daiFPUrv	128.571	0.175
2	64	4	N	N	daiFPUrv	100	0.125
2	64	5	N	N	daiFPUrv	100	TOO BIG(1)
2	32	5	N	N	daiFPUrv	100	TOO BIG(1)
2	32	4	N	N	daiFPUrv	100	0.216
CFG_CFG=2, daiFPUrv, MIG							
2	64	4	N	Y	daiFPUrv	100	0.040
CFG_CFG=2, daiFPUrv, MIG, L2Cache							
2	64	4	Y	Y	daiFPUrv	100	0.038
2	64	4	Y	Y	daiFPUrv	100	0.023
2	64	4	Y	Y	daiFPUrv	109.091	0.046
2	64	4	Y	Y	daiFPUrv	112.500	0.047
2	64	4	Y	Y	daiFPUrv	120	-0.388
CFG_CFG=2, nanofpunv, ahbram							
2	64	1	N	N	nanofpunv	120	0.285
2	64	1	N	N	nanofpunv	128.571	0.122
2	64	4	N	N	nanofpunv	100	0.153
2	64	5	N	N	nanofpunv	100	0.140
2	64	6	N	N	nanofpunv	100	TOO BIG(1)
2	64	4	N	N	nanofpunv	100	0.153
CFG_CFG=2, nanofpunv, MIG							
2	64	4	N	Y	nanofpunv	100	0.010
CFG_CFG=2, nanofpunv, MIG, L2Cache							
2	64	4	Y	Y	nanofpunv	100	0.023
2	64	4	Y	Y	nanofpunv	109.091	0.022
2	64	4	Y	Y	nanofpunv	112.500	0.016
2	64	4	Y	Y	nanofpunv	120	0.022
CFG_CFG=4, no fpu, ahbram							
4	64	1	N	N	none	120	0.523
4	64	1	N	N	none	128.571	0.544
4	64	4	N	N	none	133.333	0.069
4	64	4	N	N	none	150	-0.225
4	64	6	N	N	none	100	0.431
4	64	10	N	N	none	100	TOO BIG(1)
4	64	9	N	N	none	100	TOO BIG(1)
4	64	8	N	N	none	100	0.569
4	64	8	N	N	none	120	0.002
4	64	8	N	N	none	128.571	0.034
4	64	8	N	N	none	133.333	-0.105
CFG_CFG=4, no fpu, MIG, L2Cache							

continues on next page

Table 13 – continued from previous page

CFG	RV	CORES	L2	MIG	FPU	FREQ [MHz]	WNS [ns]
CFG	XLEN	NCPU	L2_EN	MIG_7SERIES	FPUCONF	CLKMUL, CLKDIV	Vivado
4	64	8	Y	Y	none	120	TOO BIG(2)
4	64	7	Y	Y	none	120	0.030
4	64	6	Y	Y	none	120	0.021

6.2.2 Analysis

The conclusions of the design space exploration experiments are:

1. The highest achievable clock frequency for NOEL-V designs in XCKU040 is not likely to exceed 150MHz. This frequency was achieved for a single-core design for the TINY configuration, RV32.
2. The highest number of cores is not likely to exceed 8. This was achieved for the MIN configuration, RV64 as well as RV32.
3. When using MIG and L2Cache instead of just AHBRAM, the number of cores needs to be decreased by 1 to compensate for the extra resource requirements due to the memory and cache controllers.
4. While the dual-issue GPP configuration, RV64 can be implemented with up to 3 cores in the device, the single-issue GPP configuration, RV64 can be implemented with up to 4 cores. At the same time the maximum operating frequency for the single-issue GPP implementation will be about 30% higher than for the dual-issue GPP.
5. In general, designs that are configured with the nanofpunv achieve a slightly higher operating frequency than designs with daiFPUrv. This results from the simpler nature of the nanofpunv implementation, while daiFPUrv executes operations in about half the clock cycles than nanofpunv.

The final configurations that were used for carrying out the performance experiments are summarized in Table 14.

Table 14: NOEL-V - 64-bit implementations that were used in the implementation experiments and for executing the PWLS, CoreMark, CoreMark-Pro and FPMark benchmarks. The second row in the table header states the name of the configuration constants in config.vhd without the prefix *CFG_*. *CFG_FPUCONF* was introduced as part of the daiFPUrv extensions to GRLIB 2021.2.

Config.	CFG	RV	CORES	L2	MIG	FPU	FREQ [MHz]	WNS [ns]
ID	CFG	XLEN	NCPU	L2_EN	MIG_7SERIES	FPUCONF	CLKMUL, CLKDIV	Vivado
NV1D	1	64	3	Y	Y	daiFPUrv	85.714	0.032
NV1N	1	64	3	Y	Y	nanofpunv	92.308	0.029
NV2D	2	64	4	Y	Y	daiFPUrv	112.500	0.047
NV2N	2	64	4	Y	Y	nanofpunv	120	0.022
NV4X6	4	64	6	Y	Y	none	120	0.021
NV4X7	4	64	7	Y	Y	none	120	0.030

The configurations shown in Table 15 summarizes 32-bit NOEL-V configurations that were used for comparing the difference in resource requirements between 32-bit and 64-bit NOEL-V systems.

Table 15: NOEL-V - 32-bit implementations used in implementation experiments. The second row in the table header states the name of the configuration constants in config.vhd without the prefix *CFG_*. *CFG_FPUCONF* was introduced as part of the daiFPUrv extensions to GRLIB 2021.2.

Config.	CFG	RV	CORES	L2	MIG	FPU	FREQ [MHz]	WNS [ns]
ID	CFG	XLEN	NCPU	L2_EN	MIG_7SERIES	FPUCONF	CLKMUL, CLKDIV	Vivado
NVS1D	1 X	32	3	N	N	daiFPUrv	100	-0.525
NVS1N	1 X	32	4	N	N	nanofpunv	100	0.006
NVS2D	2 X	32	4	N	N	daiFPUrv	120	0.103
NVS2N	2 X	64	4	Y	Y	nanofpunv	120	0.022
NVS4X8	4 X	64	8	Y	Y	none	120	0.021
NVS5	5 X	64	1	Y	Y	none	120	0.030

6.3 Resources

6.3.1 RV64

The following tables update the resource information shown in Table 8 for the final NOEL-V configurations generated in the implementation experiments and subsequently used to measure performance of NOEL-V with the daiFPUrv floating unit.

Table 16: NOEL-V - total resources used by the complete NOEL-V system configured with parameters shown in Table 14.

Cfg. ID	LUTs			SRLs	FFs	RAMB		URAM	DSP
.	Total	Logic	LRAM	SRLs	.	36	18	.	Blocks
NV1D	198458	194749	2392	1317	109673	94	142	0	78
NV1N	170659	167814	2152	693	95368	94	142	0	57
NV2D	214191	210794	1872	1525	126682	95	184	0	103
NV2N	176814	174569	1552	693	107606	95	184	0	75
NV4X6	186204	183959	1552	693	107987	97	184	0	99
NV4X7	212631	210306	1632	693	120624	98	212	0	115

Table 17: NOEL-V - resources used by the processor core (*cpucorenv.vhd*), including FPU, in a NOEL-V system configured with parameters shown in Table 14.

Cfg. ID	LUTs			SRLs	FFs	RAMB		URAM	DSP
.	Total	Logic	LRAM	SRLs	.	36	18	.	Blocks
NV1D	56480	55832	440	208	24287	1	42	0	25
NV1N	47294	46934	360	0	19522	1	42	0	18
NV2D	46172	45764	200	208	22518	1	42	0	25
NV2N	36872	36752	120	0	17748	1	42	0	18
NV46X	25750	25670	80	0	11207	1	28	0	16
NV47X	25749	25669	80	0	11207	1	28	0	16

Table 18: NOEL-V - resources used by the FPU (*fpucorenv.vhd*) in a NOEL-V system configured with parameters shown in Table 14.

Cfg. ID	LUTs			SRLs	FFs	RAMB		URAM	DSP
.	Total	Logic	LRAM	SRLs	.	36	18	.	Blocks
NV1D	15313	14985	120	208	5576	0	0	0	9
NV1N	6100	6060	40	0	806	0	0	0	2
NV2D	15402	15074	120	208	5576	0	0	0	9
NV2N	6106	6066	40	0	806	0	0	0	2

6.3.2 RV32

Although the 32-bit NOEL-V configurations are not of the primary interest of this activity, we present the implementation data for selected configurations here to provide a comparison between resource requirements of the 32-bit and 64-bit NOEL-V versions.

Table 19: NOEL-V - total resources used by the complete NOEL-V system configured with parameters shown in Table 14.

Cfg. ID	LUTs			SRLs	FFs	RAMB		URAM	DSP
.	Total	Logic	LRAM	SRLs	.	36	18	.	Blocks
NVS1D	158038	154713	2008	1317	99393	94	142	0	42
NVS1N	172410	169717	2000	693	102552	95	184	0	27
NVS2D	197632	194363	1744	1525	114668	95	184	0	55
NVS2N	168487	166370	1424	693	96628	95	184	0	27
NVS48X	203890	201741	1456	693	116694	99	240	0	35
NVS5	32734	30785	1256	693	33049	25	26	0	7

Table 20: NOEL-V - resources used by the processor core (*cpucorenv.vhd*), including FPU, in a NOEL-V system configured with parameters shown in Table 14.

Cfg. ID	LUTs			SRLs	FFs	RAMB		URAM	DSP
.	Total	Logic	LRAM	SRLs	.	36	18	.	Blocks
NVS1D	42411	41891	312	208	20753	1	42	0	13
NVS1N	35165	34933	232	0	16242	1	42	0	6
NVS2D	41513	41137	168	208	18752	1	42	0	13
NVS2N	34263	34175	88	0	14246	1	42	0	6
NVS48X	21053	21005	48	0	9505	1	28	0	4
NVS5	9599	9495	104	0	5300	0	18	0	4

Table 21: NOEL-V - resources used by the FPU (*fpucorenv.vhd*) in a NOEL-V system configured with parameters shown in Table 14.

Cfg. ID	LUTs			SRLs	FFs	RAMB		URAM	DSP
.	Total	Logic	LRAM	SRLs	.	36	18	.	Blocks
NVS1D	13245	12917	120	208	5317	0	0	0	9
NVS1N	6103	6063	40	0	807	0	0	0	2
NVS2D	13245	12917	120	208	5317	0	0	0	9
NVS2N	6103	6063	40	0	807	0	0	0	2

6.4 Conclusions for practical use

1. daiFPUrv requires more than 2x the LUTs, 5x the FFs and 4.5x the DSP blocks that are required by nanofpunv.
2. The dual-issue GPP core (cpucorenv) requires about 1.25x the LUTs and FFs per core that are required by the single-issue GPP core.
3. The 64-bit NOEL-V cores (RV64) require about 2-13kLUTs more than 32-bit NOEL-V configurations (RV32).
4. The 64-bit NOEL-V in the newer GRLIB 2021.2 version requires about 4-8kLUTs more per core (cpucorenv) than in GRLIB 2020.4.
5. daiFPUrv configured for RV32 requires about 2kLUTs less than when configured for RV64[3] .
6. The biggest NOEL-V configurations that can be implemented in XCKU040 use about 180-210kLUTs and 100-120kFFs.
7. For NOEL-V configurations with FPU (i.e. GPP) the maximum number of cores that fit in XCKU040 is 3-4, with the maximal operating frequency between 85-120MHz.
8. In general, designs with daiFPUrv achieve slightly lower operating frequency than designs with nanofpunv.

[3] 64-bit integer conversions not configured for RV32

7 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 22: 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 22. 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 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.

8 NOEL-V performance

8.1 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.

8.1.1 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.*

8.1.2 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).

8.1.3 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
```

8.1.4 Measurements for GRLIB 2020.4

The results shown in Table 23 were measured with PWLS benchmarks compiled for the *noel64imafd* BSP (hardware FPU). The results shown in Table 24 were measured with PWLS benchmarks compiled for the *noel64im* BSP (SoftFloat).

Table 23: 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 24: 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 1 to 3. The figures show the corresponding benchmark performance at 100MHz. For Figures 1 and 2 higher values denote better processor performance, while for Fig. 3 lower values indicate better performance.

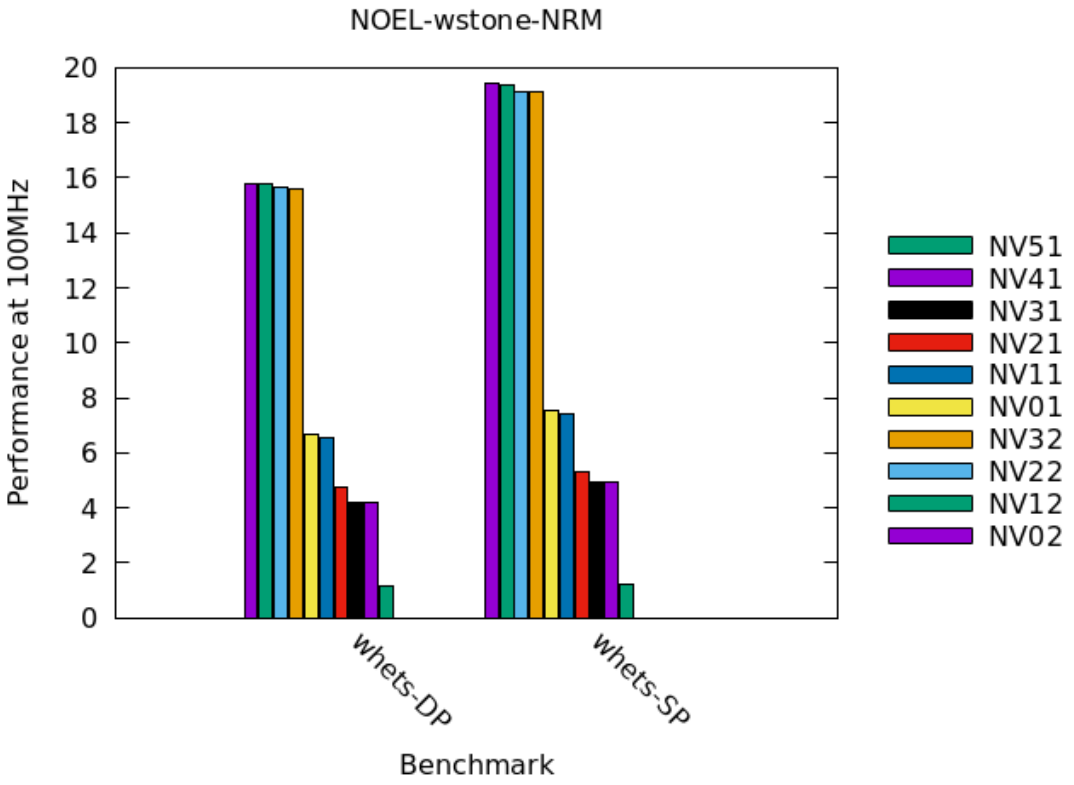

Figure 1: NOEL - Whetstone performance for NOEL targets, GRLIB 2020.4, MWIPS at 100MHz. Higher values denote better performance. Configuration names ending with `2` denote execution using the *nanoFPU*, while names ending with `1` denote execution using the *SoftFloat* library.

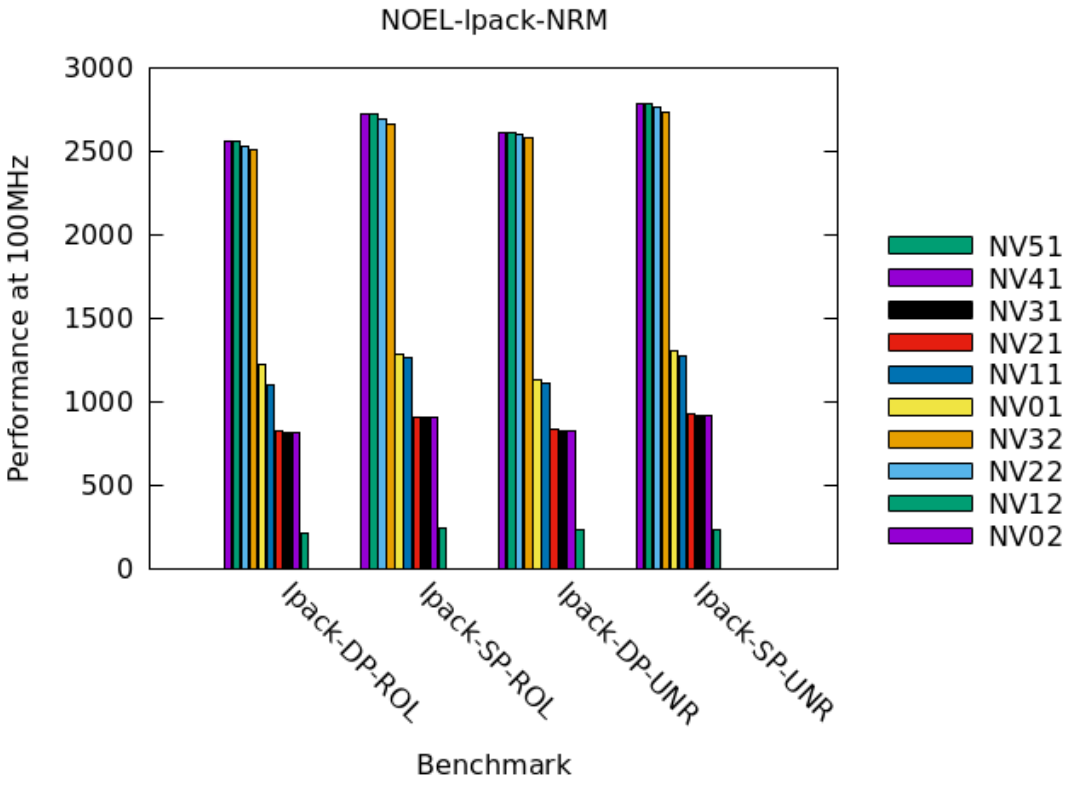

Figure 2: NOEL - Linpack performance for NOEL targets, GRLIB 2020.4, Kflops at 100MHz. Higher values denote better performance. Configuration names ending with `2` denote execution using the *nanoFPU*, while names ending with `1` denote execution using the *SoftFloat* library.

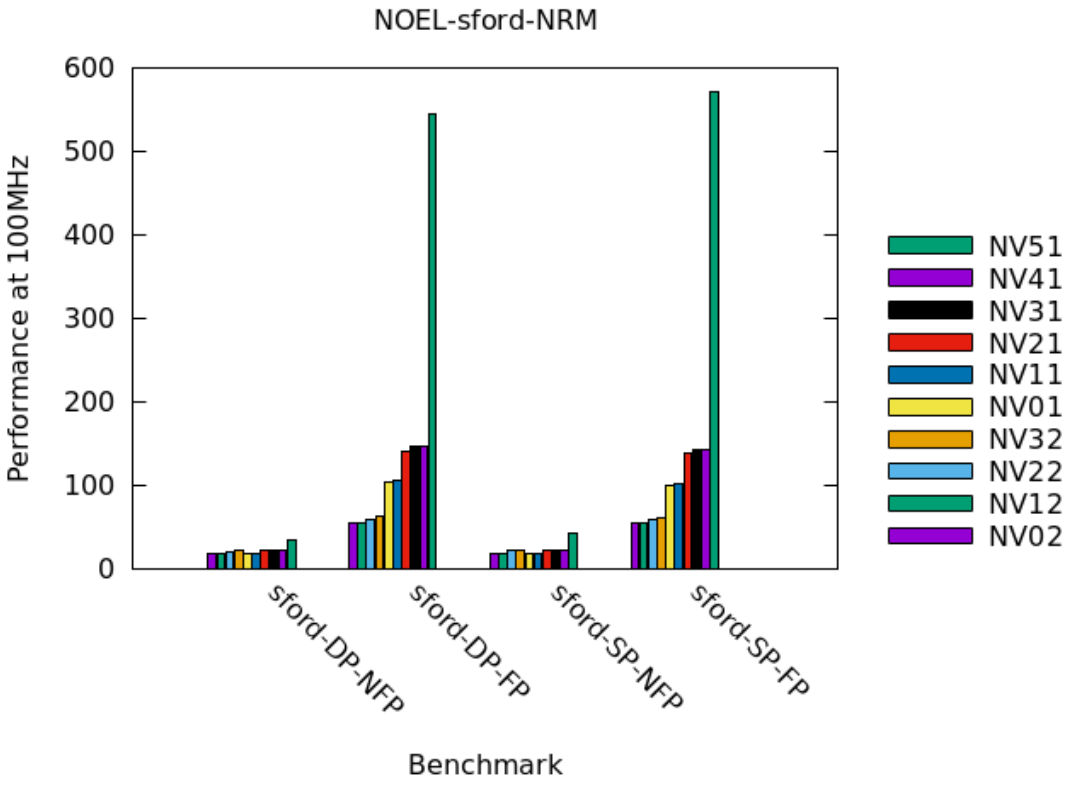

Figure 3: NOEL - Stanford performance for NOEL targets, GRLIB 2020.4, milliseconds at 100MHz. Lower values denote better performance. Configuration names ending with `2` denote execution using the *nanoFPU*, while names ending with `1` denote execution using the *SoftFloat* library.

8.1.5 Measurements for GRLIB 2021.2

The results shown in Table 25 were measured with PWLS benchmarks compiled for the *noel64imafd* BSP (hardware FPU). The results shown in Table 26 were measured with PWLS benchmarks compiled for the *noel64im* BSP (SoftFloat).

Table 25: NOEL-V single-core performance summary, daiFPUrv vs nanoFPU - PWLS, GPP configurations, native floating-point. Performance at 100MHz.

Benchmark	Units	NV1D2	NV1N2	NV2D2	NV2N2
whets-DP	MWIPS	32.656	15.580	32.044	15.459
whets-SP	MWIPS	34.160	19.305	33.433	19.101
lpack-DP-ROL	Kflops	4480	2657	4381	2623
lpack-SP-ROL	Kflops	4677	2788	4564	2747
lpack-DP-UNR	Kflops	4621	2707	4577	2692
lpack-SP-UNR	Kflops	4834	2842	4785	2825
sford-DP-NFP	ms	16	16	20	20
sford-DP-FP	ms	46	52	50	57
sford-SP-NFP	ms	16	16	20	20
sford-SP-FP	ms	46	52	50	56

Table 26: NOEL-V single-core performance summary, daiFPUrv vs nanoFPU - PWLS, GPP and MIN configurations, SoftFloat. Performance at 100MHz.

Benchmark	Units	NV1D1	NV1N1	NV2D1	NV2N1	NV46X1
whets-DP	MWIPS	6.670	6.671	4.711	4.713	4.430
whets-SP	MWIPS	7.724	7.722	5.335	5.333	5.120
lpack-DP-ROL	Kflops	1162	1162	847	847	835
lpack-SP-ROL	Kflops	1318	1318	927	927	918
lpack-DP-UNR	Kflops	1165	1165	852	852	838
lpack-SP-UNR	Kflops	1374	1374	945	946	937
sford-DP-NFP	ms	16	16	20	20	20
sford-DP-FP	ms	99	99	138	138	143
sford-SP-NFP	ms	16	16	20	20	20
sford-SP-FP	ms	95	95	136	136	138

The values in the table are also shown in Figures 4 to 6. The figures show the corresponding benchmark performance at 100MHz. For Figures 4 and 5 higher values denote better processor performance, while for Fig. 6 lower values indicate better performance.

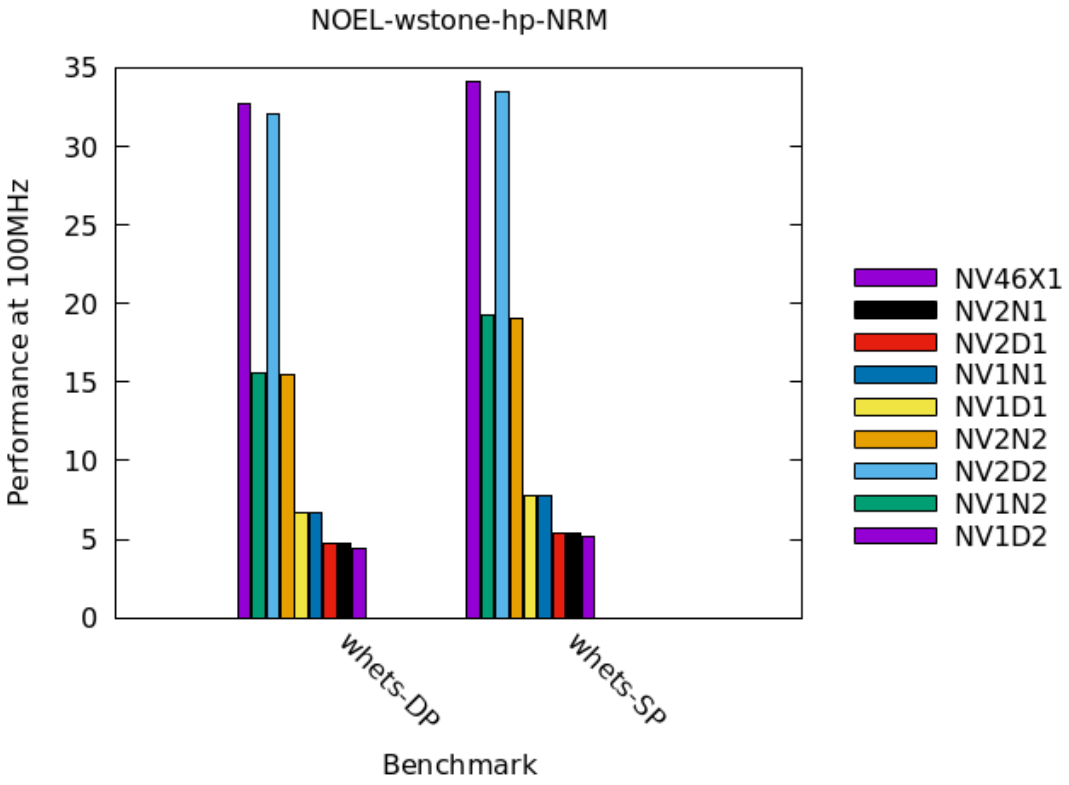

Figure 4: NOEL - Whetstone performance for NOEL targets, GRLIB 2021.2, MWIPS at 100MHz. Higher values denote better performance. Configuration names `NVxD2` denote execution using *daiFPUrv*, `NVxN2` denote *nanoFPU*, while names ending with `1` denote execution using the *SoftFloat* library.

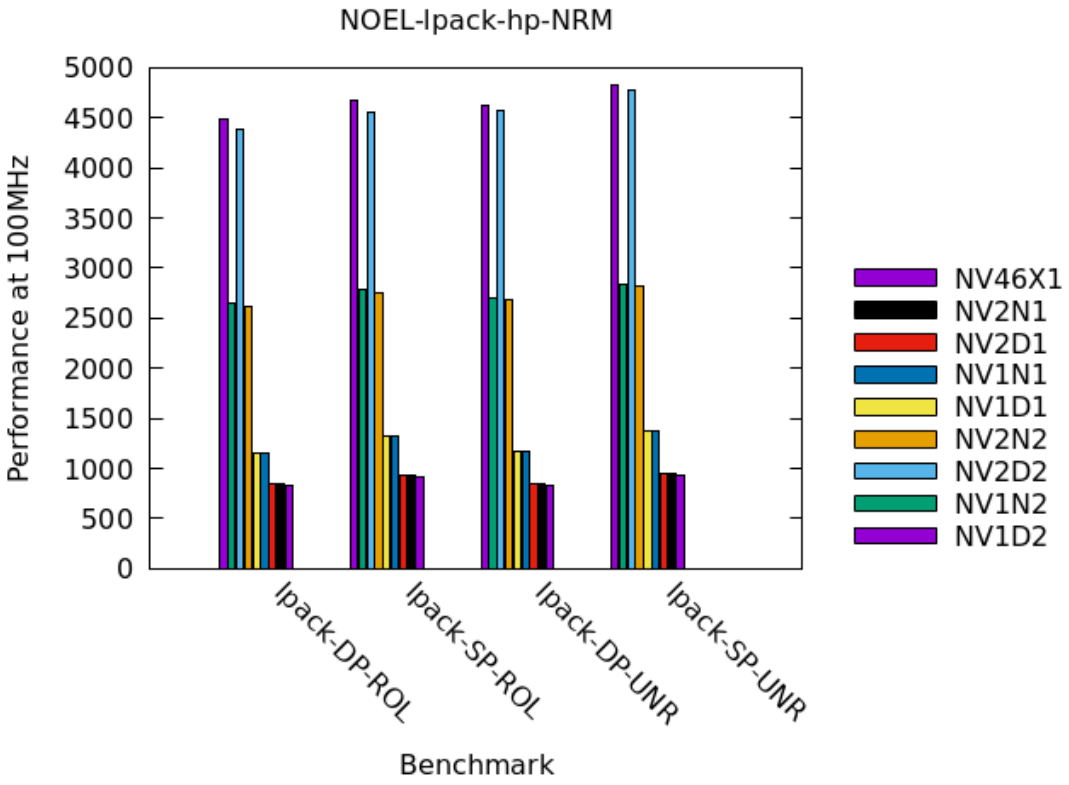

Figure 5: NOEL - Linpack performance for NOEL targets, GRLIB 2021.2, Kflops at 100MHz. Higher values denote better performance. Configuration names `NVxD2` denote execution using *daiFPUrv*, `NVxN2` denote *nanoFPU*, while names ending with `1` denote execution using the *SoftFloat* library.

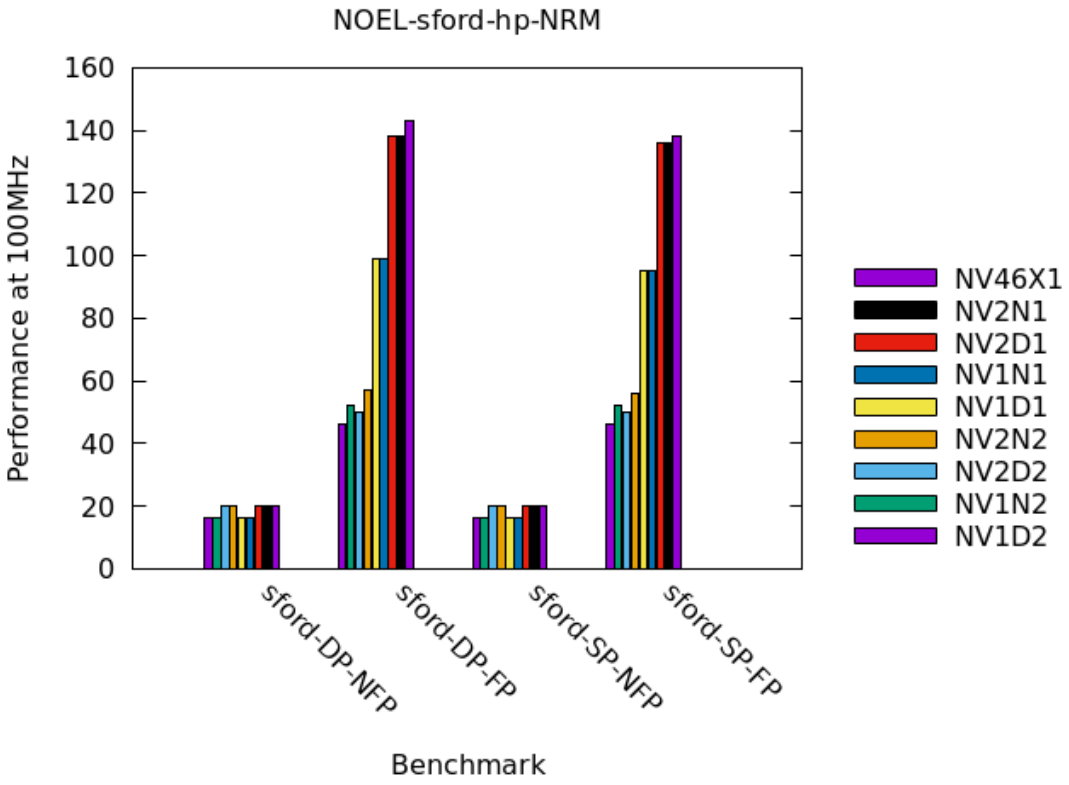

Figure 6: NOEL - Stanford performance for NOEL targets, GRLIB 2021.2, milliseconds at 100MHz. Lower values denote better performance. Configuration names `NVxD2` denote execution using *daiFPUrv*, `NVxN2` denote *nanoFPU*, while names ending with `1` denote execution using the *SoftFloat* library.

8.1.6 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 a factor of *2*.

Furthermore, using the *daiFPUrv* instead of *nanofpunv* increases the performance by a factor of *2* for double-precision configuration, and by a factor of *1.75* for single-precision configurations, as is evident from results for configurations *NV1D2*, *NV1N2* and *NV2D2* and *NV2N2*.

There is almost no difference in the single-core performance between the NOEL-V configurations that use *nanoFPU* - *NV02* to *NV32*. Configurations that use *daiFPUrv* have about 2x the performance than configurations with *nanoFPU*.

The lack of caches in *NV4* 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.

GRLIB 2020.4: When using *SoftFloat* to emulate floating-point operations, the NOEL-V configurations can be divided into the following performance groups:

- high performance - *NV0*, *NV1* with almost the same performance,
- medium performance - *NV2*, *NV3*, *NV4*, with *NV22* having a slightly higher performance than the other two in the group,
- very low performance - *NV5*, with about a 5x lower performance than *NV0/NV1* and 3x lower performance than *NV2/NV3/NV4*.

GRLIB 2021.2: The configuration HPP is the same as GPP-double issue. Apart from this, the conclusions drawn for the GRLIB 2020.4 version are valid for this version too.

daiFPUrv significantly increases the floating-point performance of the NOEL-V core, although its performance is slightly lagging behind LEON2FT with daiFPU.

8.2 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[1] , 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.

8.2.1 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
```

[1] 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.

8.2.2 Measurements for GRLIB 2020.4

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 27 for floating-point execution with binaries compiled for the *noel64imafd_smp* BSP, and Table 28 for integer execution with binaries compiled for the *noel64ima_smp*. Note that only the four configurations are listed in Table 27 that use the *nanoFPU*.

Table 27: NOEL-V scalability - CoreMark, floating-point version, iterations/second at 100MHz for noel64imafd_smp (GRLIB 2020.4) w/o L2Cache.

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 28: NOEL-V scalability - CoreMark, integer version, iterations/second at 100MHz for noel64ima_smp (GRLIB 2020.4) w/o L2Cache.

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 7 and 8. The figures show the *CoreMark* iterations per second at 100MHz. Higher values denote better processor performance.

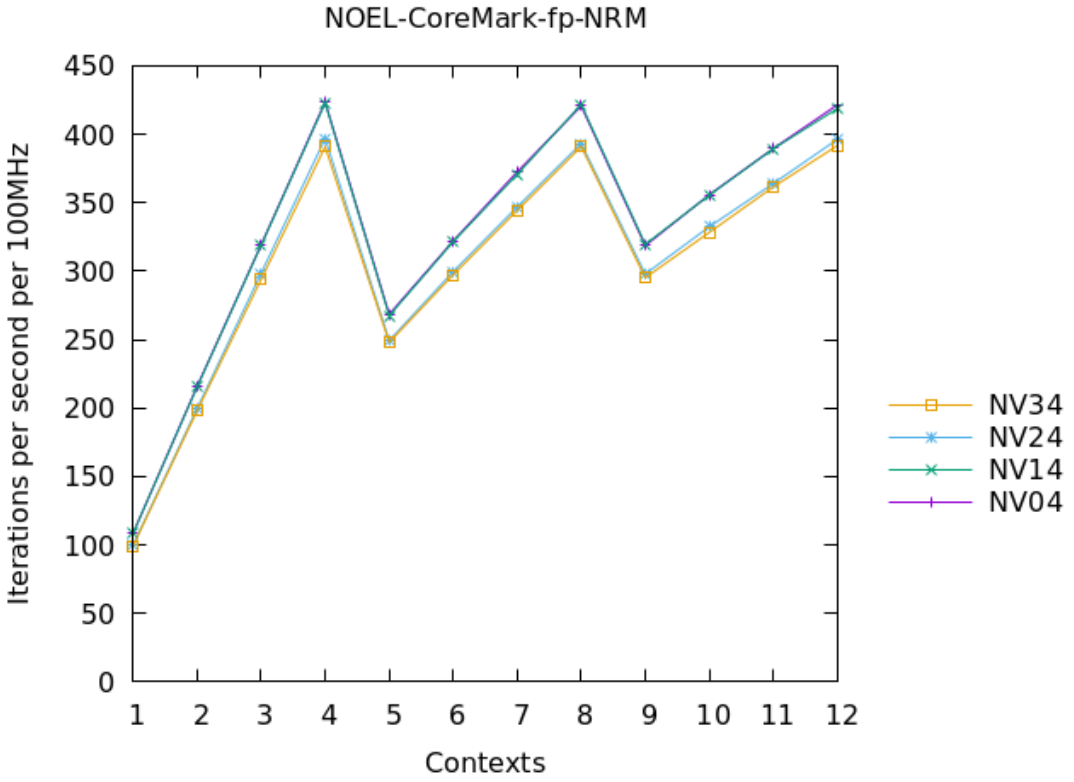

Figure 7: NOEL - CoreMark - floating-point performance for NOEL targets (GRLIB 2020.4) w/o L2Cache, CoreMark iterations at 100MHz. Higher values denote better performance.

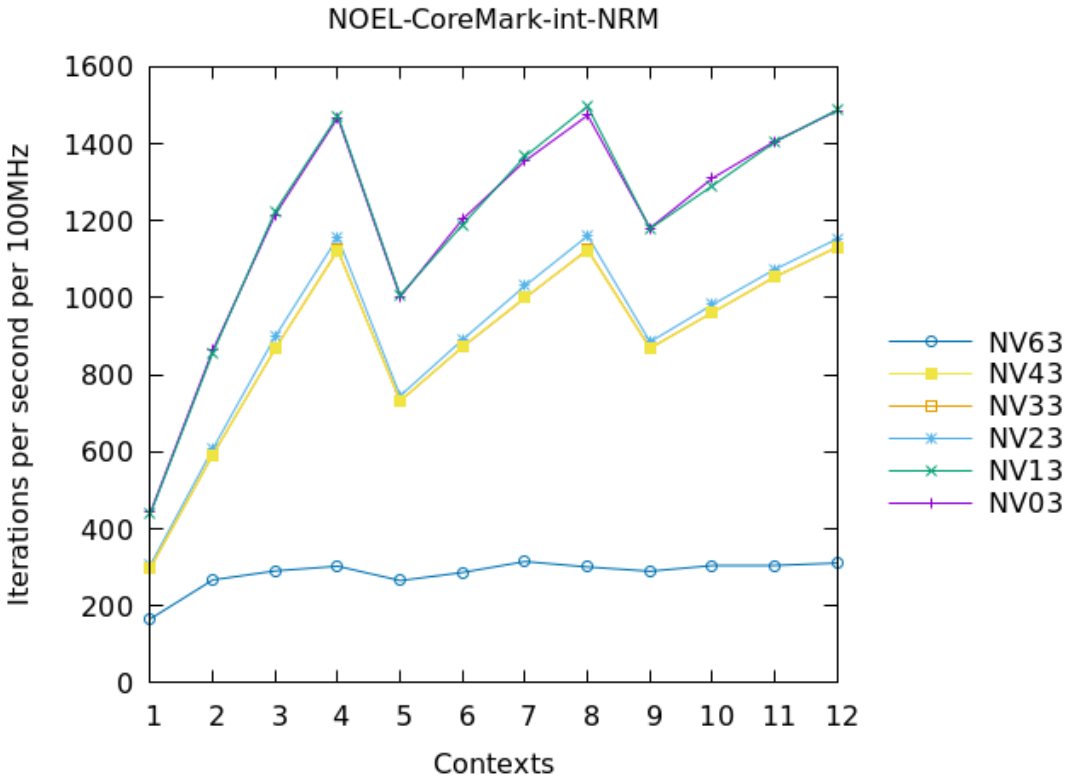

Figure 8: NOEL - CoreMark - integer performance for NOEL targets (GRLIB 2020.4) w/o L2Cache, CoreMark iterations at 100MHz. Higher values denote better performance.

8.2.3 Analysis for GRLIB 2020.4

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 all performance plots shown in Figures 7 to 10 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[2] . 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 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. 7 indicates that the floating-point performance of dual-issue NOEL-V configurations *NV04* and *NV14* is nearly the same irrespective of the smaller *branch target buffer* used in *NV14*. 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 *NV34*. The actual performance difference between NOEL-V configurations *CFG0 (HPP)*, *CFG1 (GPP-DI)*, *CFG2 (GPP-SI)* and *CFG3* is very small.

The results of the *CoreMark* integer execution shown in Fig. 8 confirm the previous observations from the *CoreMark* floating-point execution, but the performance difference between *NV03/NV13* and *NV23/NV33/NV43* is bigger. The performance of the *TINY* NOEL-V configuration *CFG6* (GRLIB 2020.4) is very low, and its performance beyond 2 contexts is almost constant.

8.2.4 Measurements for GRLIB 2021.2

This section extends the previous performance measurements to NOEL-V configurations that use *daiFPUrv*. In addition, all configurations used L2Cache; due to the nearly identical nature of the NOEL-V configurations with *nanoFPU* in GRLIB 2020.4 and 2021.2 the influence of L2Cache on performance can be also evaluated.

Table 29: NOEL-V scalability - CoreMark, floating-point version, iterations/second at 100MHz for noel64imafd_smp (GRLIB 2021.2) w/ L2Cache.

THREADS	NV1D4	NV1N4	NV2D43	NV2N43	NV2D44	NV2N44
1	142.47	108.52	127.72	99.53	127.71	99.53
2	284.96	216.26	254.65	198.49	255.11	198.85
3	427.16	324.15	381.71	297.56	381.71	297.58
4	285.68	217.01	255.33	198.97	508.22	396.29
5	356.27	270.66	318.40	248.19	319.12	248.72
6	427.04	324.57	381.81	296.19	382.80	298.37
7	333.64	253.31	298.02	232.24	445.59	347.34
8	379.83	288.61	339.67	264.76	508.17	396.37
9	427.19	324.62	381.75	297.65	383.01	298.51
10	357.45	271.38	319.34	248.85	425.54	331.66
11	392.32	297.78	350.48	273.16	466.97	364.02
12	425.71	324.69	381.88	296.72	508.54	396.59

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

Table 30: NOEL-V scalability - CoreMark, integer version, iterations/second at 100MHz for noel64ima_smp (GRLIB 2021.2) w/ L2Cache.

THREADS	NV1N3	NV2N33	NV2N34	NV46X2	NV47X2
1	440.61	301.15	301.32	284.24	271.56
2	879.90	601.16	602.43	567.22	536.65
3	1318.91	901.63	901.87	847.62	788.71
4	882.37	602.39	1201.12	1123.45	1026.14
5	1100.71	751.71	753.30	1406.71	1260.27
6	1319.08	899.26	902.84	827.96	788.70
7	1028.82	703.34	1051.64	989.14	911.12
8	1174.44	803.12	1202.96	1127.51	1027.79
9	1320.70	902.14	904.78	1267.23	1146.78
10	1103.50	753.32	1004.82	1404.23	1257.42
11	1210.56	826.82	1103.36	1040.35	951.98
12	1320.40	896.87	1203.23	1131.27	1032.35

The values in the tables are also shown in Figures 9 and 10. The figures show the *CoreMark* iterations per second at 100MHz. Higher values denote better processor performance.

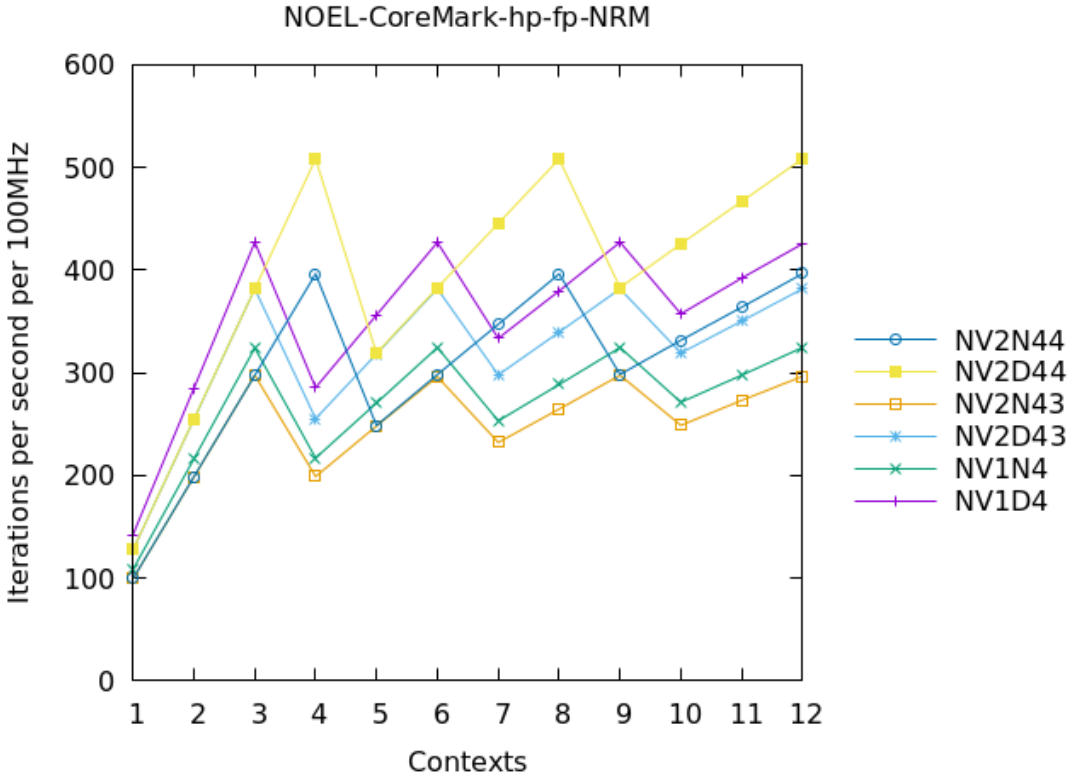

Figure 9: NOEL - CoreMark - floating-point performance for NOEL targets (GRLIB 2021.2) w/ L2Cache, CoreMark iterations at 100MHz. Higher values denote better performance.

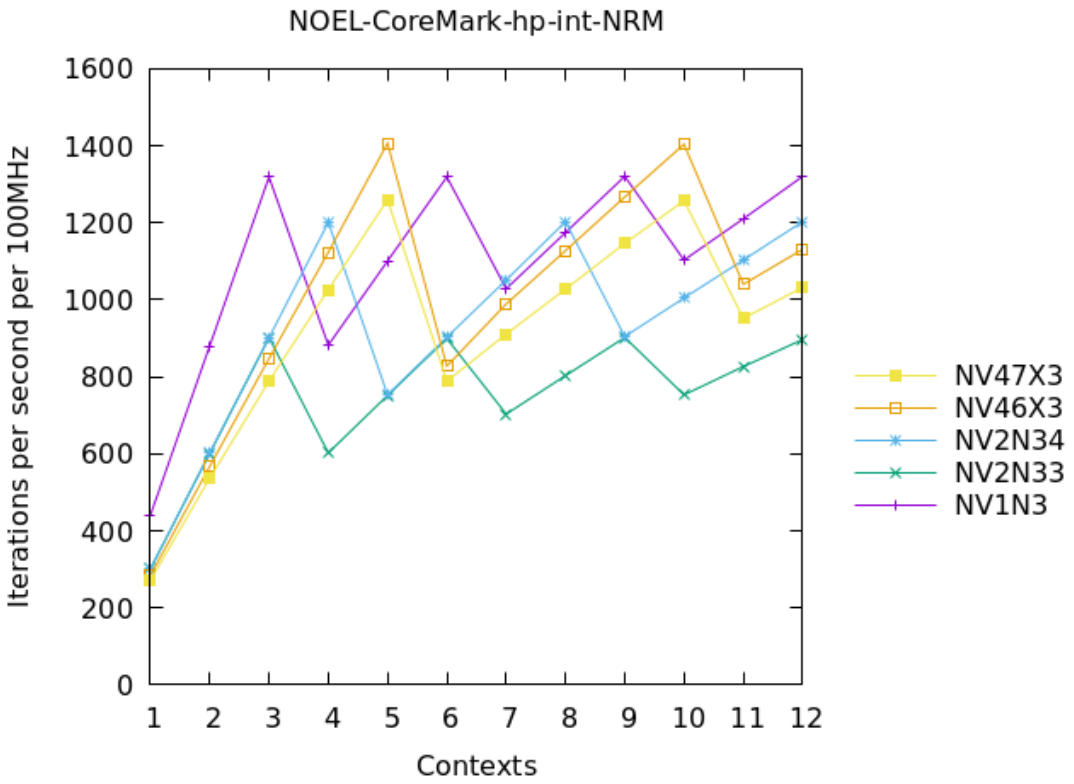

Figure 10: NOEL - CoreMark - integer performance for NOEL targets (GRLIB 2021.2) w/ L2Cache, CoreMark iterations at 100MHz. Higher values denote better performance.

8.2.5 Analysis for GRLIB 2021.2

The *CoreMark* measurements complement the implementation experiments discussed in Section 6 such that the configuration with the best performance can be determined before proceeding to measure performance for more complex and time consuming benchmarks.

The primary question is which configuration to select. The results in Section 6 show that when using *daiFPUrv* only three processor cores can fit in the XCKU040 device for the GPP dual-issue configuration that is expected to have the highest performance, but when using GPP single-issue four processor cores with *daiFPUrv* fit in the device. It is evident namely from Fig. 9 that the most efficient use of the used silicon presents the NOEL-V configuration *GPP single-issu with daiFPUrv* since it achieves the highest *CoreMark* performance, or in other words four GPP-SI cores w/ daiFPUrv are better than three GPP-DI cores w/ daiFPUrv.

Another interesting point is comparing the performance of *NV2D44* and *NV2N44*. Although *daiFPUrv* was shown to have more than twice the performance over *nanoFPU* in the PWLS benchmarks (see for example Fig. 4), for more complex workloads that have a higher portion of non-floating point instructions it is not so; for *CoreMark* the performance advantage of *daiFPUrv* over *nanoFPU* is about 1.3x for GPP dual issue, and 1.28x for GPP single issue (Table 29).

Finally, the performance measured for *NV14*, *NV24* with *NV1N4* and *NV2N44* is almost the same, thus the performance advantage of using L2Cache for *CoreMark* is not obvious, which is in accordance with the observation made in Section 8.2.3 that *CoreMark* is compute-bound.

Based on the above facts the NOEL-V configuration *NV2D44* will be used for measuring NOEL-V performance in the *CoreMark-Pro* and *FPMark* benchmarks.

8.3 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.

8.3.1 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 31 together with abbreviated names that are used in the tables with the measured data[3] . Table 32 specifies for each workload the number of so-called iterations[4] 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)[5] .

Table 31: 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

[3] 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]).

[4] *CoreMark-Pro* 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.

[5] 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 32. 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 32: 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

8.3.2 Worst-case execution time

An approximate worst-case execution time for the *CoreMark-Pro* suite executed in NOEL-V is shown in Table 33.

Table 33: NOEL-V - worst-case execution times.

Configuration	NV04	NV14	NV24	NV34	NV43	NV63
Frequency [MHz]	100	100	100	100	100	100
Execution time [hrs]	2	2	2	2	6	6

8.3.3 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
```

8.3.4 Measurements for GRLIB 2020.4

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

Table 34: 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 35: 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 36: 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 37: 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 38: 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 34 to 38 are plotted in Figures 11 to 19.

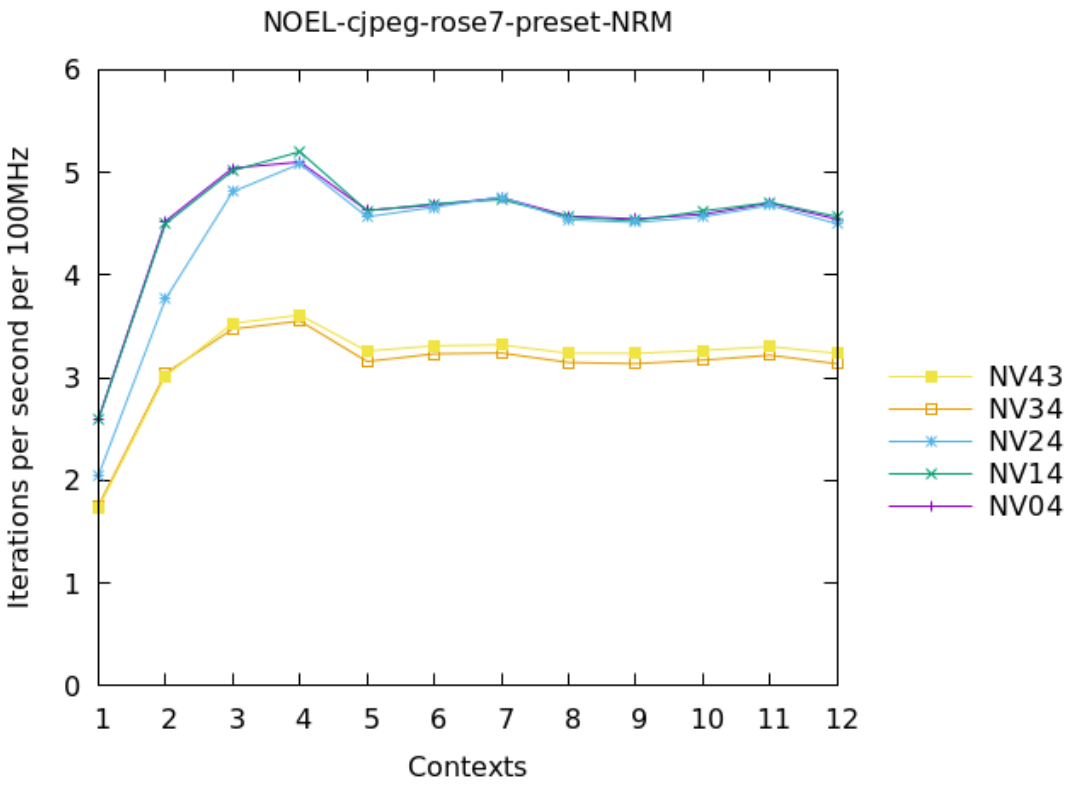

Figure 11: NOEL - CoreMark-Pro - cjpeg, iterations per second at 100MHz. Higher values denote better performance.

8.3.5 Analysis - GRLIB 2020.4

Observations based on Figures 11 to 19 are as follows:

- Branch target prediction: There is no observable performance difference between *NV04* and *NV14*.
- Dual vs. single issue: The performance of *NV24* is almost identical to *NV04* and *NV14* with the exception of *core*.
- Cache size: *NV34* achieves a significantly lower performance in *cjpeg*, *parser*, *sha*, *zip*.
- FPU performance: There is almost no performance difference between *NV04*, *NV14*, *NV24* and *NV34* in *linear*, *loops*, *nnet*, *radix2*.
- *nanoFPU* vs *SoftFloat*: Tables 37 and 38 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 *NV43* is about 4x lower than the *nanoFPU* performance in *NV34*.

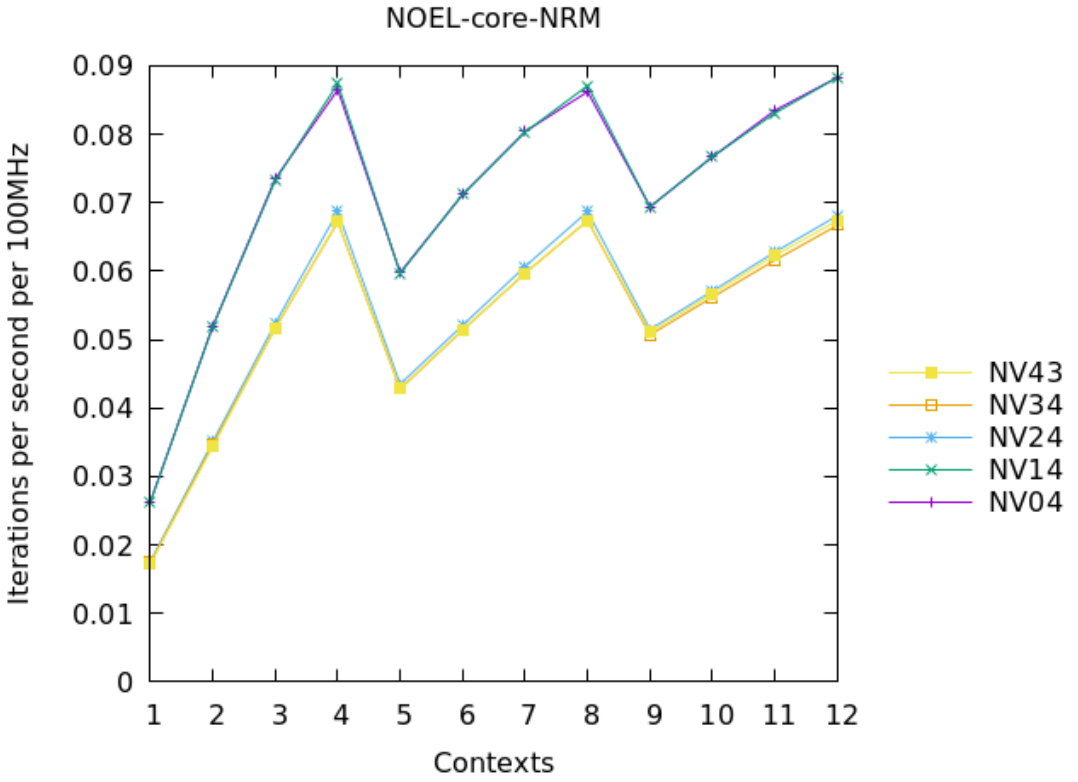

Figure 12: NOEL - CoreMark-Pro - core, iterations per second at 100MHz. Higher values denote better performance.

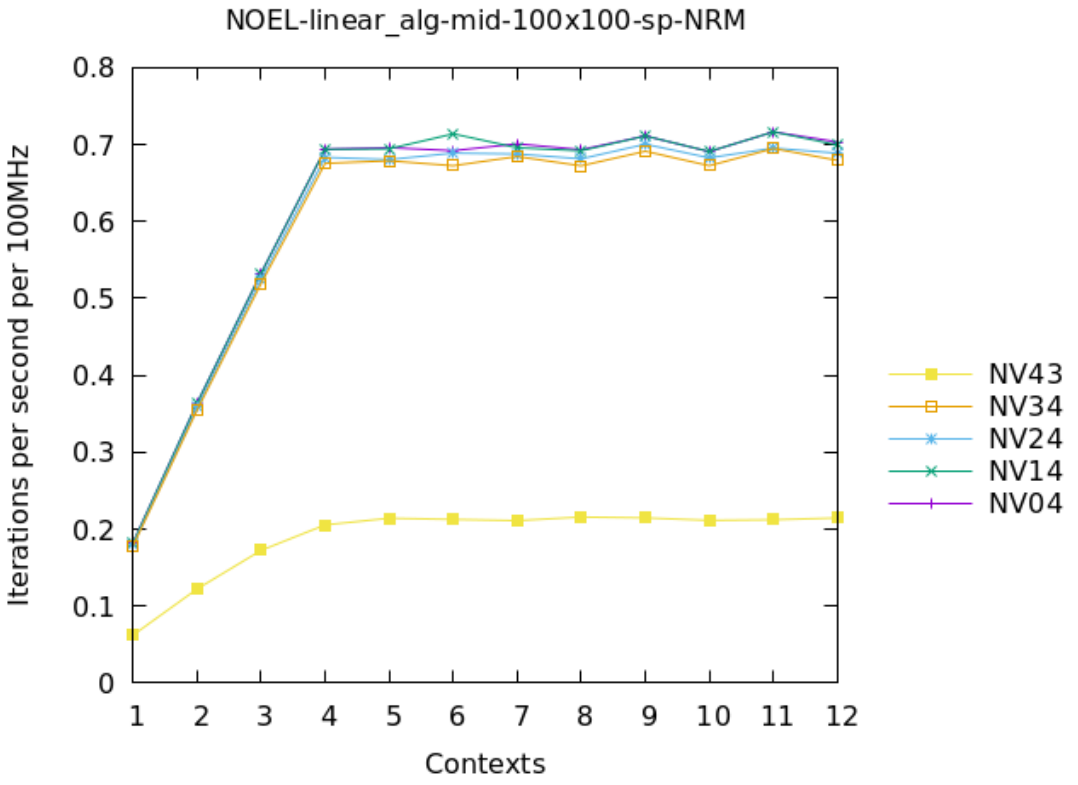

Figure 13: NOEL - CoreMark-Pro - linear, iterations per second at 100MHz. Higher values denote better performance.

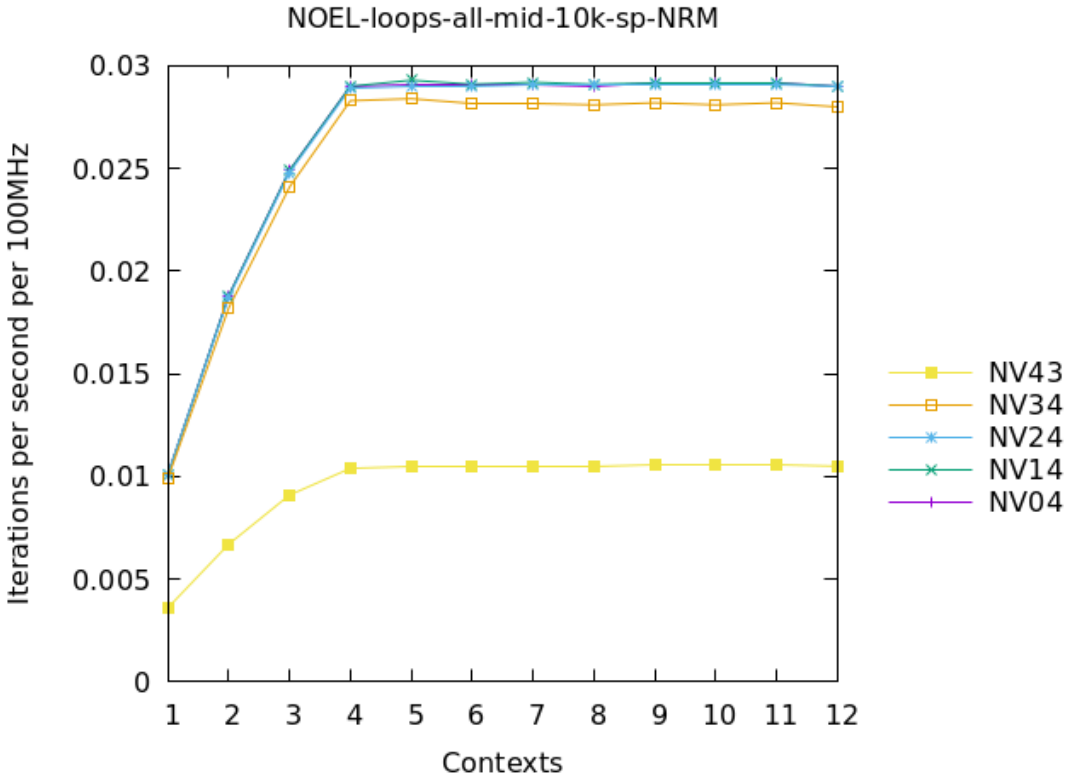

Figure 14: NOEL - CoreMark-Pro - loops, iterations per second at 100MHz. Higher values denote better performance.

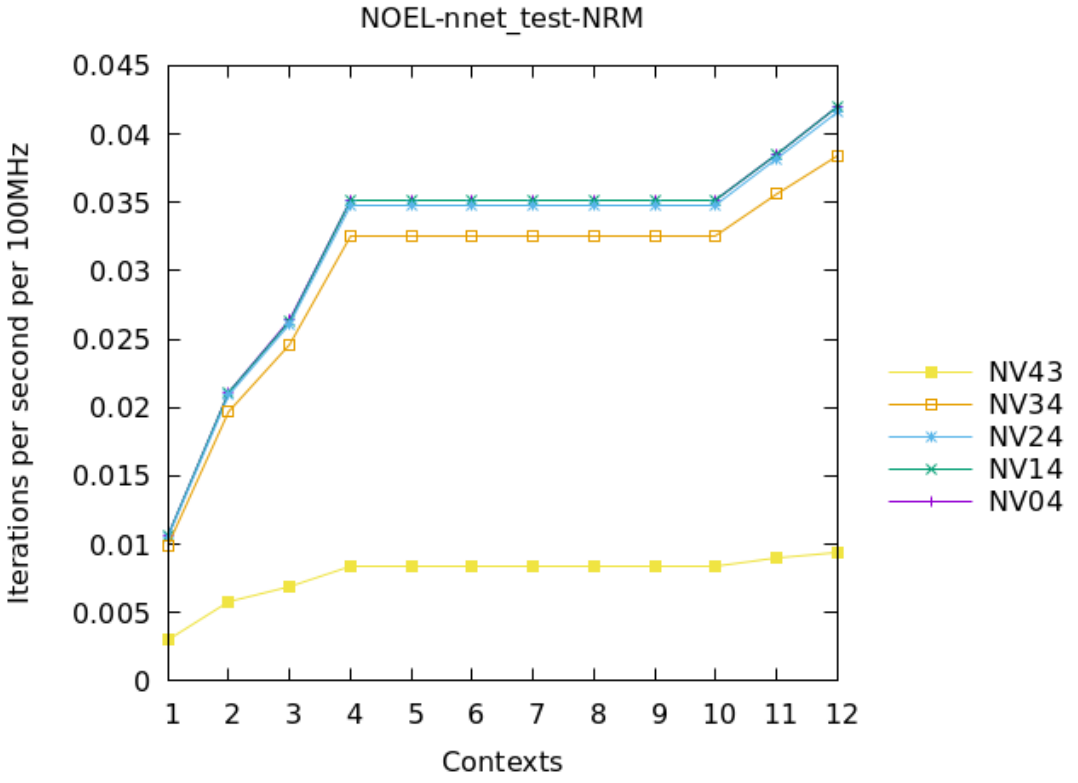

Figure 15: NOEL - CoreMark-Pro - nnet, iterations per second at 100MHz. Higher values denote better performance.

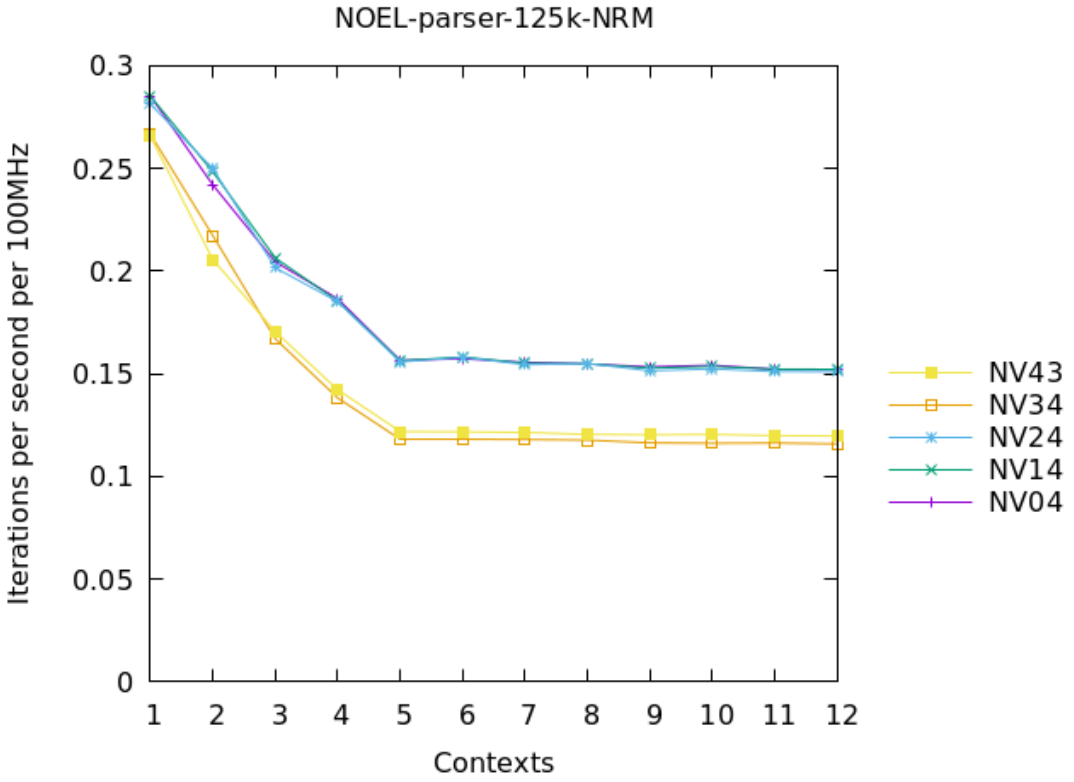

Figure 16: NOEL - CoreMark-Pro - parser, iterations per second at 100MHz. Higher values denote better performance.

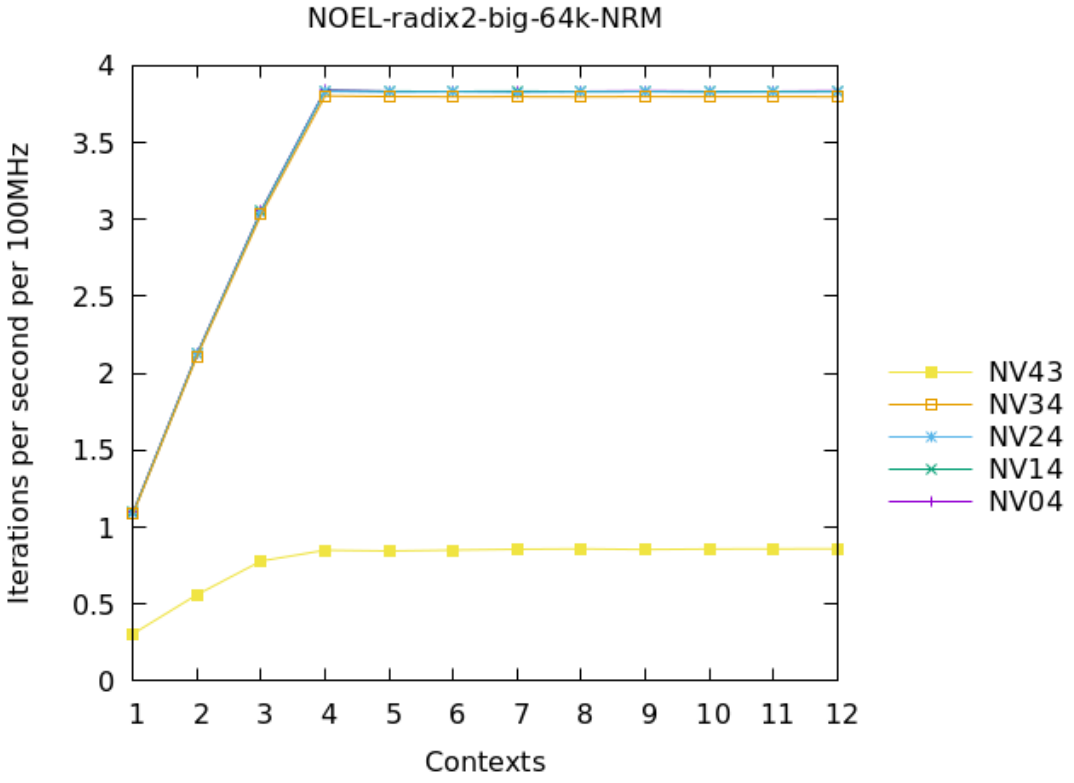

Figure 17: NOEL - CoreMark-Pro - radix2, iterations per second at 100MHz. Higher values denote better performance.

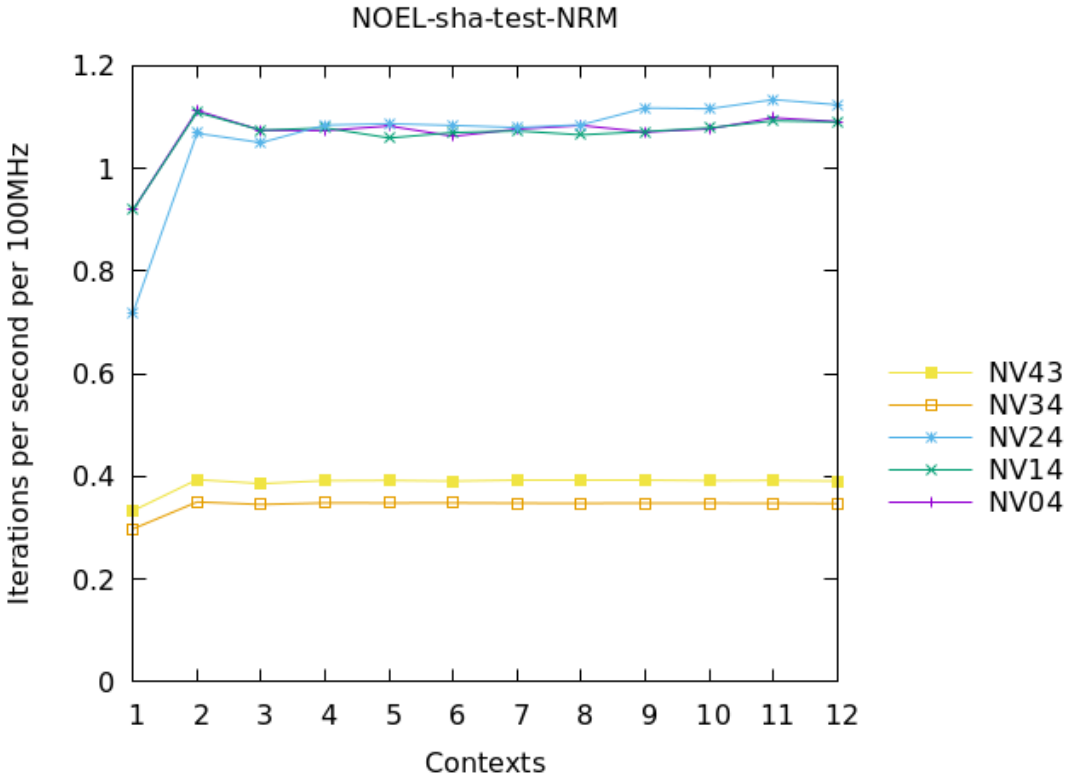

Figure 18: NOEL - CoreMark-Pro - sha, iterations per second at 100MHz. Higher values denote better performance.

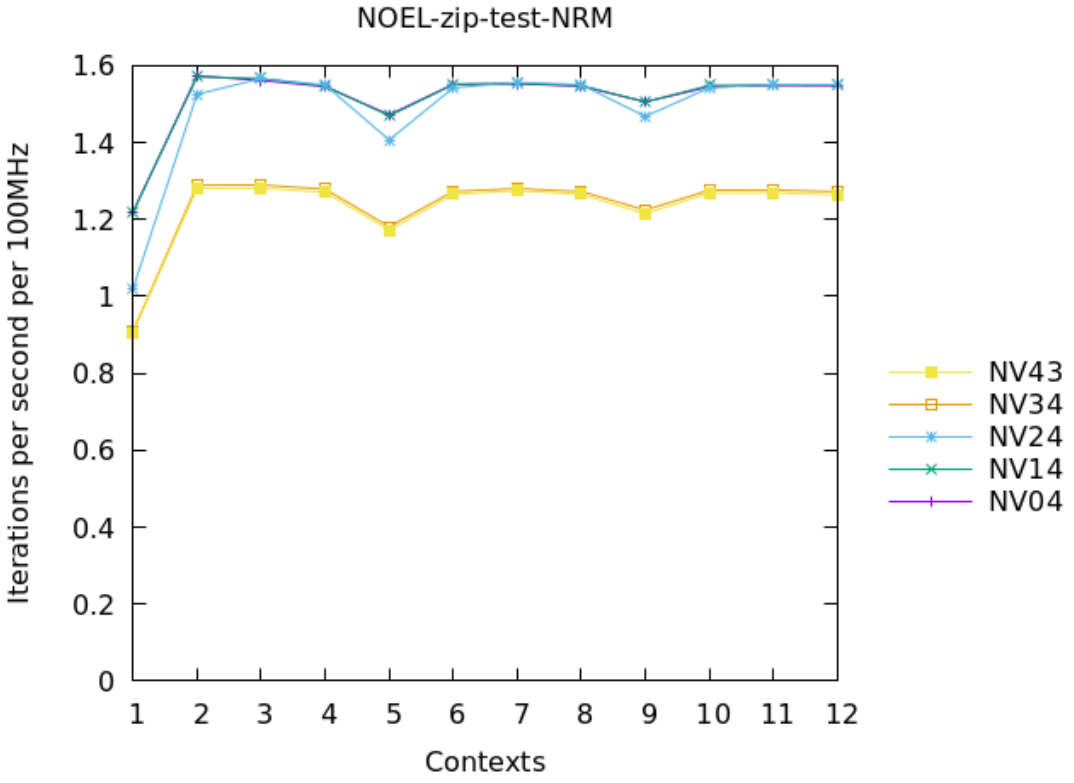

Figure 19: NOEL - CoreMark-Pro - zip, iterations per second at 100MHz. Higher values denote better performance.

8.3.6 Measurements for GRLIB 2021.2

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

Table 39: NOEL-V - CoreMark-Pro results for configuration NV1D4 - workload iterations per second at 85MHz. *-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	2.3918	4.6382	6.6138	6.4935	6.5062	6.1162	6.3131	6.4392	6.0864	6.3012	6.3898	6.0716
core	0.0225	0.0448	0.0673	0.0449	0.0562	0.0673	0.0524	0.0599	0.0673	0.0562	0.0618	0.0673
linear	0.2388	0.4755	0.7008	0.7007	0.7065	0.7006	0.7027	0.7044	0.7067	0.7046	0.6987	0.7026
loops	0.0132	0.0256	0.0367	0.0368	0.0367	0.0367	0.0367	0.0367	0.0367	0.0368	0.0368	0.0369
nnet	0.0143	0.0285	0.0358	0.0358	0.0358	0.0358	0.0358	0.0358	0.0358	0.0358	0.0393	0.0428
parser	0.4137	0.4300	0.3130	0.2486	0.2426	0.2365	0.2327	0.2294	0.2269	0.2261	0.2246	0.2221
radix	1.2955	2.4727	3.5032	3.5019	3.4980	3.5018	3.5014	3.5014	3.5007	3.5012	3.5039	3.5005
sha	1.2930	2.3480	2.7012	2.7027	2.7042	2.7034	2.7042	2.7034	2.7042	2.7020	2.9170	3.0628
zip	1.3055	2.1277	2.6549	2.1482	2.4913	2.6774	2.3641	2.5674	2.6762	2.4691	2.6234	2.6936

Table 40: NOEL-V - CoreMark-Pro results for configuration NV2D44 - workload iterations per second at 112MHz. *-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	2.4260	4.7259	6.8166	8.8183	8.3264	8.6059	8.7336	8.1566	8.2237	8.5324	8.5985	8.1566
core	0.0194	0.0389	0.0583	0.0776	0.0486	0.0583	0.0680	0.0776	0.0583	0.0648	0.0713	0.0776
linear	0.3073	0.6138	0.9020	1.1789	1.2189	1.1930	1.1798	1.1743	1.1826	1.1828	1.2139	1.1785
loops	0.0170	0.0331	0.0472	0.0588	0.0588	0.0587	0.0592	0.0592	0.0591	0.0589	0.0589	0.0589
nnet	0.0186	0.0371	0.0464	0.0618	0.0618	0.0618	0.0618	0.0618	0.0618	0.0618	0.0680	0.0740
parser	0.5247	0.5269	0.3985	0.3207	0.2851	0.2811	0.2770	0.2704	0.2688	0.2669	0.2661	0.2636
radix	1.6361	3.1437	4.3195	4.9403	4.9377	4.9368	4.9373	4.9341	4.9334	4.9396	4.9372	4.9365
sha	1.1489	2.2272	2.6781	3.3659	3.3670	3.3535	3.3636	3.3693	3.3647	3.3568	3.6364	3.8278
zip	1.3351	2.2805	2.9240	3.2653	2.5694	2.9441	3.1847	3.3501	2.8690	3.0931	3.2277	3.3529

8.3.7 Measurements for PolarFire SoC (linux)

To put the NOEL-V measurements in the context of available commercial offerings we have also included performance data for the RISC-V multi-core processor used in the PolarFire SoC FPGA.

Table 41: PFS - CoreMark-Pro results for PolarFire SoC RISC-V - workload iterations per second at 667MHz. *-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	12.2549	24.5098	35.4610	39.6825	38.3142	46.2963	46.9484	45.2489	46.0829	46.0829	45.8716	45.6621
core	0.0972	0.1943	0.2913	0.3847	0.3788	0.3823	0.3832	0.3853	0.3827	0.3850	0.3845	0.3854
linear	5.1477	10.2838	15.1561	19.8020	19.8886	20.0401	20.0803	19.7239	19.9362	19.5008	20.2840	19.7161
loops	0.1997	0.3644	0.5016	0.6112	0.6243	0.6037	0.5898	0.5876	0.6015	0.5982	0.5971	0.5795
nnet	0.2809	0.5635	0.7042	0.9391	1.1212	1.1033	1.0151	0.9351	0.8658	1.1000	1.1084	1.1136
parser	2.3310	4.4743	5.8708	6.9085	6.2344	6.0914	6.3120	6.1824	6.0565	6.1013	6.0076	6.0211
radix	19.2348	23.0028	31.0936	31.2969	32.1978	31.7370	30.7163	29.6130	30.3416	30.0273	29.9097	29.4308
sha	7.8802	15.6986	19.5695	26.1097	26.8097	27.4725	26.3158	26.1780	29.0698	29.8507	30.1370	31.2500
zip	5.8824	11.4286	16.5746	20.8333	16.8350	18.2927	17.1990	18.3066	18.0723	18.7970	18.6125	18.6047

The values shown in Tables 39 to 41 are plotted in Figures 20 to 28.

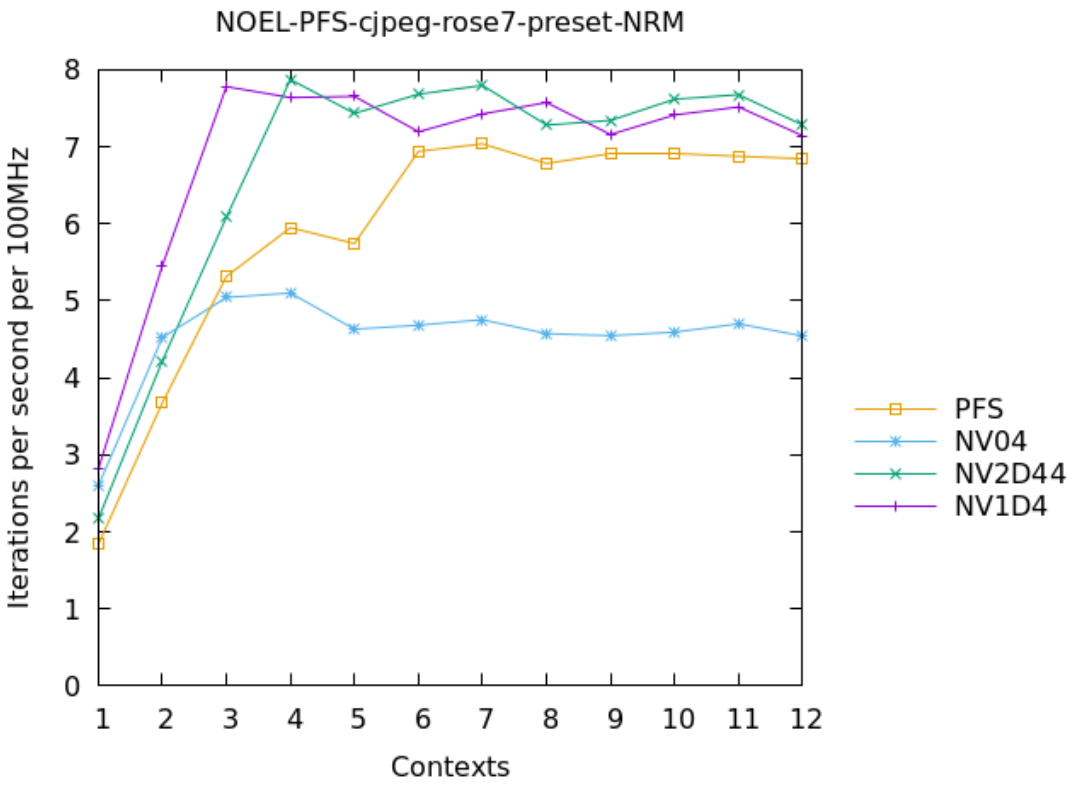

Figure 20: NOEL - CoreMark-Pro - cjpeg, iterations per second at 100MHz. Higher values denote better performance.

8.3.8 Analysis - GRLIB 2021.2

The high-performance configurations of interest are:

- *NV04* - NOEL-V HPP, 4 cores, 2 issues per core, 1 nanoFPU per core. In total: 8 (integer) issues, 4 nanoFPUs.
- *NV1D4* - NOEL-V GPP, 3 cores, 2 issues per core, 1 daiFPUrv per core. In total: 6 (integer) issues, 3 daiFPUrvs.
- *NV2D44* - NOEL-V GPP, 4 cores, 1 issue per core, 1 daiFPUrv per core. In total: 4 (integer) issues, 4 daiFPUrvs.
- *PFS* - a (4+1)-core RISC-V system used in PolarFire SoC FPGAs.

Observations for these configurations, based on Figures 20 to 28, are as follows:

- As can be seen from the figures, the *core* workload is very specific in that is is strictly compute-bound already for *NV01* that does not use any L2Cache. The highest performance for *core* is achieved for configuration *NV04* that uses 4 HPP cores, but when comparing performance for 3 contexts between *NV04* and *NV1D4*, *NV1D4* is slightly better. This is due to two factors. First, as was mentioned in the previous section, there is a slight performance difference between *NV04* (HPP) and *NV14* (GPP dual-issue) on one side, and *NV24* (GPP single-issue) on the other side, in favour of *NV04* and *NV14*. Second, *NV1D4* uses an L2Cache that increases its average performance bandwidth.

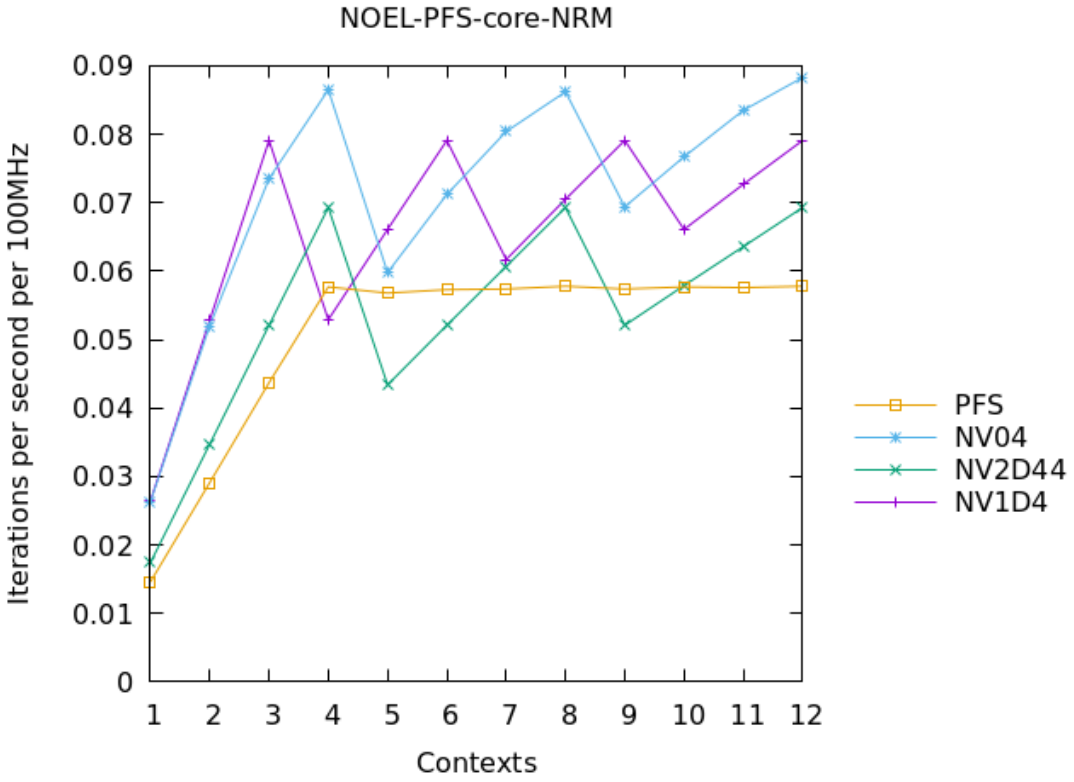

Figure 21: NOEL - CoreMark-Pro - core, iterations per second at 100MHz. Higher values denote better performance.

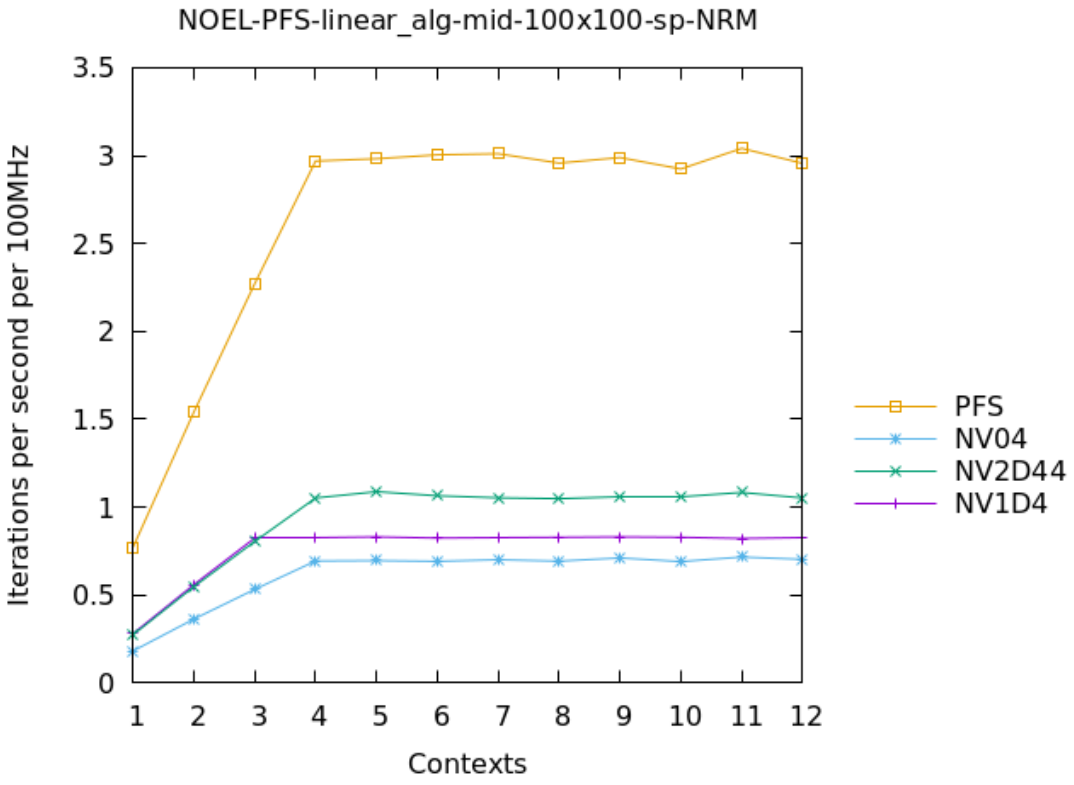

Figure 22: NOEL - CoreMark-Pro - linear, iterations per second at 100MHz. Higher values denote better performance.

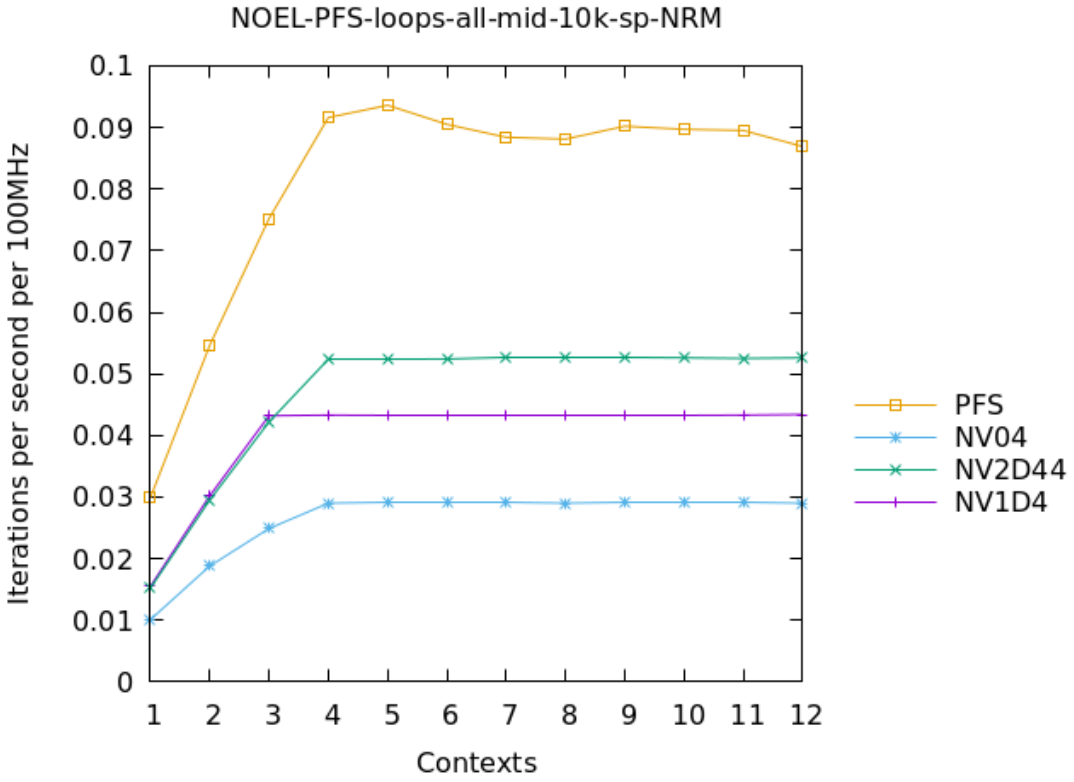

Figure 23: NOEL - CoreMark-Pro - loops, iterations per second at 100MHz. Higher values denote better performance.

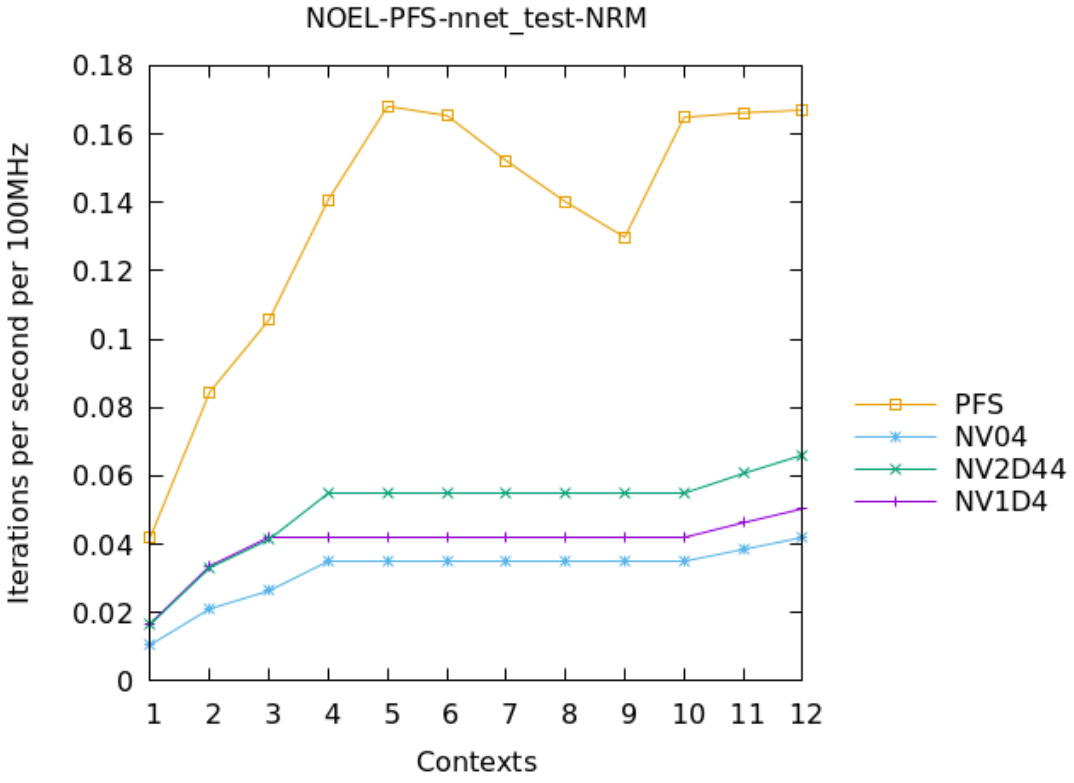

Figure 24: NOEL - CoreMark-Pro - nnet, iterations per second at 100MHz. Higher values denote better performance.

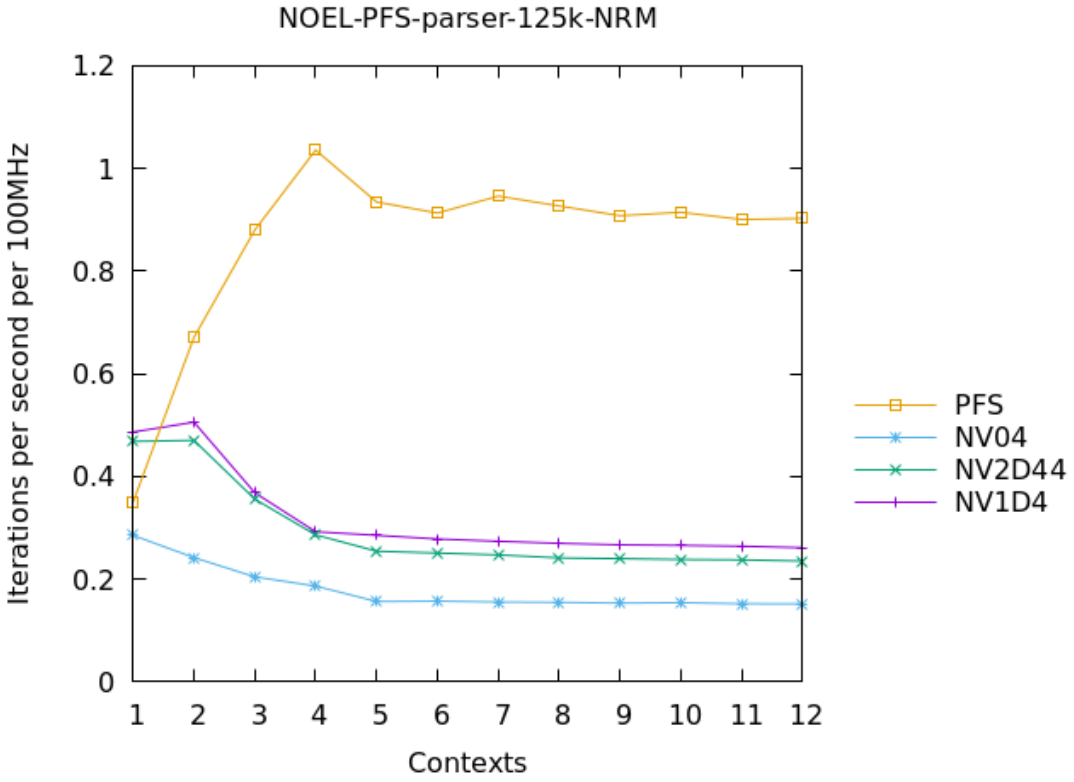

Figure 25: NOEL - CoreMark-Pro - parser, iterations per second at 100MHz. Higher values denote better performance.

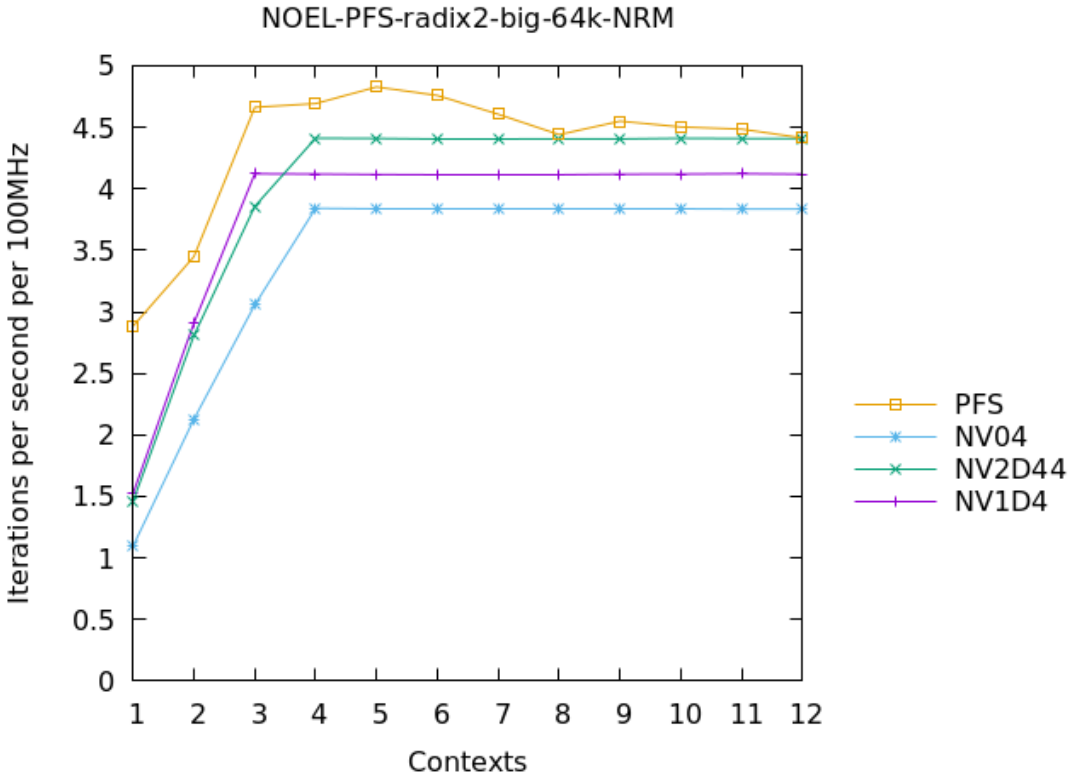

Figure 26: NOEL - CoreMark-Pro - radix2, iterations per second at 100MHz. Higher values denote better performance.

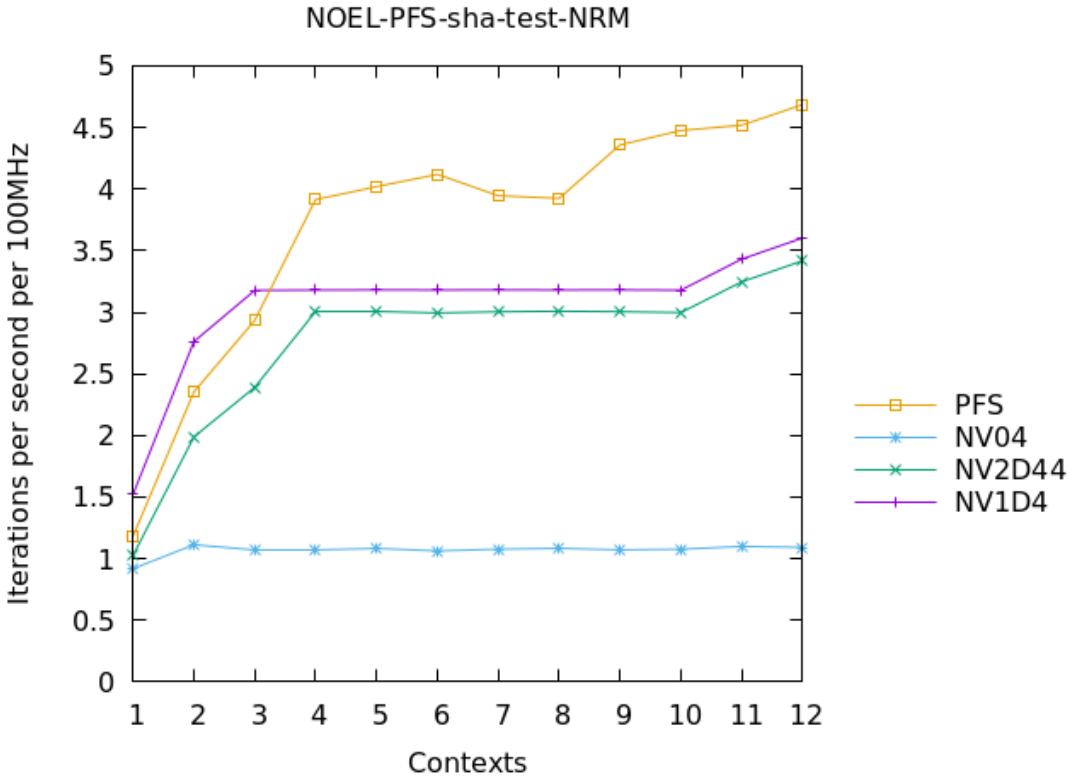

Figure 27: NOEL - CoreMark-Pro - sha, iterations per second at 100MHz. Higher values denote better performance.

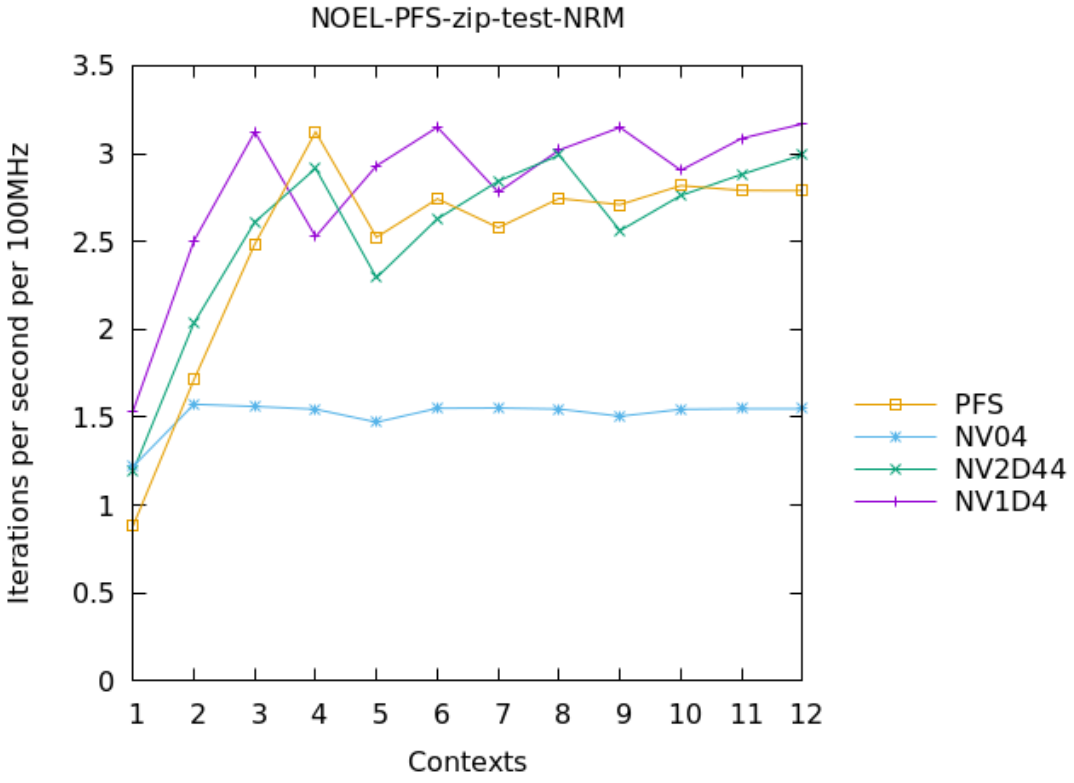

Figure 28: NOEL - CoreMark-Pro - zip, iterations per second at 100MHz. Higher values denote better performance.

- The performance advantage of the dual-issue configuration *NV1D4* for integer workloads is clearly seen is Figures 21 and 27, while for the rest of the workloads there is almost no difference.
- The advantage of the L2Cache is clearly seen for memory-bound workloads, that is all but *core* (see Fig. 21). Fig. 28 nicely demonstrates how the L2Cache helped move the workload from the memory-bound region to the compute-bound region (compare *NV04* that does not use the L2Cache with the other two configurations that use the L2Cache).
- *NV2D44* shows the best performance for the floating-point workloads (4 cores, 1 issue per core, 1 FPU per core), while for integer workloads the best performance is achieved with *NV1D4* (3 cores, 2 issues per core, 1 FPU per core). This is mostly driven by the fact that NOEL-V can execute floating-point instructions only in one issue in the dual-issue configurations. The performance of floating-point workloads is determined by the time needed to execute all the floating-point instructions; obviously the time is shorter in configurations that can fit more FPUs in the FPGA device.
- For five CoreMark-Pro workloads (*cjpeg*, *core*, *radix2*, *sha-test*, *zip-test*) the normalized performance of NOEL-V is comparable to the PolarFire SoC RISC-V.

8.4 FPMark

The *FPMark* benchmark suite consists of a mix of floating-point workloads that should provide better insight into the performance and scaling properties of NOEL-V.

8.4.1 Specification of the workloads

FPMark consists of nine workloads that sample different characteristics of parallel execution in multi-core systems. An overview of the benchmark kernels is shown in Table 42 together with abbreviated names that are used in the tables with the measured data[6] .

Note that the *FPMark* scores published in this report are not the certified scores as their measurement would require several order of magnitude longer execution time[8] .

The FPMark workloads consist a mix of benchmark kernels that work with usually three data sets - big, medium and small. This way each kernel samples different properties of a compiler and processor architecture, and each data set size samples different regions of the memory architecture. Citing from [FPM], “Using the same example, a different design that has even performance for all three workloads would get very high scores for the large and small workloads but a middling score for the medium workloads because it is compared to an arbitrary reference with its own idiosyncrasies. However, if the scores were normalized to the workload, such a design should get fairly consistent scores for the different sized workloads.”

[6] For a detailed explanation of the *FPMark* workloads see the *datasheet.txt* files in the *benchmarks* directory in the *FPMark* distribution package (e.g. [FPM]).

[8] 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 43. 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 42: FPMark - kernels (description taken from [FPMark]).

Kernel name	Workload name	Description
ArcTan	atan	Calculates angles of right triangle by using the ratio of two sides of the triangle to calculate the angle between them.
Black-Scholes	blacks	A mathematical model for the dynamics of a financial market containing derivative investment instruments.
Horner's Method	horner	A method to approximate the roots of a polynomial; more information at Wikipedia.
Fast Fourier Transform	radix2	Takes any function and converts it to an equivalent set of sine waves; applications such as audio, spectral analysis, and image compression (computed at radix 2).
Linear Algebra	linear_alg	Derived from Linpack; useful for understanding balancing forces in structural engineering, converting between reference frames in relativity, solving differential equations, and understanding rotation and fluid flow, for example.
Enhanced Livermore Loops	loops, inner-product	This one kernel contains two dozen real-world functions extracted from programs used at Lawrence Livermore Labs. They are used to test the computational capabilities of parallel hardware and cover areas such as 2D Particle-in-Cell, Tri-diagonal Elimnation and Planckian Distribution.
LU Decomposition	lu	Performs lower-upper matrix decomposition.
Neural Net	nnet	A small neural-net inference engine.
Ray-Tracer	ray	A technique for image generation by tracing light path through pixels in an image plane and simulating the effects of its encounters with virtual objects.
Fourier Coefficients	xp1px	Numerical analysis routine for calculating series or representing a periodic function by a discrete sum of complex exponentials, also known as (x+1)^x, defined on the interval [0+epsilon,2-epsilon].

A structure of the instruction mix for each kernel is shown in Fig. 29.

Table 43: FPMark workloads, iterations computed per measurement. *-cx* denotes the number of parallel execution contexts. Values: 1k=1000, 10k=10000, 100k=100000.

.	Iterations											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
atan-1k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k
atan-1k-sp	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k
atan-64k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
atan-64k-sp	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
atan-1M	10	10	10	10	10	10	10	10	10	10	11	12
atan-1M-sp	10	10	10	10	10	10	10	10	10	10	11	12
blacks-big-n5000v200	10	10	10	10	10	10	10	10	10	10	11	12
blacks-big-n5000v200-sp	10	10	10	10	10	10	10	10	10	10	11	12

continues on next page

Table 43 – continued from previous page

.	Iterations											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
blacks-mid-n1000v40	10	10	10	10	10	10	10	10	10	10	11	12
blacks-mid-n1000v40-sp	10	10	10	10	10	10	10	10	10	10	11	12
blacks-sml-n500v20	10	10	10	10	10	10	10	10	10	10	11	12
blacks-sml-n500v20-sp	10	10	10	10	10	10	10	10	10	10	11	12
horner-big-100k	100	100	100	100	100	100	100	100	100	100	100	100
horner-big-100k-sp	100	100	100	100	100	100	100	100	100	100	100	100
horner-mid-10k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
horner-mid-10k-sp	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
horner-sml-1k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k
horner-sml-1k-sp	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k
inner-product-big-100k	20	20	20	20	20	20	20	20	20	20	20	20
inner-product-big-100k-sp	20	20	20	20	20	20	20	20	20	20	20	20
inner-product-mid-10k	400	400	400	400	400	400	400	400	400	400	400	400
inner-product-mid-10k-sp	400	400	400	400	400	400	400	400	400	400	400	400
inner-product-sml-1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
inner-product-sml-1k-sp	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
linear_alg-big-1000x1000	5	5	5	5	5	6	7	8	9	10	11	12
linear_alg-big-1000x1000-sp	5	5	5	5	5	6	7	8	9	10	11	12
linear_alg-mid-100x100	50	50	50	50	50	50	50	50	50	50	50	50
linear_alg-mid-100x100-sp	50	50	50	50	50	50	50	50	50	50	50	50
linear_alg-sml-50x50	500	500	500	500	500	500	500	500	500	500	500	500
linear_alg-sml-50x50-sp	500	500	500	500	500	500	500	500	500	500	500	500
loops-all-big-100k	5	5	5	5	5	6	7	8	9	10	11	12
loops-all-big-100k-sp	5	5	5	5	5	6	7	8	9	10	11	12
loops-all-mid-10k-sp	50	50	50	50	50	50	50	50	50	50	50	50
loops-all-mid-10k	50	50	50	50	50	50	50	50	50	50	50	50
loops-all-tiny	500	500	500	500	500	500	500	500	500	500	500	500
loops-all-tiny-sp	500	500	500	500	500	500	500	500	500	500	500	500
lu-big-2000x2_50	10	10	10	10	10	10	10	10	10	10	11	12
lu-big-2000x2_50-sp	10	10	10	10	10	10	10	10	10	10	11	12
lu-mid-200x2_50	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
lu-mid-200x2_50-sp	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
lu-sml-20x2_50	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k
lu-sml-20x2_50-sp	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k
nnet_data1	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
nnet-data1-sp	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
nnet_test	10	10	10	10	10	10	10	10	10	10	11	12
nnet_test-sp	10	10	10	10	10	10	10	10	10	10	11	12
radix2-big-64k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k
radix2-mid-8k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k	10k
radix2-sml-2k	100k	100k	100k	100k	100k	100k	100k	100k	100k	100k	100k	100k
ray-64x48at4s	50	50	50	50	50	50	50	50	50	50	50	50
ray-320x240at8s	10	10	10	10	10	10	10	10	10	10	11	12
ray-1024x768at24s	2	2	3	4	5	6	7	8	9	10	11	12

continues on next page

Table 43 – continued from previous page

.	Iterations											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
xp1px-big-c10000n2000	10	10	10	10	10	10	10	10	10	10	11	12
xp1px-mid-c1000n200	10	10	10	10	10	10	10	10	10	10	11	12
xp1px-sml-c100n20	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k	1k

8.4.2 Compiler options

The following options were used for compiling the *FPMark* 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
-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
```

8.4.3 Measurements

Tables 44 to 45 show measurements for the following NOEL-V configurations:

- NV04 - NOEL-V HPP (dual-issue), 4 cores, 4 nanoFPUs, GRLIB 2020.4
- NV2D44 - NOEL-V GPP single-issues, 4 cores, 4 daiFPUrvs, GRLIB 2021.2

In addition, the measurements are compared to the multi-core RISC-V architecture implemented inside the PolarFire SoC device

- PFS - RISC-V in PolarFire SoC [PFS].

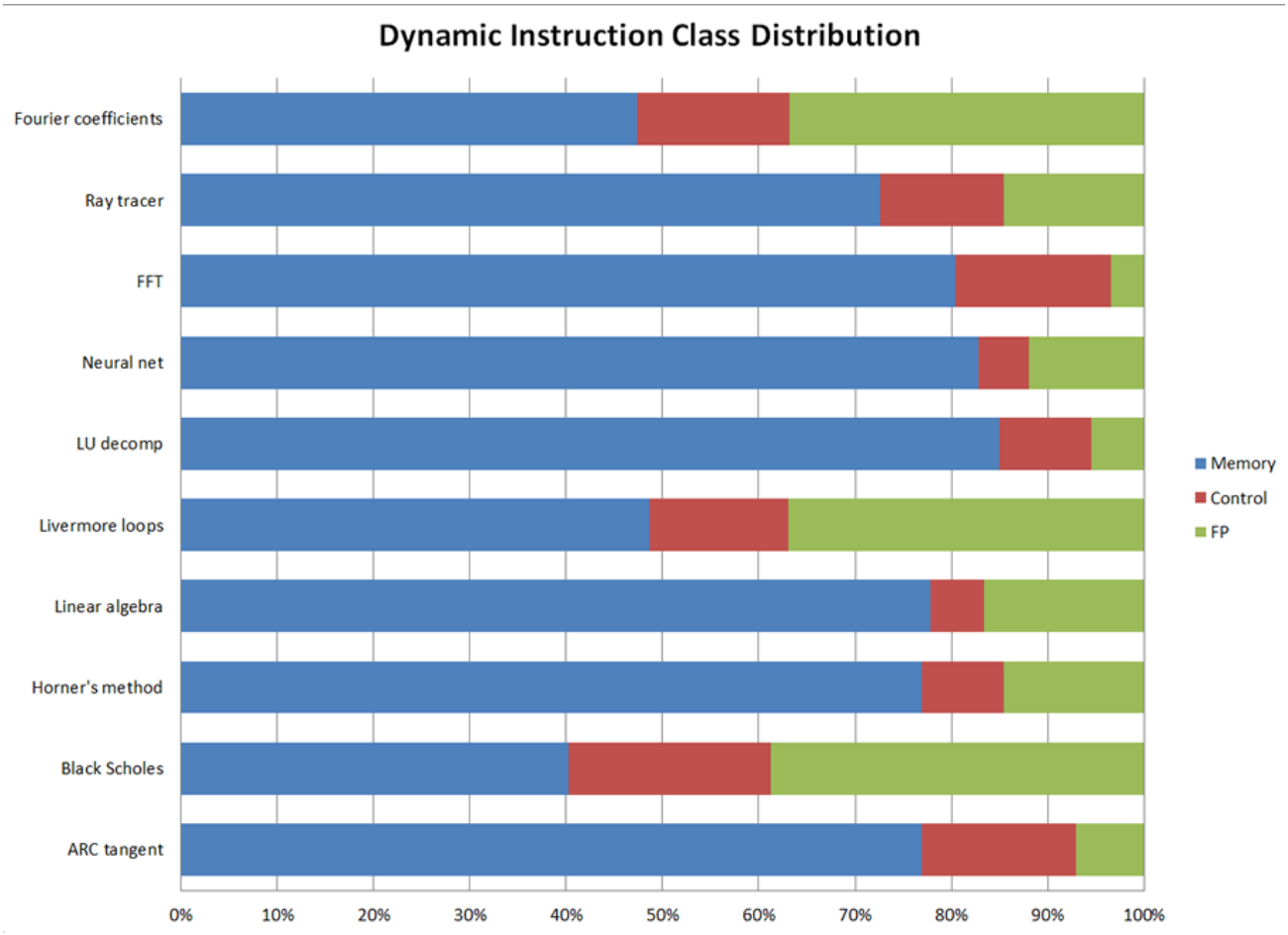

Figure 29: FPMark - instruction mix in each benchmark kernel [FPM].

For each configuration the measurements are presented in one set of tables that show workload iterations computed per second.

Table 44: NOEL-V - FPMark 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
atan-1M	0.0740	0.1480	0.1850	0.2467	0.2467	0.2467	0.2467	0.2467	0.2467	0.2467	0.2714	0.2954
atan-1M-sp	0.0938	0.1876	0.2345	0.3126	0.3126	0.3126	0.3126	0.3126	0.3126	0.3126	0.3438	0.3744
atan-1k	76.2265	152.3763	228.4826	304.1270	295.6306	296.0507	294.6376	293.9707	293.8497	293.2723	292.6629	293.0403
atan-1k-sp	95.8681	191.6039	287.3150	382.2630	369.6858	367.3769	367.2420	365.1367	365.9652	366.2467	366.9186	366.6630
atan-64k	1.2130	2.4265	3.6320	4.8422	4.8405	4.8407	4.8402	4.8394	4.8400	4.8411	4.8390	4.8395
atan-64k-sp	1.5360	3.0723	4.5992	6.1322	6.1297	6.1285	6.1300	6.1283	6.1284	6.1282	6.1281	6.1281
blacks-big-n5000v200	0.0021	0.0043	0.0053	0.0071	0.0071	0.0071	0.0071	0.0071	0.0071	0.0071	0.0078	0.0085
blacks-big-n5000v200-sp	0.0025	0.0049	0.0062	0.0082	0.0082	0.0082	0.0082	0.0082	0.0082	0.0082	0.0090	0.0098
blacks-mid-n1000v40	0.0534	0.1068	0.1335	0.1780	0.1780	0.1780	0.1780	0.1780	0.1780	0.1780	0.1958	0.2132
blacks-mid-n1000v40-sp	0.0617	0.1233	0.1542	0.2055	0.2055	0.2055	0.2055	0.2055	0.2055	0.2055	0.2261	0.2461
blacks-sml-n500v20	0.2136	0.4271	0.5338	0.7116	0.7115	0.7116	0.7116	0.7115	0.7115	0.7115	0.7826	0.8520
blacks-sml-n500v20-sp	0.2467	0.4929	0.6163	0.8214	0.8213	0.8213	0.8214	0.8213	0.8213	0.8213	0.9031	0.9833
horner-big-100k	0.3246	0.6489	0.9540	1.2948	1.2942	1.2942	1.2936	1.2938	1.2940	1.2941	1.2937	1.2937
horner-big-100k-sp	0.3324	0.6644	0.9771	1.3262	1.3259	1.3258	1.3248	1.3254	1.3257	1.3256	1.3250	1.3253
horner-mid-10k	3.2403	6.4758	9.6976	12.9266	12.8687	12.8788	12.8848	12.8679	12.8245	12.8588	12.8738	12.8302
horner-mid-10k-sp	3.3182	6.6334	9.9321	13.2391	13.1952	13.1966	13.1879	13.1785	13.1726	13.1851	13.1827	13.1789
horner-sml-1k	31.8446	63.4872	95.4681	126.9180	120.6404	121.1798	120.9951	112.7523	111.9871	113.9588	117.2553	111.4678
horner-sml-1k-sp	32.7633	65.3509	97.8751	130.3288	124.4694	125.7846	124.6712	117.7953	122.8954	119.9918	122.5926	116.8975
inner-product-big-100k	0.1602	0.2939	0.3889	0.4797	0.4627	0.4648	0.4436	0.4618	0.4778	0.4716	0.4461	0.4672
inner-product-big-100k-sp	0.1810	0.3386	0.4583	0.6031	0.6078	0.6172	0.5742	0.6161	0.6213	0.6106	0.5948	0.6081
inner-product-mid-10k	1.5972	2.9657	4.0822	4.7589	4.6398	4.6577	4.6540	4.6583	4.6294	4.6383	4.6310	4.6434
inner-product-mid-10k-sp	1.8068	3.3531	4.8859	6.3312	5.9026	6.3011	6.1369	5.9718	6.1191	6.1537	6.1490	6.1639

continues on next page

Table 44 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
inner-product-sml-1k	19.6202	30.7891	47.6168	65.7722	45.1080	47.3440	52.1050	45.5145	43.5028	45.1937	49.8480	48.8711
inner-product-sml-1k-sp	19.7235	31.9071	44.9843	56.6348	44.6369	47.0788	51.4801	44.8893	43.3783	45.0755	48.5625	47.4541
linear_alg-big-1000x1000	0.0008	0.0013	0.0019	0.0019	0.0024	0.0028	0.0027	0.0028	0.0028	0.0027	0.0028	0.0028
linear_alg-big-1000x1000-sp	0.0008	0.0014	0.0020	0.0020	0.0025	0.0030	0.0029	0.0030	0.0030	0.0029	0.0030	0.0031
linear_alg-mid-100x100	0.1718	0.3372	0.4903	0.6315	0.6323	0.6355	0.6323	0.6290	0.6311	0.6374	0.6399	0.6393
linear_alg-mid-100x100-sp	0.1826	0.3638	0.5329	0.6938	0.6910	0.7141	0.6972	0.7139	0.7113	0.6910	0.7114	0.7034
linear_alg-sml-50x50	1.3384	2.6653	3.9700	5.2455	5.2379	5.2474	5.2442	5.2416	5.2469	5.2466	5.2334	5.2394
linear_alg-sml-50x50-sp	1.4069	2.8126	4.2096	5.6026	5.5831	5.6017	5.5820	5.5856	5.5932	5.5893	5.5870	5.5943
loops-all-big-100k	0.0009	0.0014	0.0019	0.0018	0.0018	0.0021	0.0023	0.0025	0.0021	0.0022	0.0024	0.0025
loops-all-big-100k-sp	0.0010	0.0016	0.0022	0.0021	0.0021	0.0024	0.0027	0.0029	0.0024	0.0026	0.0027	0.0029
loops-all-mid-10k	0.0086	0.0160	0.0213	0.0250	0.0253	0.0253	0.0249	0.0251	0.0251	0.0251	0.0251	0.0250
loops-all-mid-10k-sp	0.0101	0.0188	0.0249	0.0291	0.0291	0.0292	0.0292	0.0291	0.0293	0.0292	0.0291	0.0290
loops-all-tiny	4.3512	8.3651	12.9756	17.1804	15.6666	16.5552	16.2718	16.2027	15.2128	15.7500	16.1051	15.6710
loops-all-tiny-sp	4.8886	9.2005	13.9113	19.2649	17.7318	18.4373	18.1858	17.2729	16.6528	17.2479	17.8291	17.0870
lu-big-2000x2_50	0.0080	0.0157	0.0196	0.0259	0.0305	0.0306	0.0280	0.0258	0.0239	0.0305	0.0306	0.0305
lu-big-2000x2_50-sp	0.0084	0.0167	0.0209	0.0278	0.0330	0.0331	0.0302	0.0277	0.0257	0.0330	0.0331	0.0331
lu-mid-200x2_50	0.7000	1.3978	2.0900	2.7793	2.6292	2.6250	2.6235	2.6197	2.6151	2.6151	2.6169	2.6164
lu-mid-200x2_50-sp	0.7021	1.4032	2.0997	2.7991	2.6776	2.6733	2.6672	2.6646	2.6616	2.6603	2.6597	2.6623
lu-sml-20x2_50	7.8143	15.6021	23.3711	30.9987	28.9995	28.9667	28.8907	28.8305	28.7870	28.7685	28.7502	28.7345
lu-sml-20x2_50-sp	8.1561	16.2969	24.4229	32.4780	30.6028	30.5256	30.4270	30.3662	30.3385	30.2954	30.2894	30.2861
nnet-data1-sp	6.2273	12.4171	18.5836	24.7819	24.7170	24.7133	24.6920	24.7011	24.7145	24.7097	24.7807	24.7084
nnet_data1	5.4324	10.8527	16.2586	21.6263	21.5801	21.5787	21.5647	21.5806	21.5764	21.6258	21.5619	21.6258
nnet_test	0.0106	0.0211	0.0264	0.0351	0.0351	0.0351	0.0351	0.0351	0.0351	0.0351	0.0385	0.0420
nnet_test-sp	0.0098	0.0196	0.0246	0.0327	0.0327	0.0327	0.0327	0.0327	0.0327	0.0327	0.0360	0.0392
radix2-big-64k	1.0966	2.1274	3.0579	3.8416	3.8381	3.8376	3.8361	3.8367	3.8376	3.8371	3.8362	3.8362
radix2-mid-8k	15.3371	28.9392	38.7986	43.0159	42.5072	42.5114	42.4657	42.4488	42.3100	42.3810	42.3650	42.3492
radix2-sml-2k	74.9412	149.0149	221.3859	287.5331	261.3190	259.4996	269.0783	260.5619	263.9755	254.3753	264.1764	261.3778

continues on next page

Table 44 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
ray-1024x768at24s	0.0000	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
ray-320x240at8s	0.0009	0.0018	0.0022	0.0029	0.0029	0.0029	0.0029	0.0029	0.0029	0.0029	0.0032	0.0035
ray-64x48at4s	0.0436	0.0872	0.1283	0.1677	0.1677	0.1677	0.1677	0.1677	0.1677	0.1677	0.1677	0.1677
xp1px-big-c10000n2000	0.0006	0.0012	0.0015	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0022	0.0024
xp1px-mid-c1000n200	0.0617	0.1234	0.1542	0.2056	0.2056	0.2056	0.2056	0.2056	0.2056	0.2056	0.2261	0.2462
xp1px-sml-c100n20	6.3868	12.7703	19.1113	25.4732	25.3756	25.3762	25.4001	25.3685	25.3563	25.3633	25.3704	25.3620

Table 45: NOEL-V - FPMark results for configuration NV2D44 - workload iterations per second at 112MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
atan-1M	.	0.3214	0.4013	0.5349	0.5343	0.5344	0.5344	0.5343	.	.	.	.
atan-1M-sp	.	0.3628	0.4536	0.6043	0.6036	0.6034	0.6037	0.6034	0.6042	.	.	.
atan-1k	.	330.3928	497.8592	662.6905	641.8897	637.7551	639.0593	627.9435	.	.	.	.
atan-1k-sp	.	370.9336	556.2973	740.1377	713.9797	705.7163	711.6931	704.2749	.	.	.	.
atan-64k	.	5.2510	7.8758	10.5043	10.5126	10.5110	10.5145	10.4991	10.5095	.	.	.
atan-64k-sp	.	5.9265	8.8964	11.8773	11.8443	11.8488	11.8527	11.8455	11.8451	.	.	.
blacks-big-n5000v200	.	0.0083	0.0104	0.0138	0.0138	0.0138	0.0138	0.0138	.	.	.	.
blacks-big-n5000v200-sp	.	0.0087	0.0109	0.0145	0.0145	0.0145	0.0145	0.0145	0.0145	.	.	.
blacks-mid-n1000v40	.	0.2070	0.2588	0.3450	0.3450	0.3450	0.3450	0.3450	0.3450	.	.	.
blacks-mid-n1000v40-sp	.	0.2175	0.2719	0.3624	0.3624	0.3624	0.3624	0.3624	.	.	.	.
blacks-sml-n500v20	.	0.8279	1.0348	1.3793	1.3795	1.3795	1.3795	1.3795	1.3795	.	.	.
blacks-sml-n500v20-sp	.	0.8692	1.0867	1.4484	1.4482	1.4482	1.4484	1.4482	1.4482	.	.	.
horner-big-100k	.	1.2544	1.8453	2.5033	2.5026	2.5024	2.5009	2.5008	2.5022	.	.	.
horner-big-100k-sp	.	1.2644	1.8603	2.5263	2.5261	2.5260	2.5230	2.5245	2.5260	.	.	.

continues on next page

Table 45 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
horner-mid-10k	.	12.4305	18.6773	24.9140	24.6810	24.7917	24.8084	24.5206	24.6585	.	.	.
horner-mid-10k-sp	.	12.6108	18.8430	25.0878	24.9296	25.0589	25.0897	24.7746	24.8874	.	.	.
horner-sml-1k	.	123.1770	184.7336	245.7184	235.9938	235.2443	235.9882	227.1230	230.5103	.	.	.
horner-sml-1k-sp	.	123.6598	185.7114	246.7186	237.0174	235.3052	237.2761	231.2406	.	.	.	.
inner-product-big-100k	.	0.3031	0.3770	0.4342	0.4286	0.4190	0.4157	0.4221	0.4403	.	.	.
inner-product-big-100k-sp	.	0.3970	0.5193	0.6989	0.6821	0.6936	0.6560	0.6923	0.7144	.	.	.
inner-product-mid-10k	.	3.0859	4.0334	4.4111	4.4069	4.4065	4.3836	4.3874	.	.	.	.
inner-product-mid-10k-sp	.	4.4130	6.3646	7.2947	7.3794	7.3862	7.3130	7.2595	7.3103	.	.	.
inner-product-sml-1k	.	47.9478	71.2048	85.8295	84.4951	83.2917	81.2414	80.7233	79.4407	.	.	.
inner-product-sml-1k-sp	.	50.0676	73.2011	101.2351	94.3218	94.7508	93.1012	92.2509	.	.	.	.
linear_alg-big-1000x1000	.	0.0019	0.0029	0.0029	0.0035	0.0042	0.0039	0.0041	.	.	.	.
linear_alg-big-1000x1000-sp	.	0.0022	0.0032	0.0033	0.0040	0.0048	0.0045	0.0047	0.0048	.	.	.
linear_alg-mid-100x100	.	0.5845	0.8556	1.0797	1.0822	1.0946	1.1117	1.0994	.	.	.	.
linear_alg-mid-100x100-sp	.	0.6138	0.9021	1.1788	1.1917	1.1930	1.1742	1.1742	1.1826	.	.	.
linear_alg-sml-50x50	.	4.4216	6.6151	8.8041	8.7895	8.7778	8.7898	8.7906	8.7960	.	.	.
linear_alg-sml-50x50-sp	.	4.5830	6.8600	9.1441	9.1228	9.1376	9.1146	9.1266	.	.	.	.
loops-all-big-100k	.	0.0021	0.0026	0.0025	0.0025	0.0029	0.0031	0.0032	.	.	.	.
loops-all-big-100k-sp	.	0.0023	0.0029	0.0029	0.0029	0.0032	0.0035	0.0037	.	.	.	.
loops-all-mid-10k	.	0.0289	0.0390	0.0453	0.0455	0.0454	0.0457	0.0454	0.0453	.	.	.
loops-all-mid-10k-sp	.	0.0331	0.0473	0.0589	0.0590	0.0587	0.0592	0.0591	0.0591	.	.	.
loops-all-tiny	.	14.1379	21.1425	28.1341	27.2390	27.3718	27.2124	26.7967	.	.	.	.
loops-all-tiny-sp	.	14.6130	21.8876	29.1477	27.7239	28.4398	27.9767	27.7223	27.0256	.	.	.
lu-big-2000x2_50	.	0.0279	0.0349	0.0461	0.0537	0.0543	0.0495	0.0454	0.0420	.	.	.

continues on next page

Table 45 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
lu-big-2000x2_50-sp	.	0.0288	0.0360	0.0480	0.0571	0.0570	0.0520	0.0478	0.0441	.	.	.
lu-mid-200x2_50	.	2.3463	3.5027	4.6664	4.5135	4.4976	4.4812	4.4676	4.4532	.	.	.
lu-mid-200x2_50-sp	.	2.2976	3.4385	4.5877	4.4412	4.4397	4.4260	4.4217	.	.	.	.
lu-sml-20x2_50	.	26.1340	39.3224	52.2128	50.1910	49.9958	49.7882	49.6236	49.5201	.	.	.
lu-sml-20x2_50-sp	.	26.5861	39.8917	53.1163	50.9671	50.8525	50.6984	50.5753	.	.	.	.
nnet-data1-sp	.	20.1898	30.2215	40.3193	40.3210	40.2690	40.2625	40.2350	40.2625	.	.	.
nnet_data1	.	18.9534	28.3567	37.8515	37.8515	37.7943	37.8515	37.8515	37.8515	.	.	.
nnet_test	.	0.0371	0.0464	0.0618	0.0618	0.0618	0.0618	0.0618	0.0618	.	.	.
nnet_test-sp	.	0.0321	0.0402	0.0536	0.0536	0.0536	0.0535	0.0535	.	.	.	.
radix2-big-64k	.	3.1415	4.3189	4.9390	4.9330	4.9334	4.9364	4.9387	.	.	.	.
radix2-mid-8k	.	41.0583	59.8390	74.9580	74.1939	74.3224	73.7126	74.6141	74.0790	.	.	.
radix2-sml-2k	.	214.0113	321.7886	430.6335	421.8715	420.8134	419.1695	416.8595	418.6342	.	.	.
ray-1024x768at24s	.	0.0001	0.0002	0.0003	0.0002	0.0002	0.0002	0.0003	0.0002	.	.	.
ray-320x240at8s	.	0.0040	0.0050	0.0066	0.0066	0.0066	0.0066	0.0066	.	.	.	.
ray-64x48at4s	.	0.1970	0.2897	0.3787	0.3787	0.3787	0.3788	0.3787	0.3787	.	.	.
xp1px-big-c10000n2000	.	0.0024	0.0030	0.0040	0.0040	0.0040	0.0040	0.0040	.	.	.	.
xp1px-mid-c1000n200	.	0.2426	0.3033	0.4044	0.4044	0.4044	0.4044	0.4044	0.4044	.	.	.
xp1px-sml-c100n20	.	24.8979	37.2703	49.7092	49.5221	49.5074	49.5025	49.4878	.	.	.	.

Table 46: PFS - FPMark results for PolarFire SoC RISC-V - workload iterations per second at 667MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
atan-1M	1.0995	2.1891	2.7211	3.5549	4.0950	4.1305	3.9698	3.6140	3.4141	4.2882	4.2802	4.1768
atan-1M-sp	1.4124	2.8209	3.5100	4.6447	5.3079	5.2966	5.0150	4.6882	4.3309	5.5371	5.5000	5.5970
atan-1k	1204.8193	2412.5452	3607.5036	4823.9267	4807.6923	4814.6365	4812.3195	4649.0005	4810.0048	4807.6923	4805.3820	4805.3820
atan-1k-sp	1496.5572	2985.9660	4486.3167	5740.5285	5711.0223	5977.2863	5973.7157	5973.7157	5966.5871	5720.8238	5717.5529	5966.5871
atan-64k	18.9797	37.1485	54.8847	71.9632	72.2909	72.6427	72.2543	72.6903	72.5005	72.1293	72.3956	72.6744
atan-64k-sp	24.0964	48.2323	71.6384	94.5001	94.0468	93.6242	93.6154	93.2923	93.3968	93.1966	93.1966	93.2053

continues on next page

Table 46 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
blacks-big-n5000v200	0.1219	0.2431	0.3049	0.4042	0.4766	0.4819	0.4430	0.4048	0.3755	0.4828	0.4836	0.4813
blacks-big-n5000v200-sp	0.1660	0.3332	0.4168	0.5537	0.6529	0.6532	0.6042	0.5516	0.5155	0.6536	0.6573	0.6609
blacks-mid-n1000v40	3.0321	6.0901	7.6104	10.1420	11.2613	10.9170	10.4058	10.0705	9.5057	11.7371	11.8662	12.0846
blacks-mid-n1000v40-sp	4.1649	8.3195	10.3950	13.8504	14.9701	14.6843	14.0056	13.8313	14.1443	15.5280	15.4494	16.4159
blacks-sml-n500v20	11.9904	24.2131	30.3951	40.4858	40.3226	40.3226	40.6504	40.3226	46.0829	45.4545	46.0251	47.6190
blacks-sml-n500v20-sp	16.5563	33.2226	41.4938	54.6448	54.9451	54.9451	54.9451	61.7284	54.9451	60.6061	63.9535	64.5161
horner-big-100k	6.7254	12.9182	18.7723	24.8633	25.1953	24.8818	24.8880	24.5640	24.6548	25.1953	25.1889	25.1636
horner-big-100k-sp	8.5106	16.4690	24.0616	32.5098	31.5457	32.1854	32.1854	32.3102	31.9183	32.0102	31.2402	31.2110
horner-mid-10k	75.2785	150.3986	222.5189	284.4141	286.1230	272.5538	276.0144	270.1972	263.7826	268.5285	265.8867	264.0613
horner-mid-10k-sp	87.4661	174.5810	262.0545	349.2840	346.9813	346.3803	342.2313	340.5995	339.7893	337.1544	325.6268	331.1258
horner-sml-1k	737.0826	1472.7541	2212.3894	2882.6751	2944.6408	2874.3892	2941.1765	2937.7203	2862.0492	2935.1335	2935.1335	2935.1335
horner-sml-1k-sp	851.4261	1700.3911	2538.7154	3401.3605	3399.0483	3394.4331	3305.7851	3309.0668	3390.9800	3390.9800	3302.5099	3387.5339
inner-product-big-100k	0.9751	1.5782	2.0333	2.4808	2.4789	2.3313	2.4579	2.4152	2.3049	2.3926	2.3909	2.4007
inner-product-big-100k-sp	2.2622	3.1701	3.8588	4.5455	4.4743	4.2955	4.5424	4.3649	4.1152	4.2762	4.3687	4.2185
inner-product-mid-10k	14.5059	27.7047	36.8494	42.6758	41.6536	42.0345	40.9458	41.0130	40.9668	41.0214	40.6835	40.7415
inner-product-mid-10k-sp	23.7713	45.1671	62.4122	79.5387	78.4160	77.9575	76.3796	76.8344	75.5572	74.1152	74.2253	73.5835
inner-product-sml-1k	284.9003	427.7160	499.0020	515.9959	583.4306	582.7506	549.4505	517.5983	558.0357	560.5381	540.8329	515.1984
inner-product-sml-1k-sp	403.8772	749.0637	1070.6638	1243.7811	1203.3694	1158.7486	984.2520	995.0249	1066.0981	1050.4202	1066.0981	1041.6667
linear_alg-big-1000x1000	0.0084	0.0134	0.0192	0.0194	0.0284	0.0283	0.0282	0.0282	0.0282	0.0282	0.0282	0.0282
linear_alg-big-1000x1000-sp	0.0134	0.0212	0.0306	0.0311	0.0459	0.0456	0.0459	0.0459	0.0458	0.0459	0.0457	0.0459
linear_alg-mid-100x100	3.0916	6.1820	9.1008	11.8963	11.9019	12.1981	11.9818	11.7758	12.0948	12.0192	12.0163	11.3482

continues on next page

Table 46 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
linear_alg-mid-100x100-sp	4.1425	8.2754	12.1625	15.6495	15.8629	16.3132	15.9898	15.8328	15.7928	15.8228	15.8228	15.9134
linear_alg-sml-50x50	24.2565	48.5060	72.7379	97.0685	96.8804	96.7680	96.7492	96.8242	96.5624	96.6931	96.2464	95.0932
linear_alg-sml-50x50-sp	30.2535	60.4814	90.6618	121.0361	118.5396	118.6240	119.7031	118.3712	118.5396	118.2592	118.3152	120.6564
loops-all-big-100k	0.0088	0.0113	0.0134	0.0121	0.0147	0.0146	0.0144	0.0142	0.0142	0.0142	0.0142	0.0142
loops-all-big-100k-sp	0.0131	0.0164	0.0175	0.0168	0.0176	0.0173	0.0171	0.0168	0.0169	0.0168	0.0168	0.0168
loops-all-mid-10k	0.1317	0.2323	0.3107	0.3683	0.3727	0.3612	0.3554	0.3523	0.3596	0.3569	0.3582	0.3493
loops-all-mid-10k-sp	0.1869	0.3429	0.4743	0.5858	0.5936	0.5729	0.5592	0.5586	0.5719	0.5658	0.5679	0.5506
loops-all-tiny	91.0747	182.2822	271.1497	363.6364	363.1082	362.3188	361.2717	362.8447	361.5329	361.5329	348.6750	350.1401
loops-all-tiny-sp	107.7122	215.5172	322.7889	431.0345	429.5533	428.0822	427.3504	428.8165	426.9855	427.3504	428.0822	428.0822
lu-big-2000x2_50	0.1543	0.3086	0.3853	0.5106	0.5548	0.6001	0.5550	0.5079	0.4735	0.5959	0.5939	0.5933
lu-big-2000x2_50-sp	0.1767	0.3534	0.4421	0.5872	0.6898	0.6919	0.6397	0.5852	0.5492	0.6916	0.6944	0.6948
lu-mid-200x2_50	12.9512	25.8900	38.8154	51.4086	51.3795	51.3057	51.4165	51.2269	51.1771	51.1247	51.1352	51.1326
lu-mid-200x2_50-sp	14.0050	28.0167	41.9992	55.7631	55.5093	55.5216	55.7103	55.4539	55.4570	55.3403	55.4416	55.3802
lu-sml-20x2_50	144.7639	289.6284	434.4615	574.5806	574.2506	576.4353	575.8710	572.0824	575.1093	572.1478	575.2747	572.4426
lu-sml-20x2_50-sp	161.7626	323.6770	485.5547	641.6838	644.2469	643.9150	640.4509	640.2459	643.2523	639.8771	643.0868	639.8771
nnet-data1-sp	162.7869	324.7808	487.0921	621.8905	648.0881	648.9293	619.1950	645.9948	648.0881	647.6684	647.2492	643.0868
nnet_data1	123.8850	247.8315	371.1952	492.3683	477.0992	494.0711	493.5834	493.0966	492.8536	492.8536	492.1260	492.8536
nnet_test	0.2455	0.4911	0.6141	0.8189	0.9593	0.9594	0.8952	0.8142	0.7572	0.9692	0.9691	0.9681
nnet_test-sp	0.2620	0.5235	0.6553	0.8680	1.0376	1.0249	0.9417	0.8727	0.8276	1.0301	1.0294	1.0385
radix2-big-64k	18.8576	26.2151	31.9234	31.9479	31.9428	30.7069	30.3794	28.6533	29.5561	28.7365	29.3032	29.0639
radix2-mid-8k	239.1944	475.3078	706.8636	918.0207	929.8001	929.1090	921.5740	921.1496	919.2866	909.2562	904.2409	907.7705
radix2-sml-2k	1697.2454	4390.8655	5086.2113	6752.1945	6746.2727	6742.1791	6731.7402	6729.0223	6695.2336	6727.6642	6691.2011	6728.1168
ray-1024x768at24s	0.0009	0.0018	0.0027	0.0036	0.0036	0.0036	0.0036	0.0036	0.0036	0.0036	0.0036	0.0036
ray-320x240at8s	0.0277	0.0554	0.0692	0.0920	0.1097	0.1097	0.1004	0.0917	0.0851	0.1096	0.1097	0.1097
ray-64x48at4s	1.3785	2.7542	4.0555	5.2582	5.2804	5.3237	5.4154	5.2527	5.4413	5.4366	5.4520	5.2676
xp1px-big-c10000n2000	0.0320	0.0640	0.0800	0.1062	0.1266	0.1266	0.1157	0.1060	0.0994	0.1265	0.1269	0.1269
xp1px-mid-c1000n200	3.2051	6.3857	8.0257	10.6724	11.9760	12.0192	11.4943	10.6724	10.1833	12.3762	11.7146	12.7389
xp1px-sml-c100n20	325.0975	649.3506	970.8738	1293.6611	1305.4830	1290.3226	1287.0013	1290.3226	1285.3470	1297.0169	1283.6970	1285.3470

The values shown in Tables 44 to 46 are plotted in Figures 30 to 82.

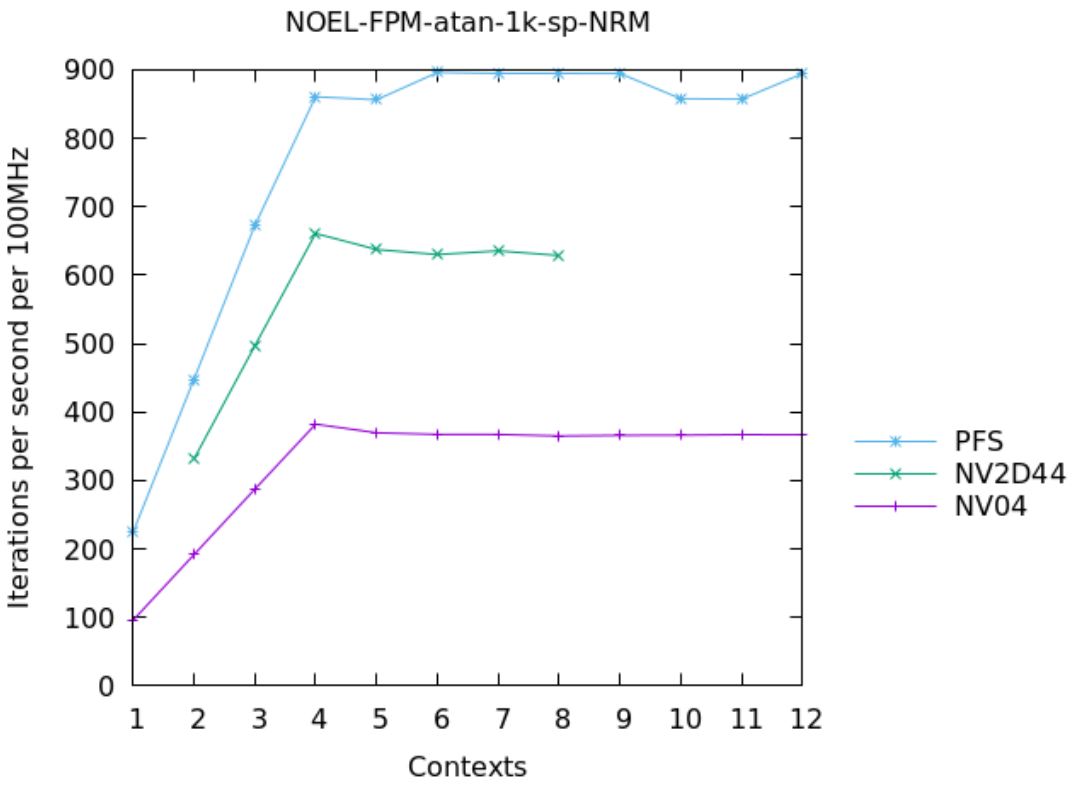

Figure 30: NOEL - FPMark - atan-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

8.4.4 Analysis

The observations are as follows. The workloads can be grouped into three categories depending on the difference and position of the performance curve of the three targets:

- Both the NOEL-V targets have significantly lower performance than RISC-V in PFS. Examples are *blacks, linear_alg, lu, nnet, xp1px.*
- The performance curves are nearly equally spaced in the order NV04, NV2D44, PFS (going from lower to higher performance). Examples are *atan, horner, ray*.
- A specific category is the inner, loops-all and radix2 workloads that in some cases exhibits quite a unique behaviour, not only for RISC-V targets, but also for Intel targets.

The *ray-1024x768at24s* workload is very specific; computing one iteration takes a very long time. The plotted curves are significantly affected by the precision used to measure the iterations per second, the values included in the log files are limited to four decimal places in the FPMark framework.

It is interesting that for a few workloads the NOEL-V can compete with the RISC-V multicore in PolarFire SoC. The reasons for this have not been analysed yet.

Overall, the current version of NOEL-V has a significantly lower normalized floating-point performance than the RISC-V (4+1)core processor implemented in PolarFire SoC. The most likely cause is a more efficient, pipelined FPU used in PolarFire SoC.

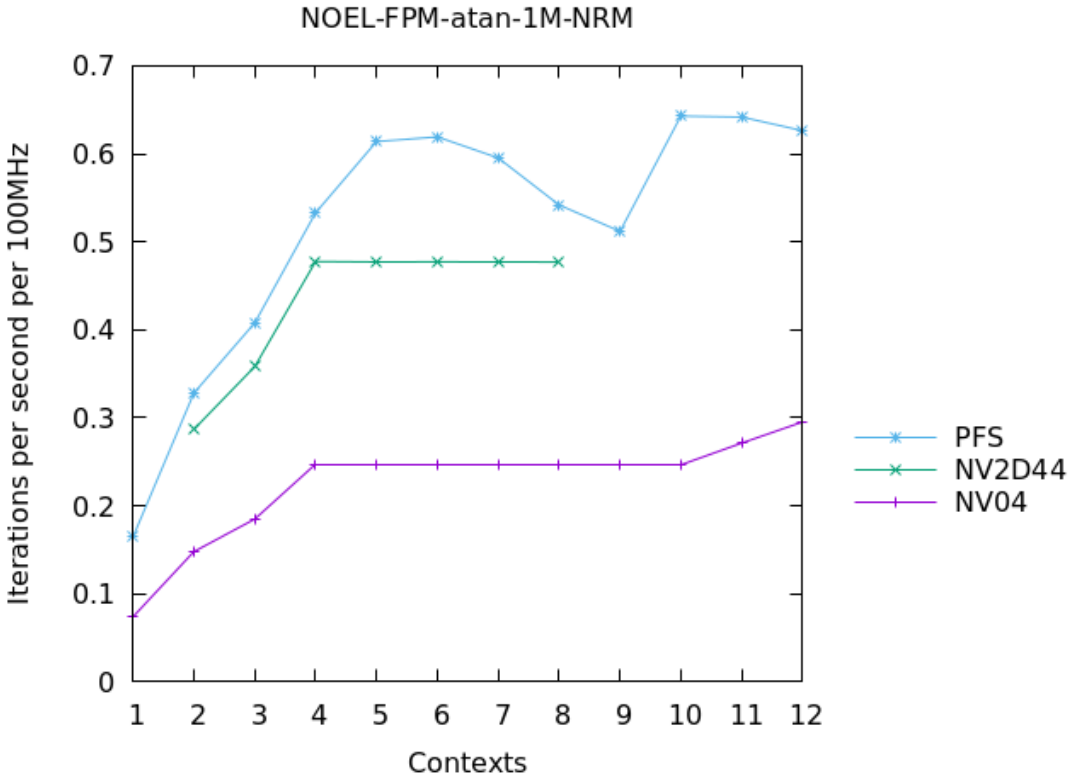

Figure 31: NOEL - FPMark - atan-1M, iterations per second at 100MHz. Higher values denote better performance.

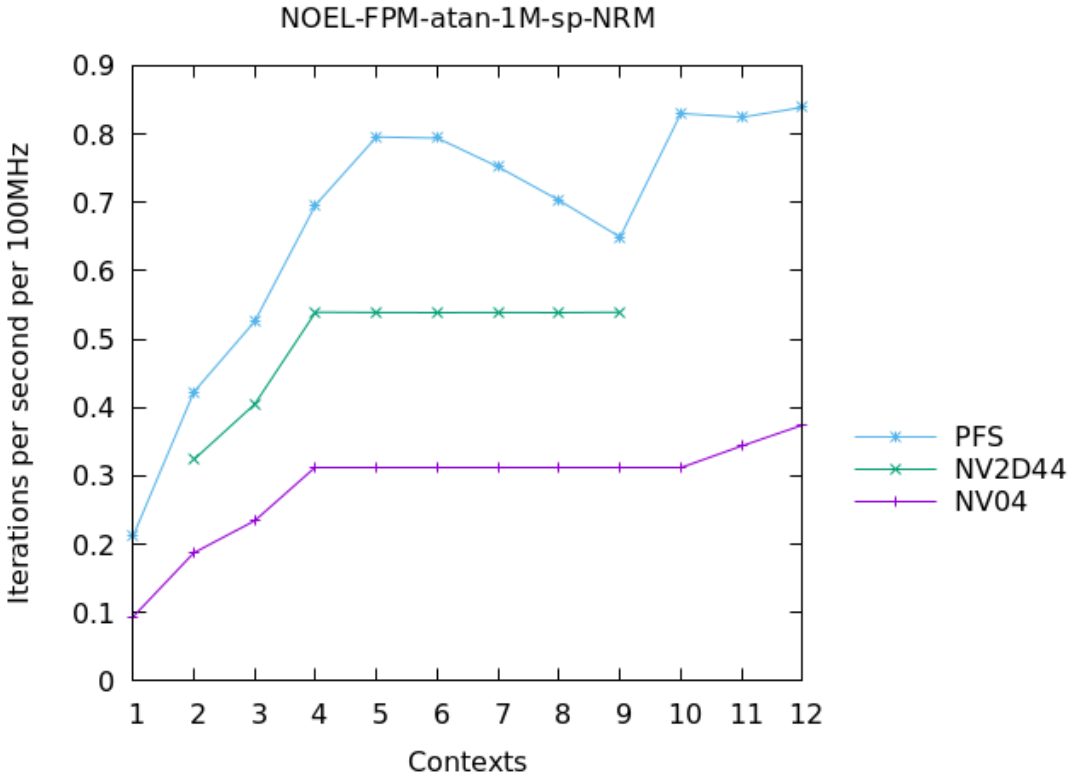

Figure 32: NOEL - FPMark - atan-1M-sp, iterations per second at 100MHz. Higher values denote better performance.

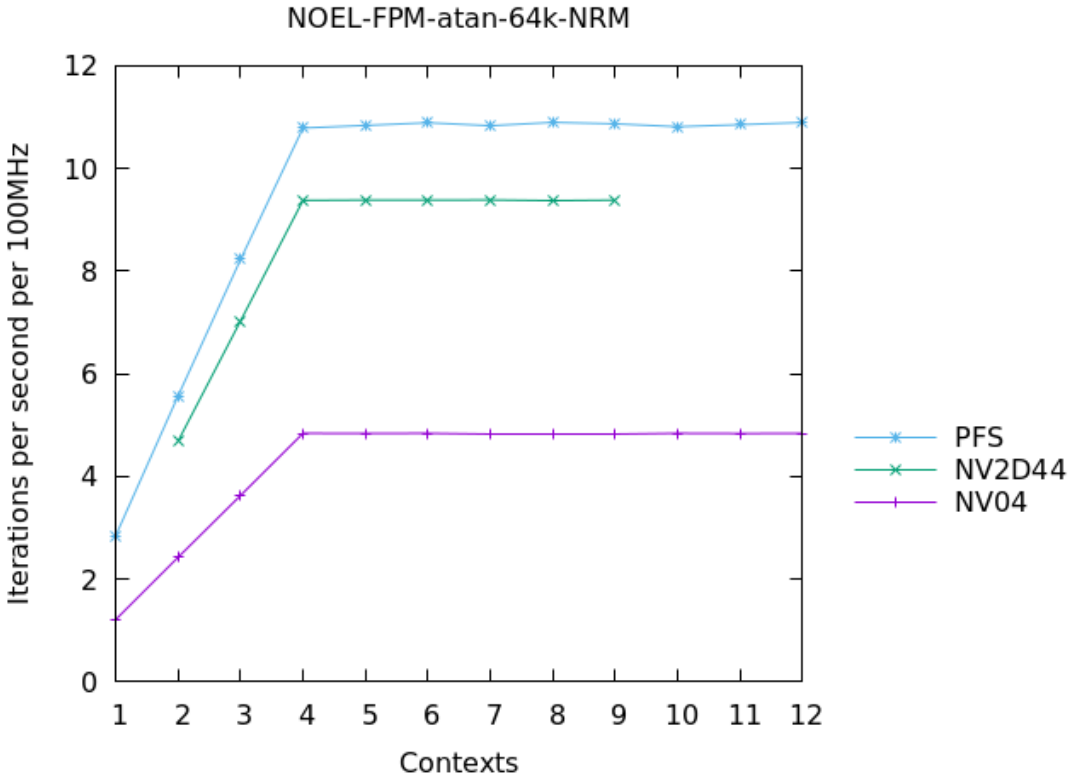

Figure 33: NOEL - FPMark - atan-64k, iterations per second at 100MHz. Higher values denote better performance.

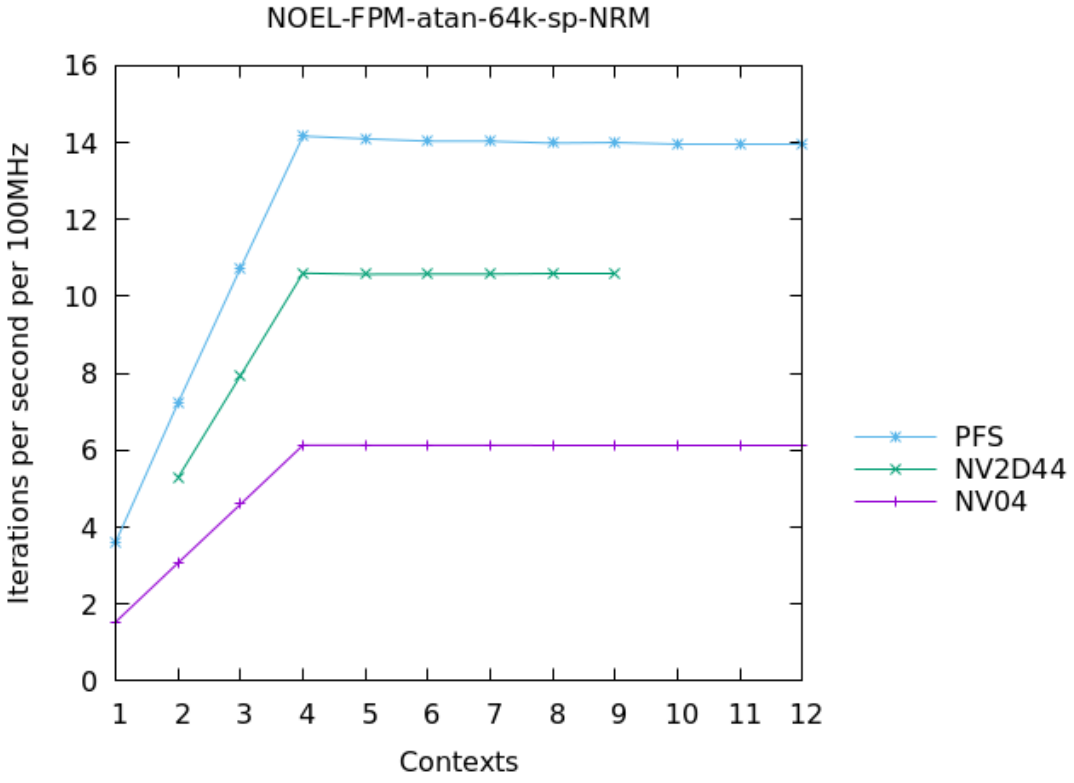

Figure 34: NOEL - FPMark - atan-64k-sp, iterations per second at 100MHz. Higher values denote better performance.

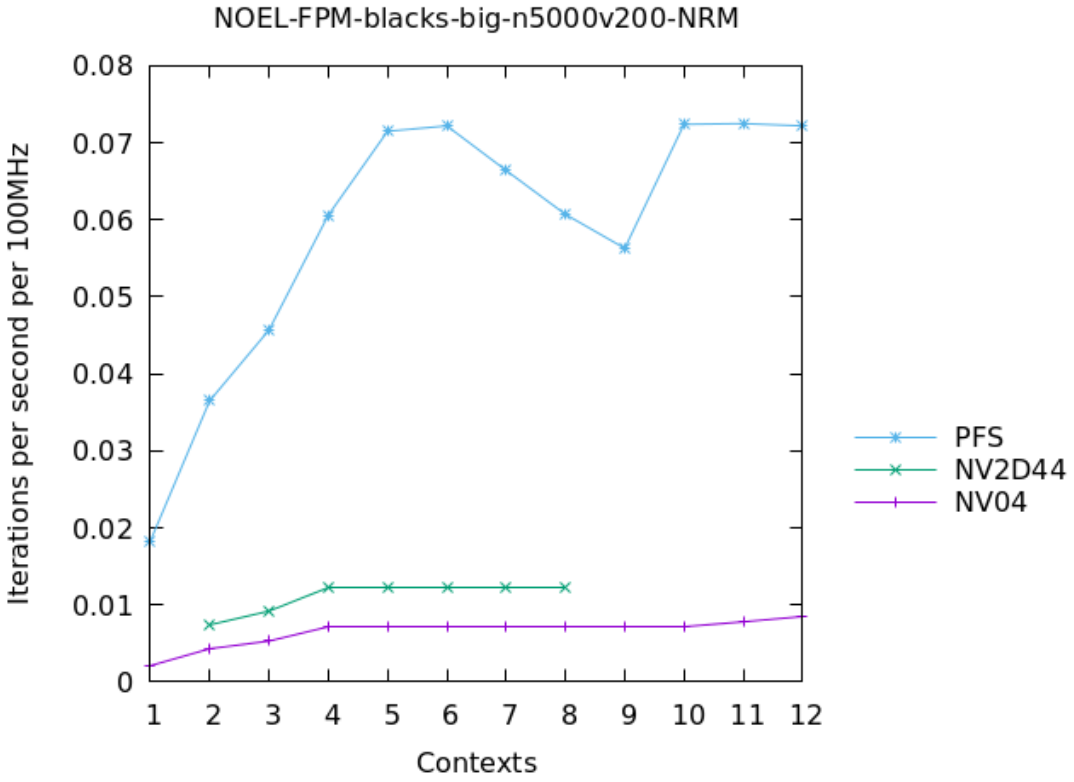

Figure 35: NOEL - FPMark - blacks-big-n5000v200, iterations per second at 100MHz. Higher values denote better performance.

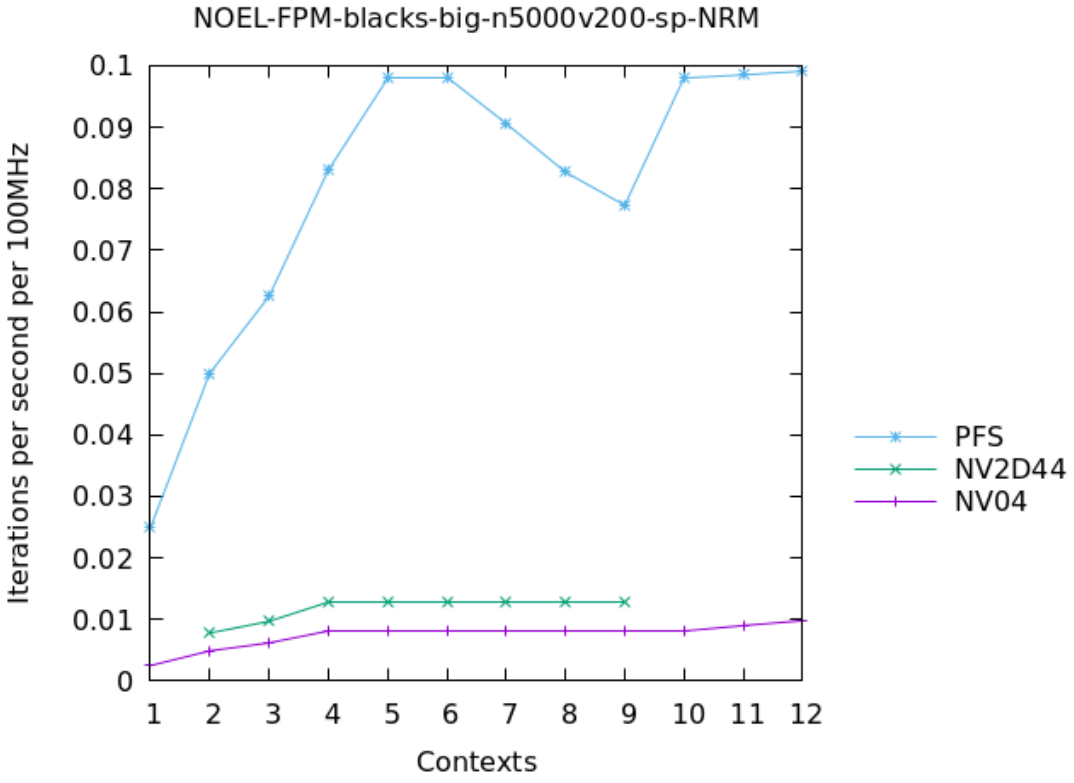

Figure 36: NOEL - FPMark - blacks-big-n5000v200-sp, iterations per second at 100MHz. Higher values denote better performance.

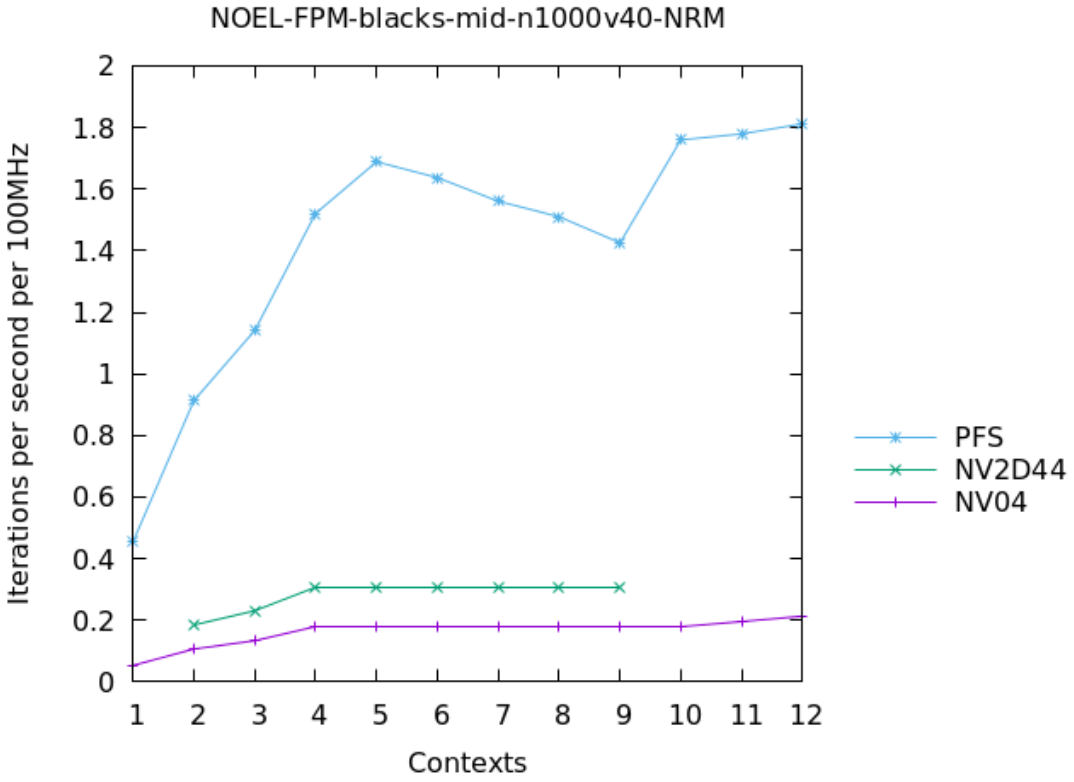

Figure 37: NOEL - FPMark - blacks-mid-n1000v40, iterations per second at 100MHz. Higher values denote better performance.

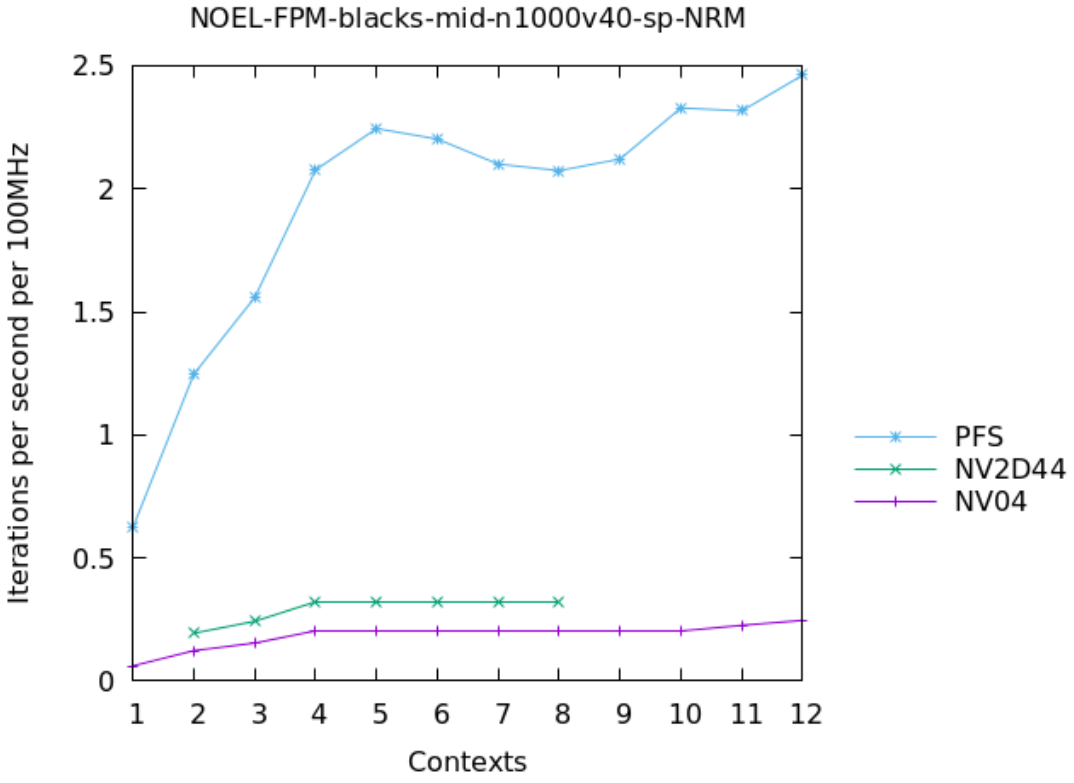

Figure 38: NOEL - FPMark - blacks-mid-n1000v40-sp, iterations per second at 100MHz. Higher values denote better performance.

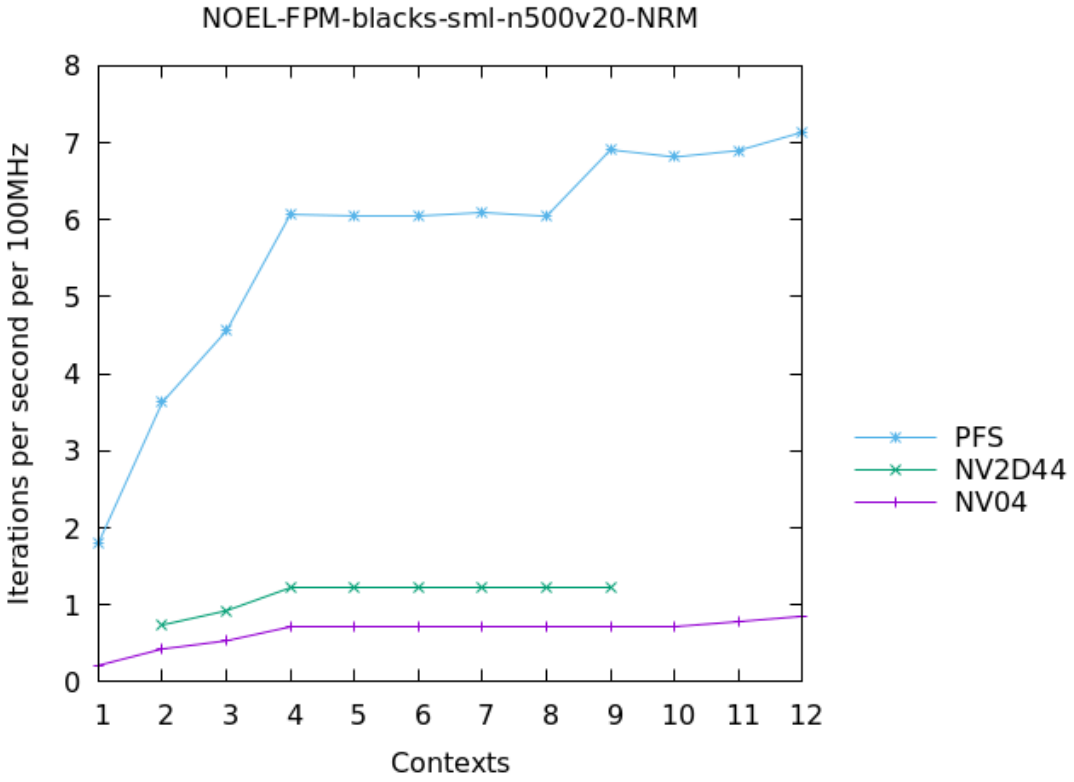

Figure 39: NOEL - FPMark - blacks-sml-n500v20, iterations per second at 100MHz. Higher values denote better performance.

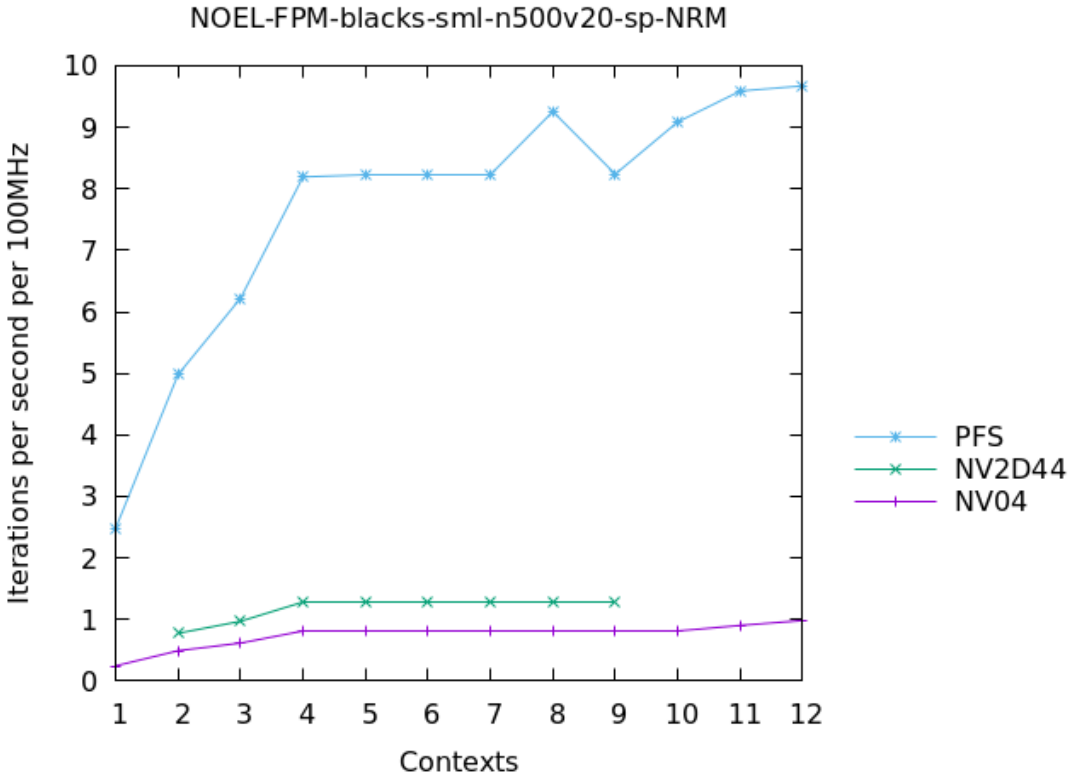

Figure 40: NOEL - FPMark - blacks-sml-n500v20-sp, iterations per second at 100MHz. Higher values denote better performance.

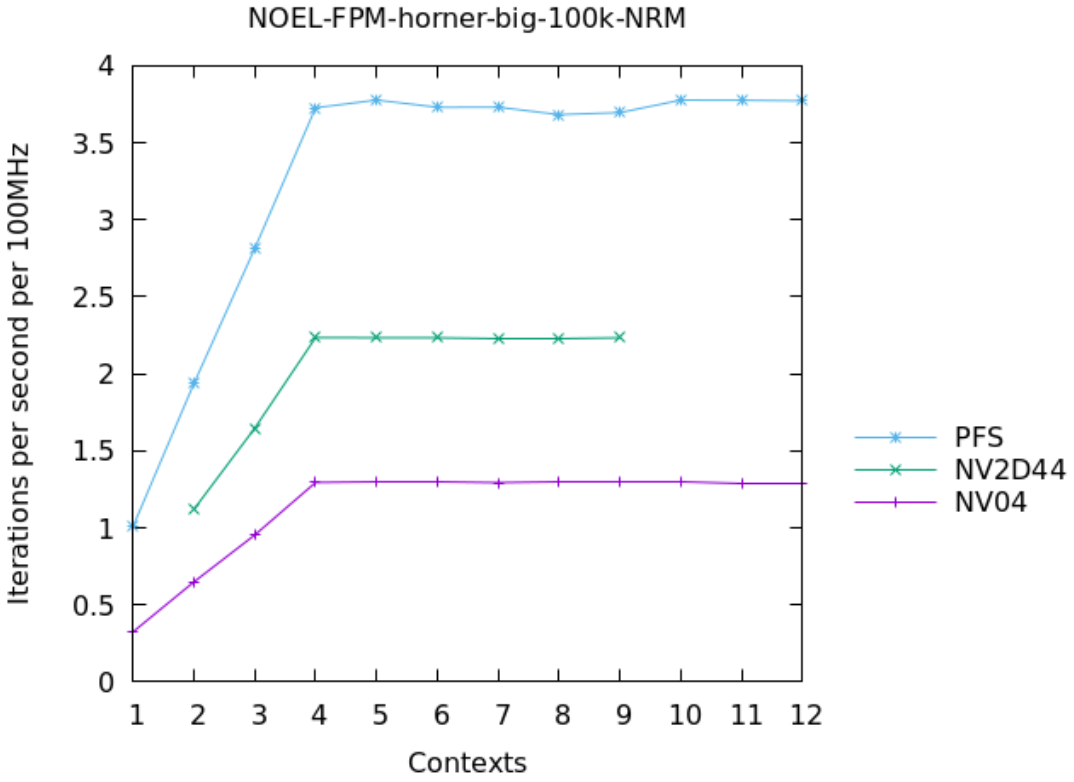

Figure 41: NOEL - FPMark - horner-big-100k, iterations per second at 100MHz. Higher values denote better performance.

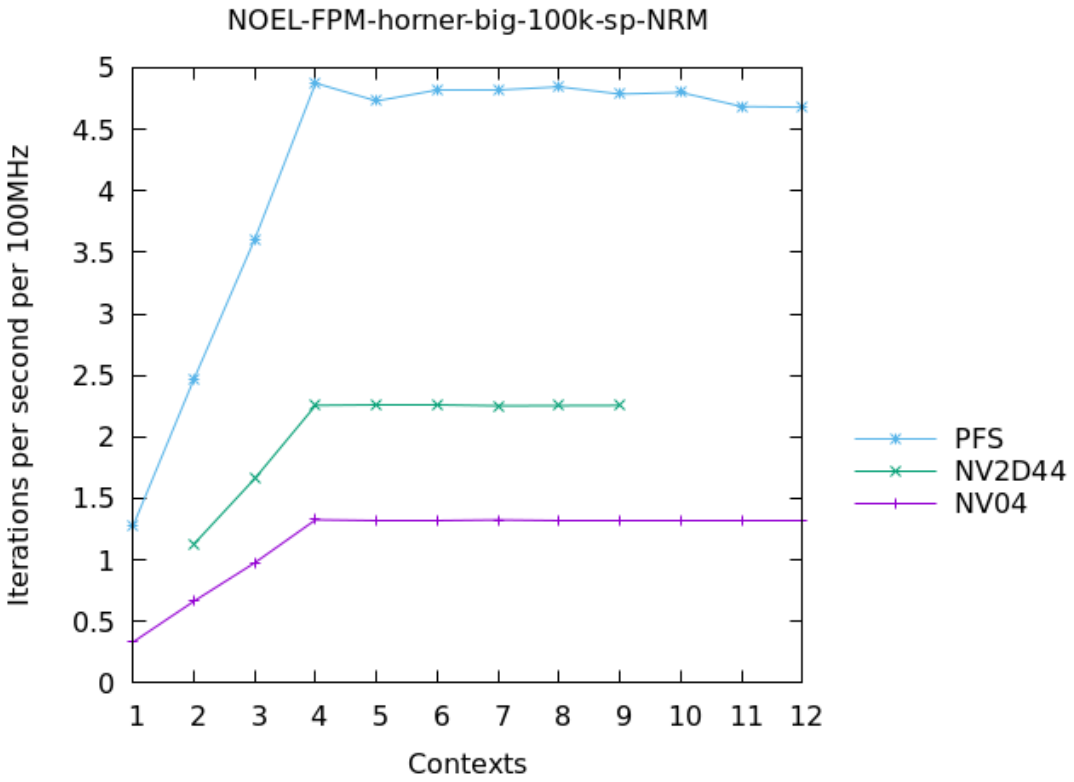

Figure 42: NOEL - FPMark - horner-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

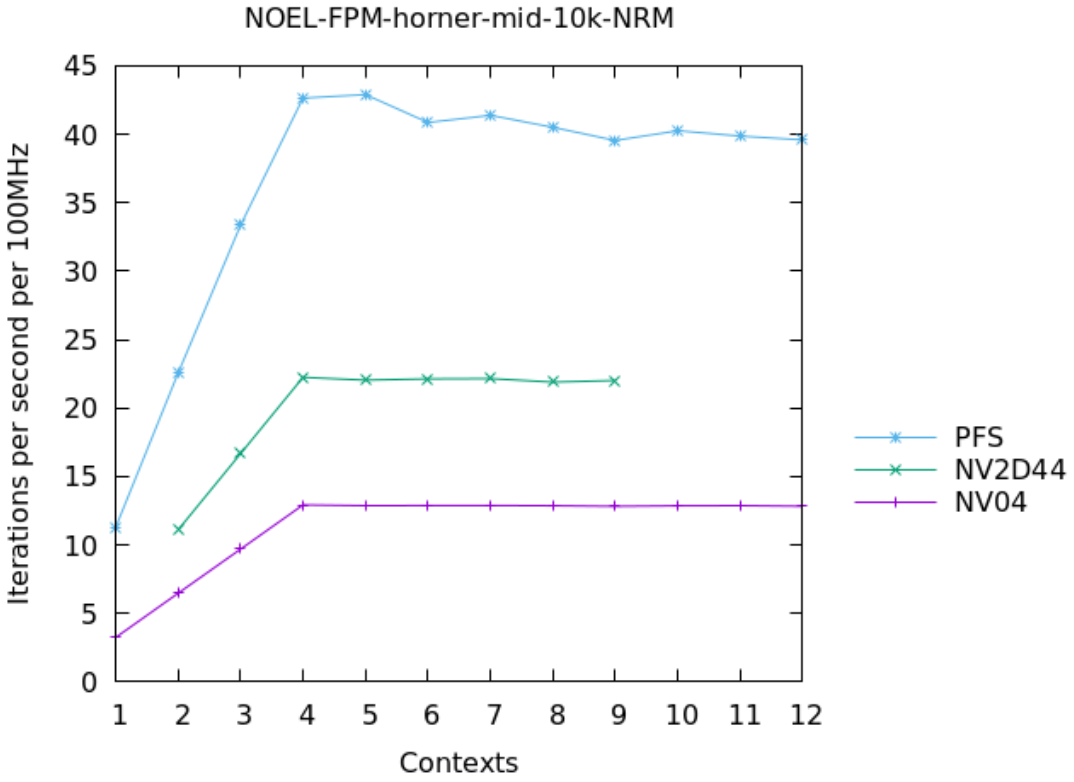

Figure 43: NOEL - FPMark - horner-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

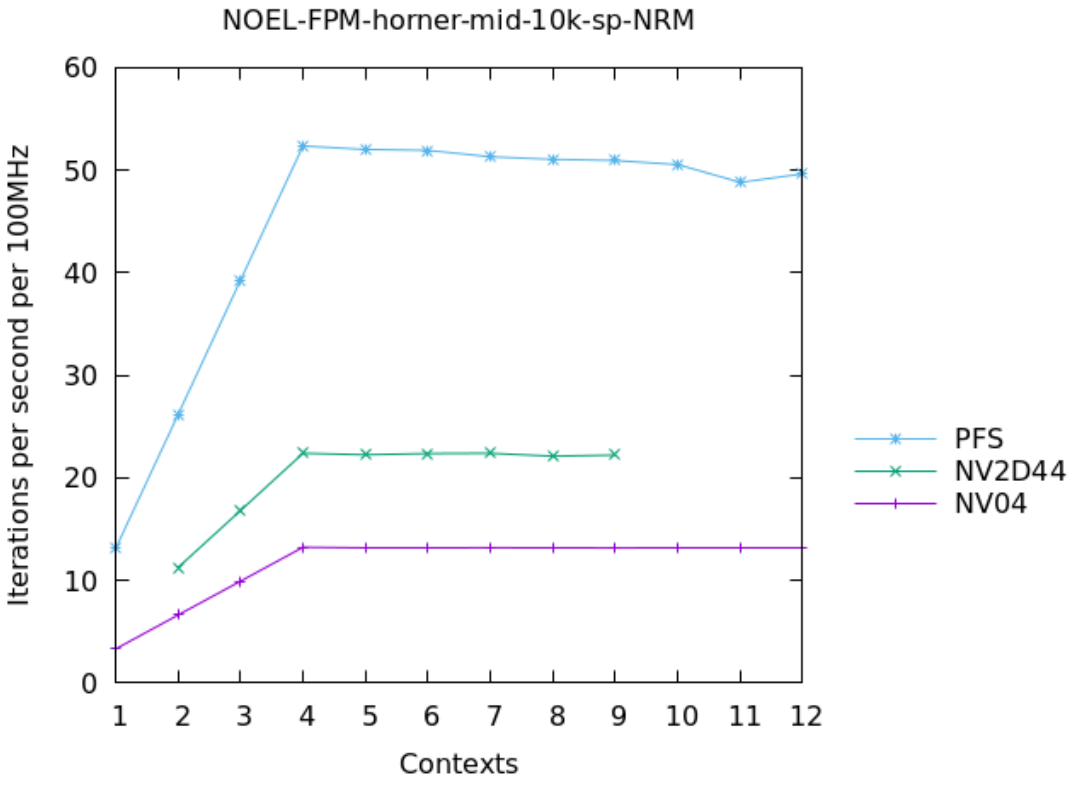

Figure 44: NOEL - FPMark - horner-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

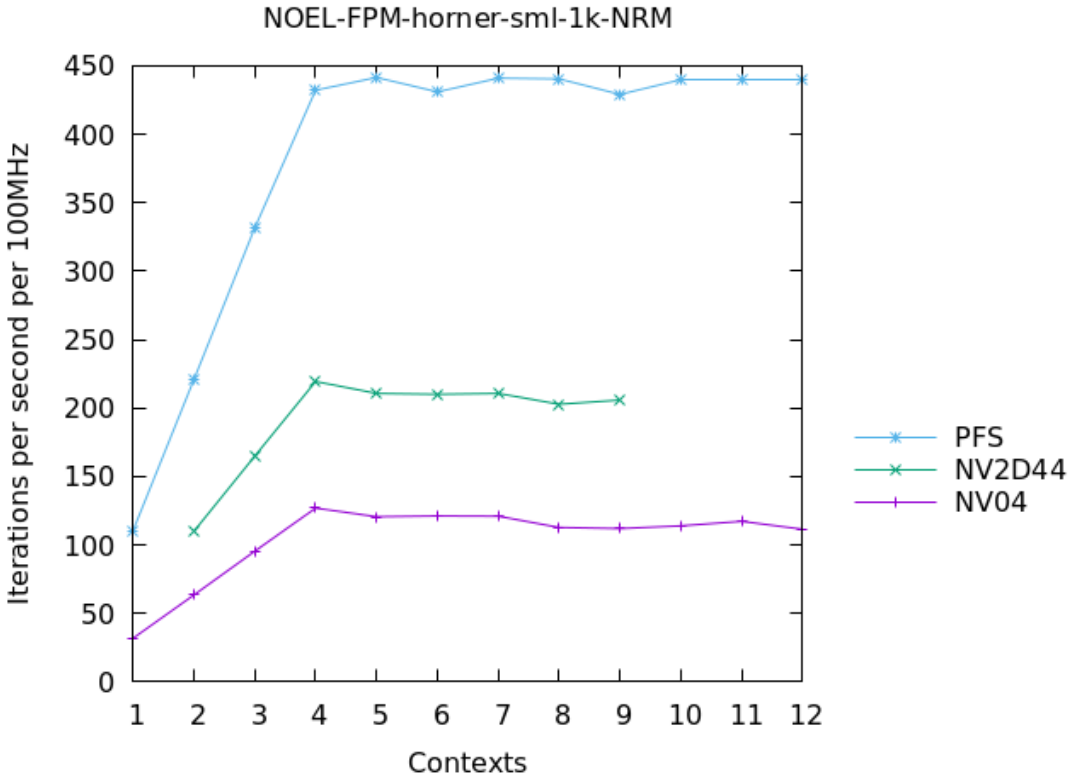

Figure 45: NOEL - FPMark - horner-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

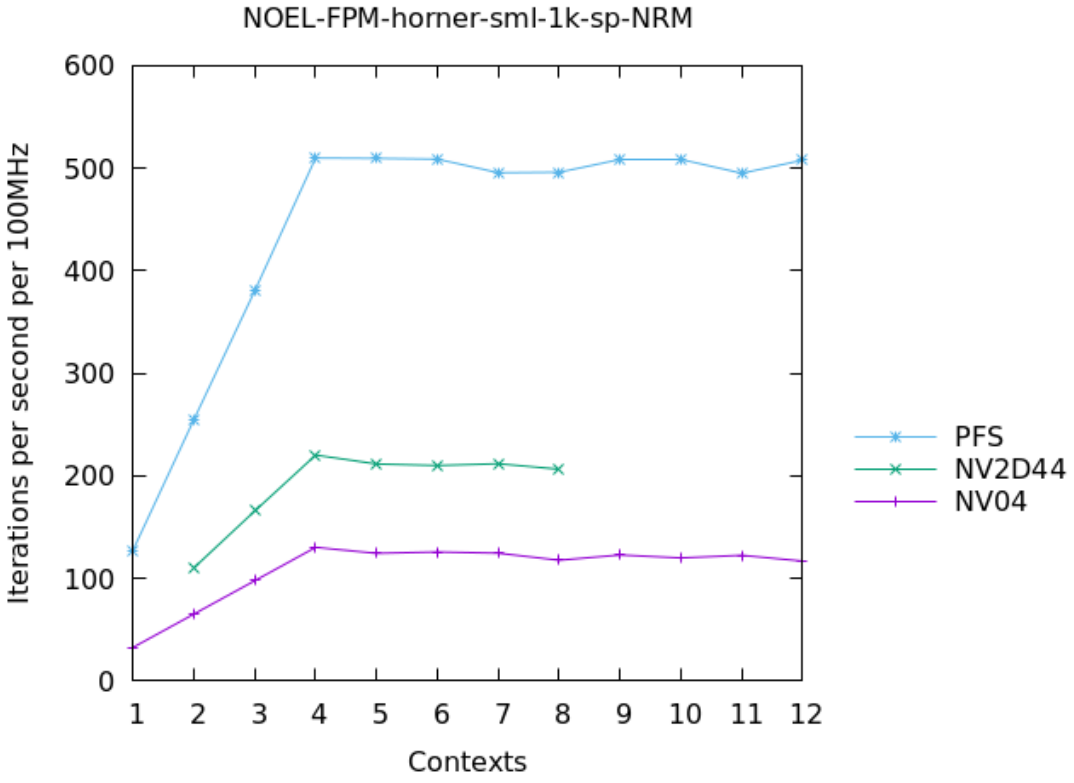

Figure 46: NOEL - FPMark - horner-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

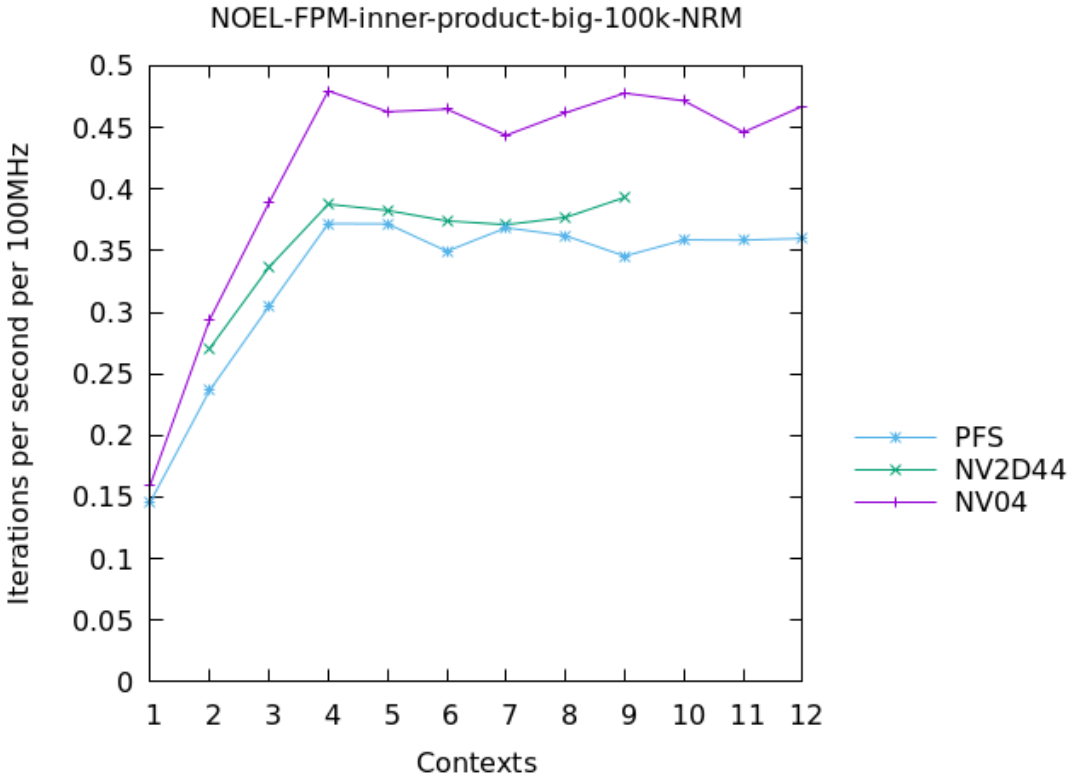

Figure 47: NOEL - FPMark - inner-product-big-100k, iterations per second at 100MHz. Higher values denote better performance.

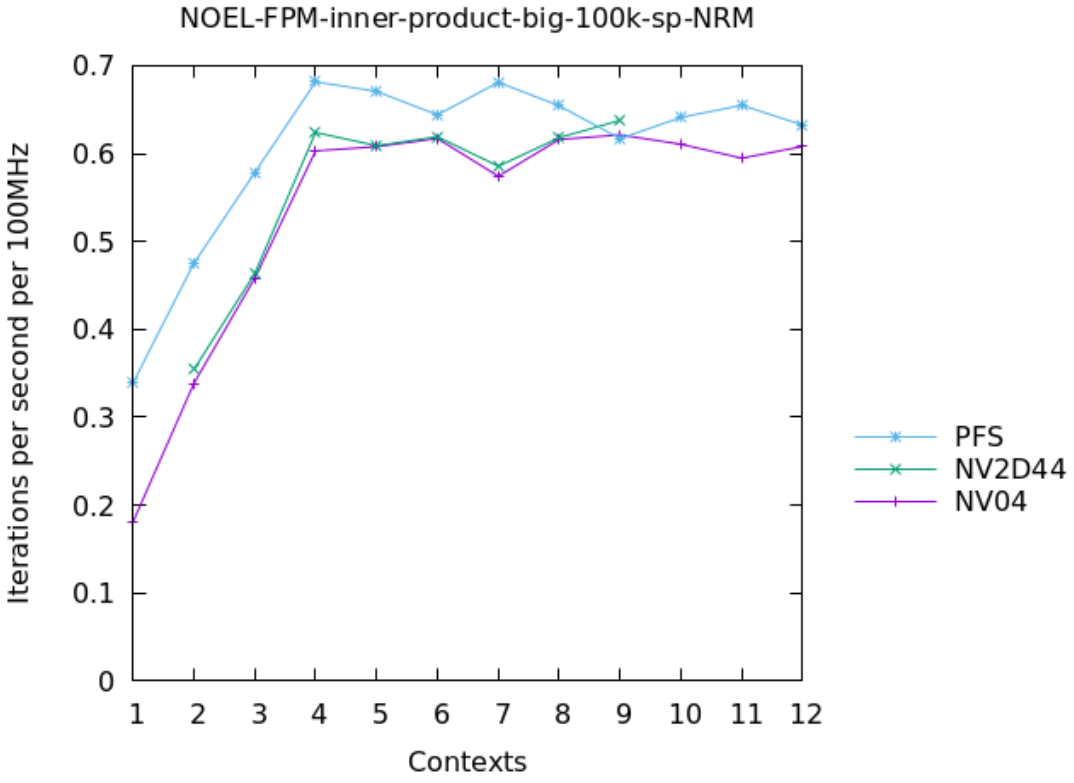

Figure 48: NOEL - FPMark - inner-product-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

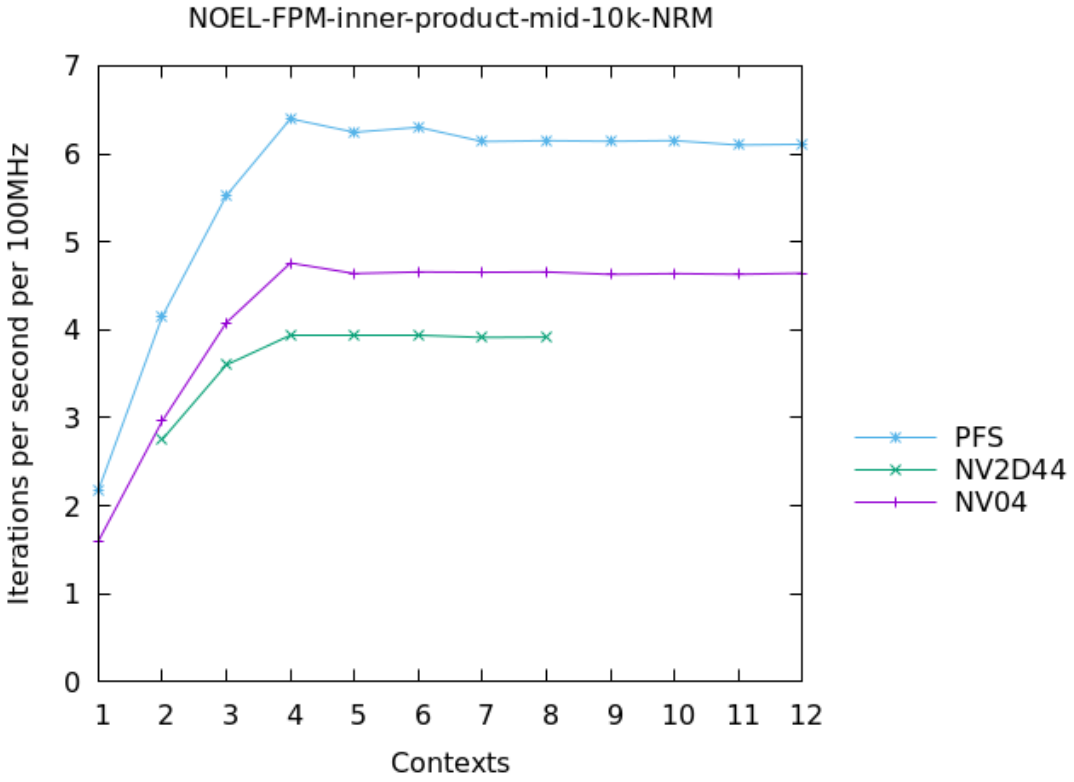

Figure 49: NOEL - FPMark - inner-product-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

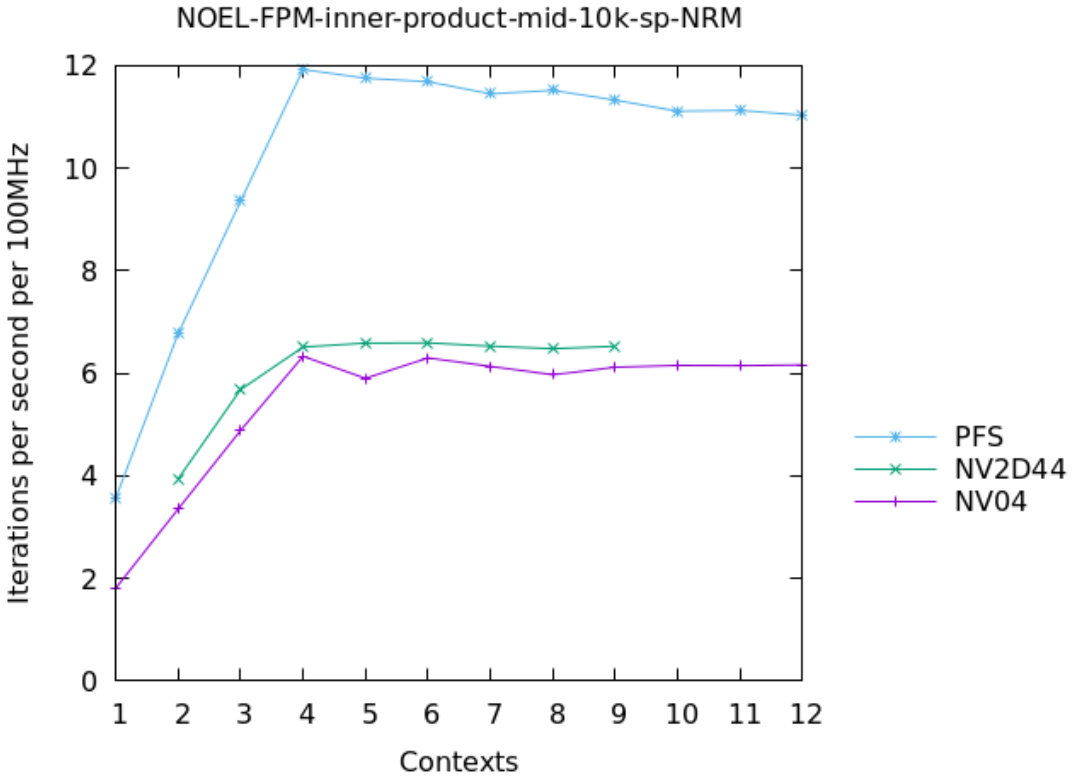

Figure 50: NOEL - FPMark - inner-product-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

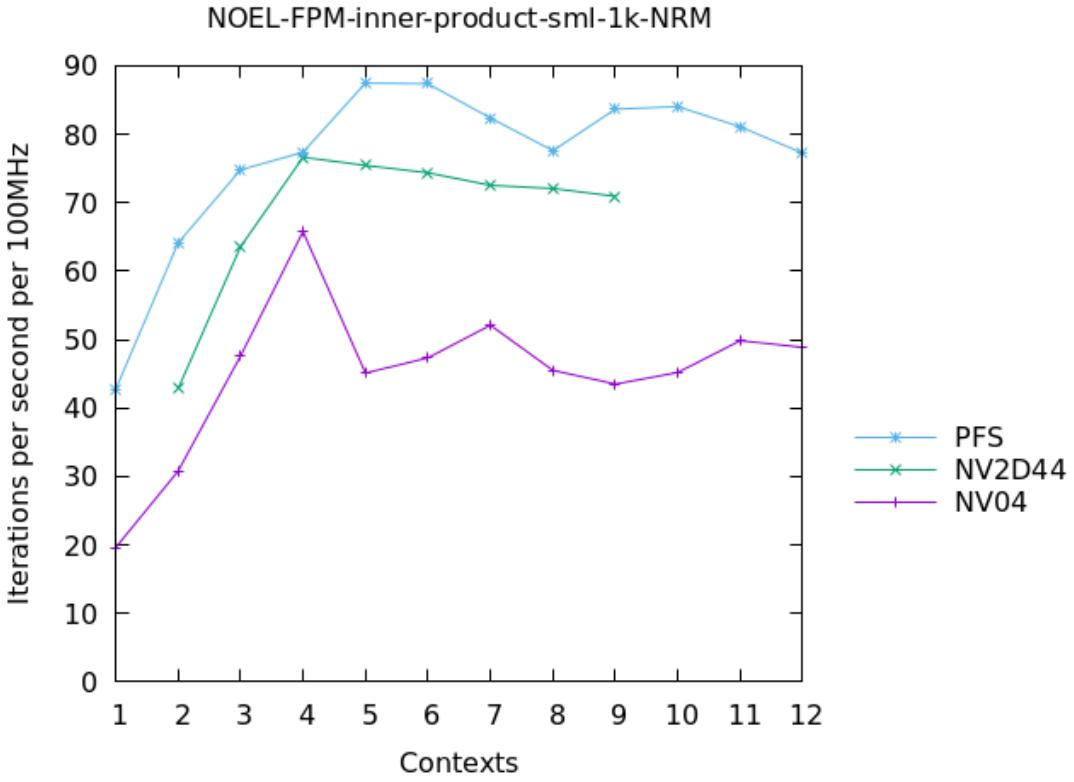

Figure 51: NOEL - FPMark - inner-product-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

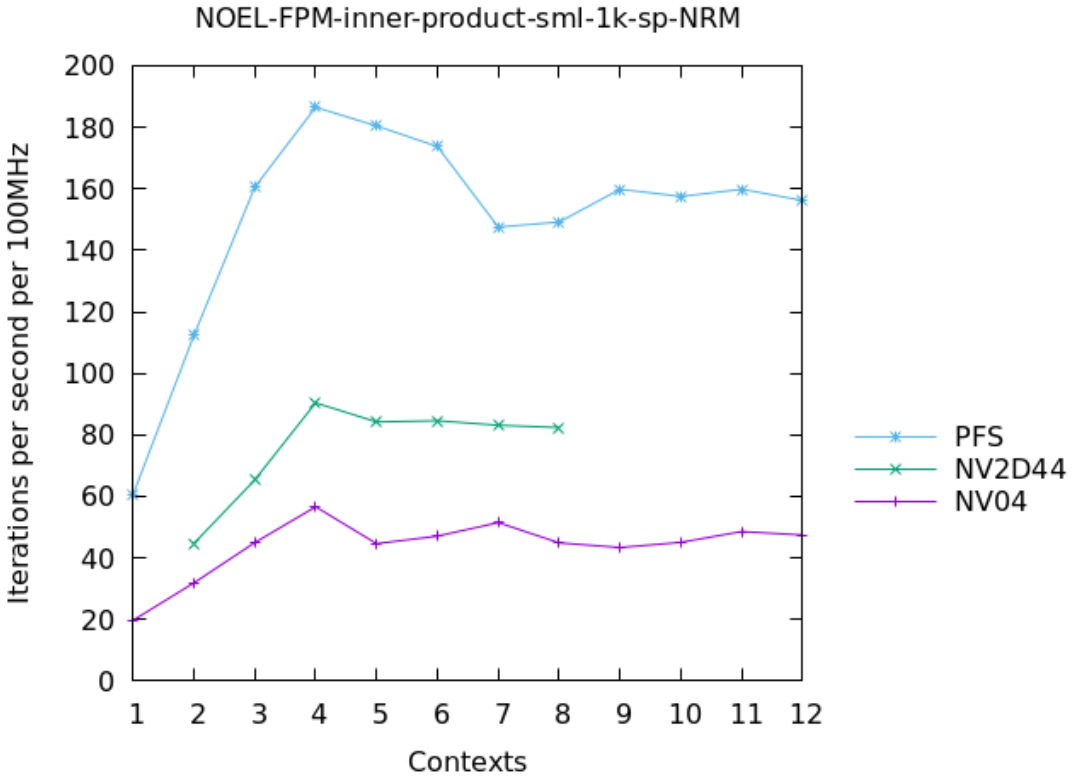

Figure 52: NOEL - FPMark - inner-product-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

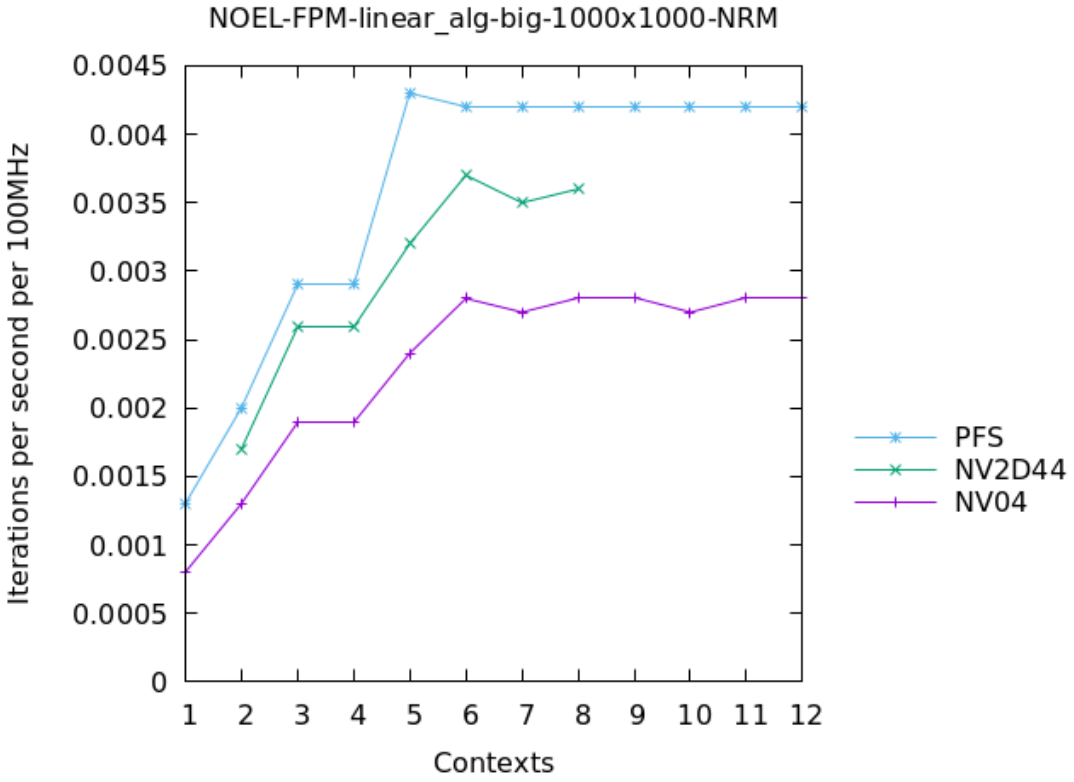

Figure 53: NOEL - FPMark - linear_alg-big-1000x1000, iterations per second at 100MHz. Higher values denote better performance.

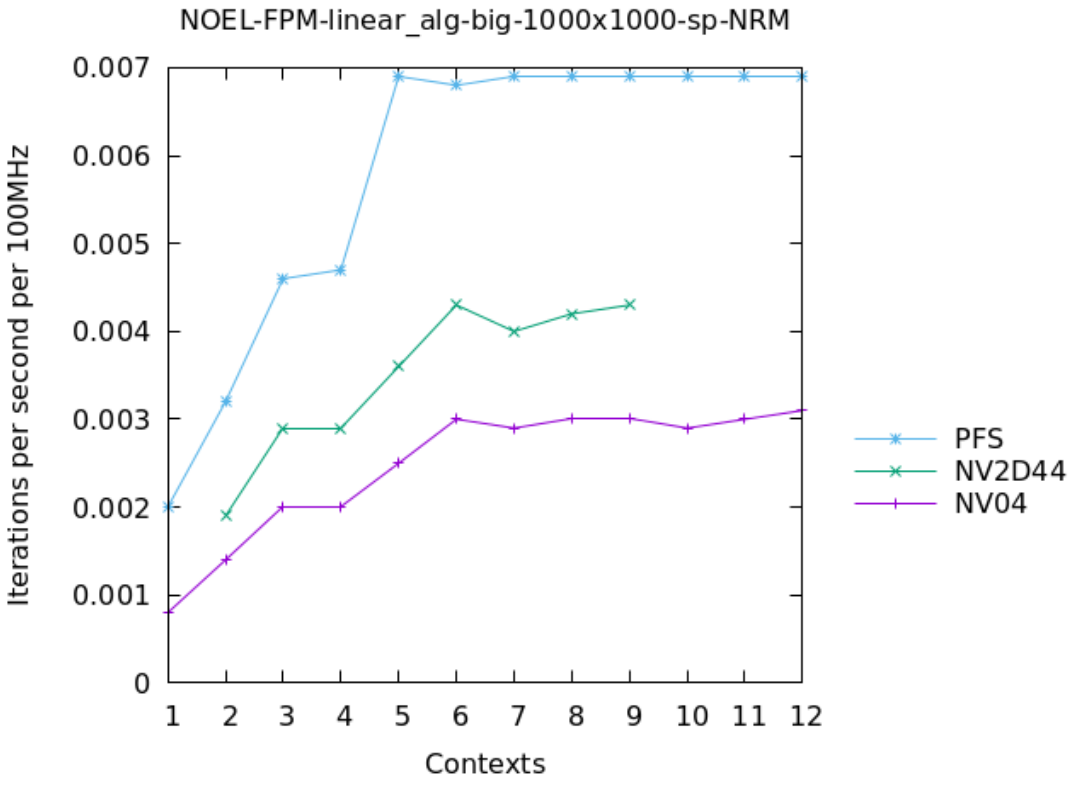

Figure 54: NOEL - FPMark - linear_alg-big-1000x1000-sp, iterations per second at 100MHz. Higher values denote better performance.

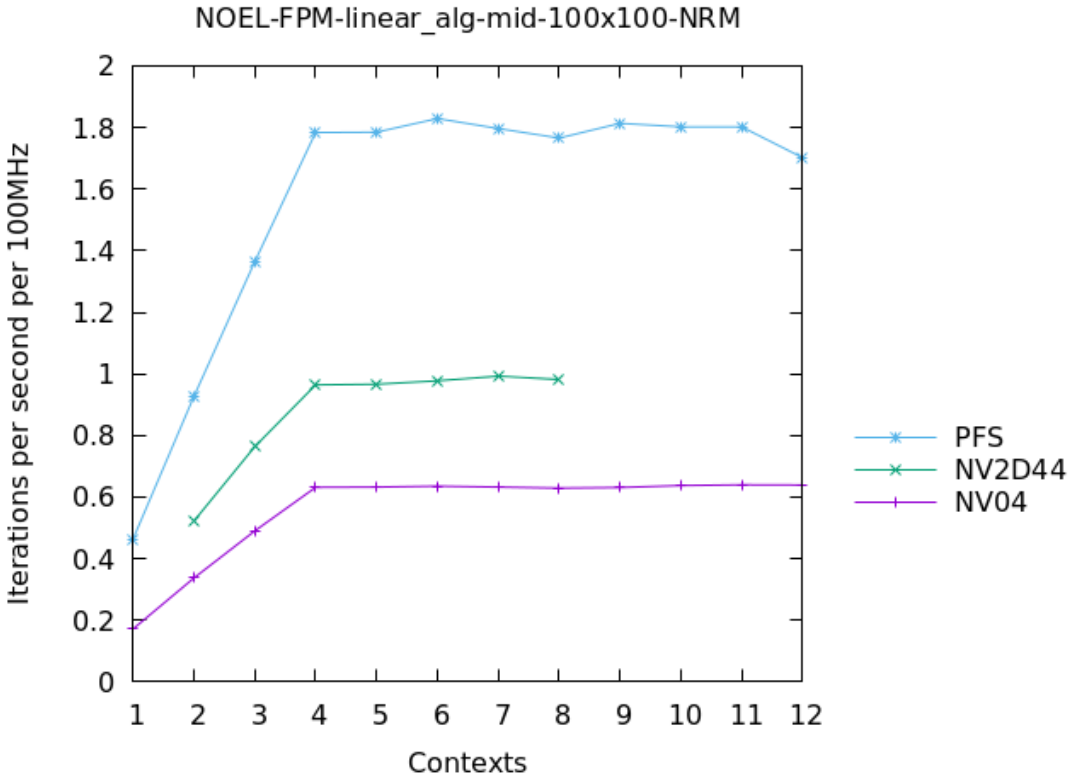

Figure 55: NOEL - FPMark - linear_alg-mid-100x100, iterations per second at 100MHz. Higher values denote better performance.

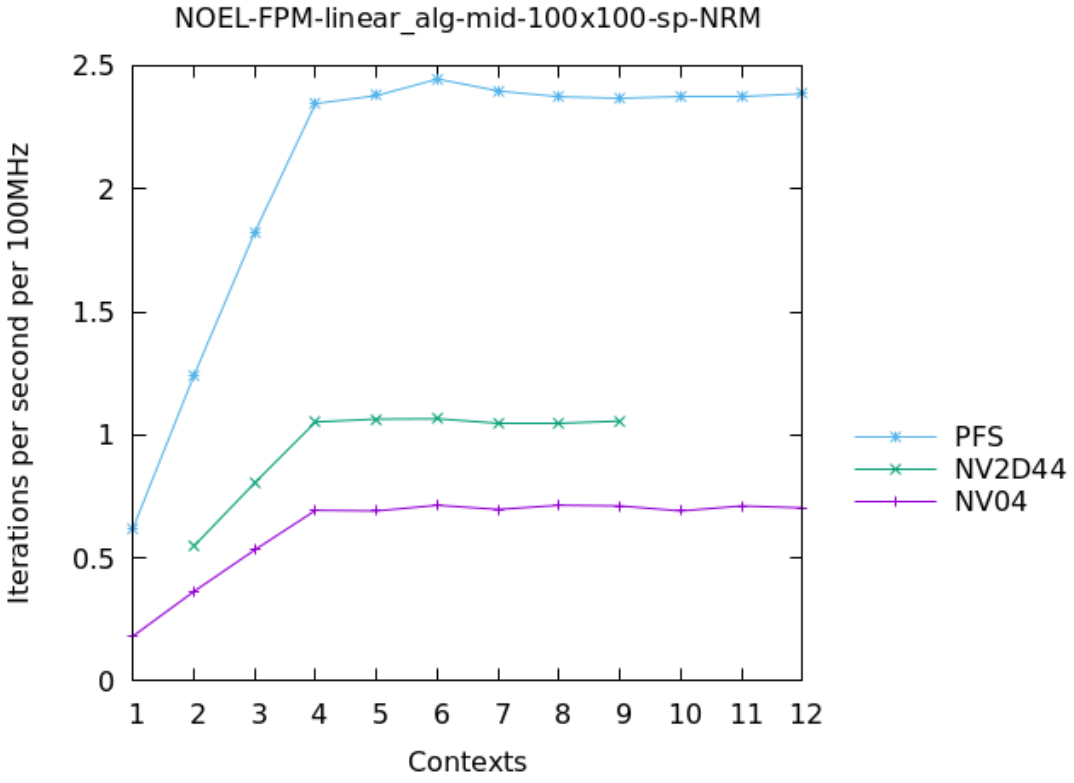

Figure 56: NOEL - FPMark - linear_alg-mid-100x100-sp, iterations per second at 100MHz. Higher values denote better performance.

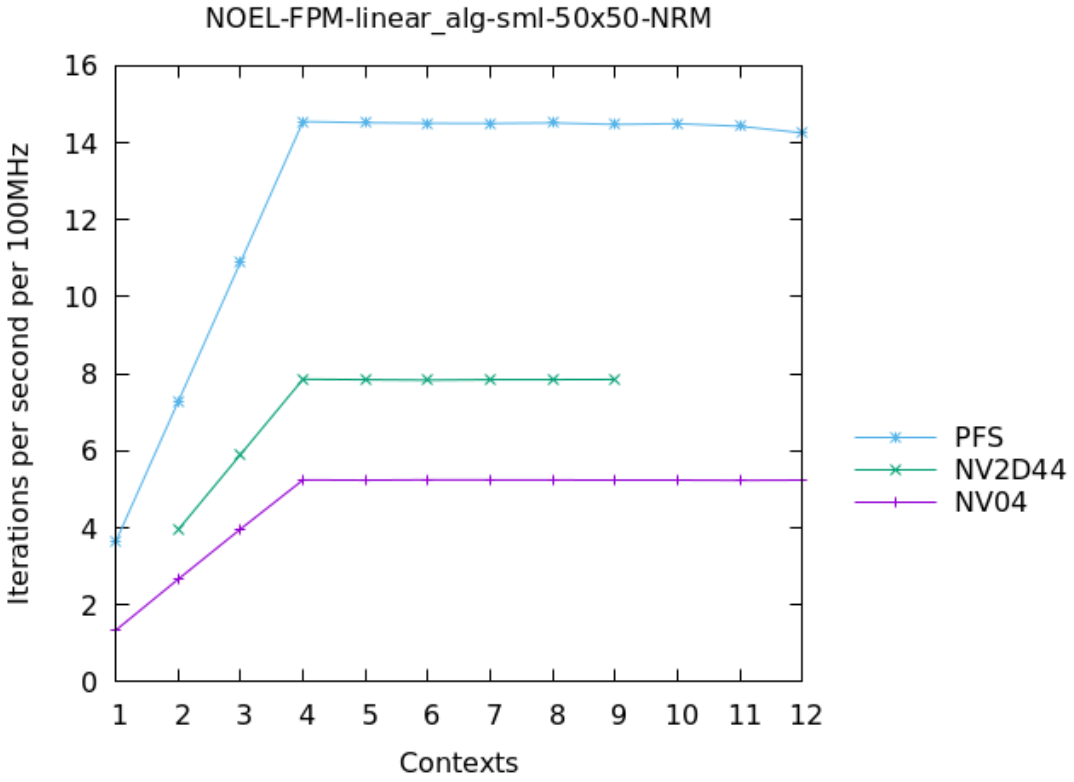

Figure 57: NOEL - FPMark - linear_alg-sml-50x50, iterations per second at 100MHz. Higher values denote better performance.

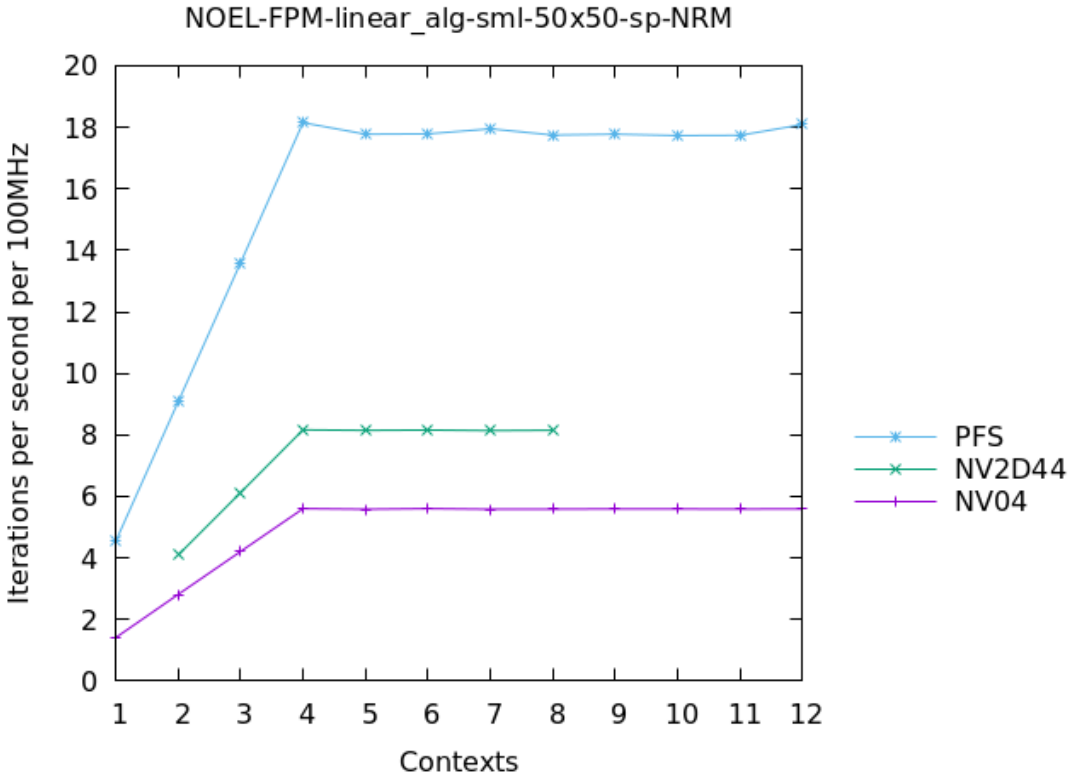

Figure 58: NOEL - FPMark - linear_alg-sml-50x50-sp, iterations per second at 100MHz. Higher values denote better performance.

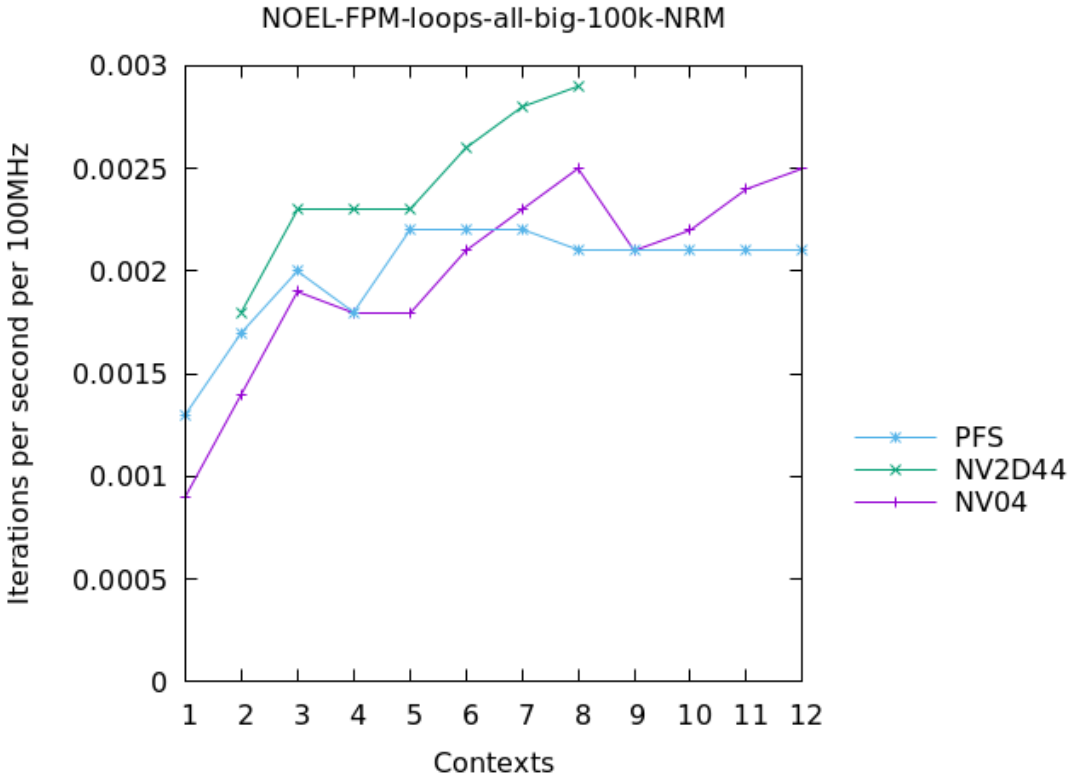

Figure 59: NOEL - FPMark - loops-all-big-100k, iterations per second at 100MHz. Higher values denote better performance.

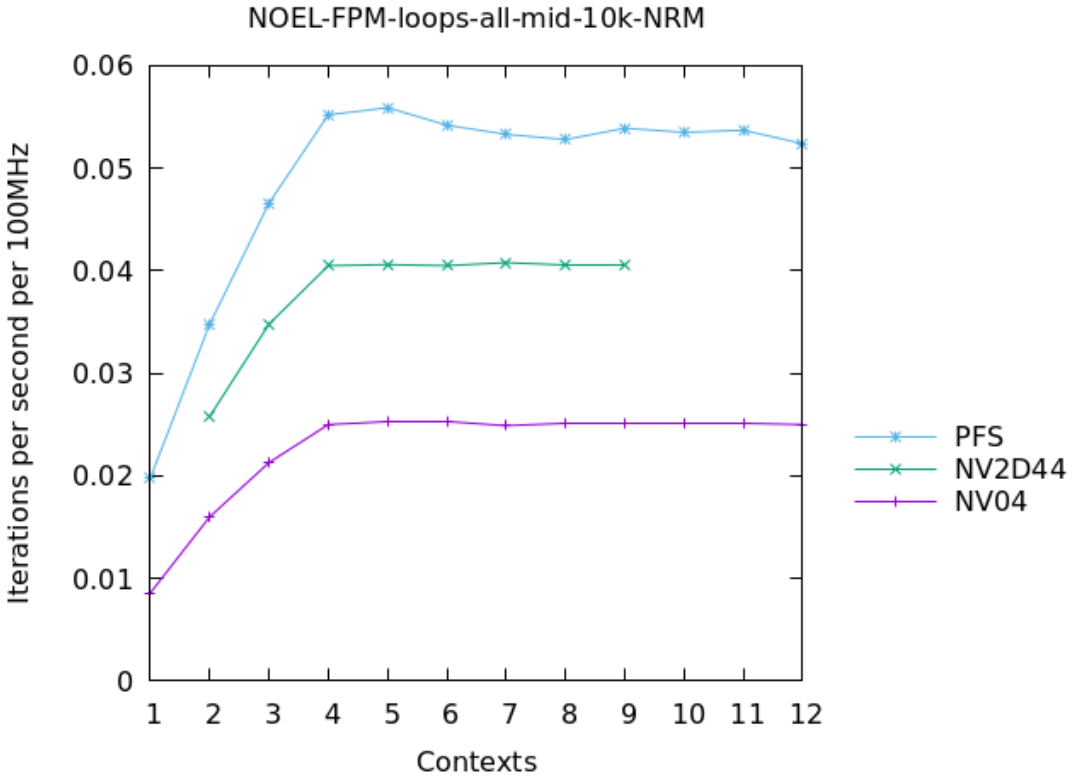

Figure 60: NOEL - FPMark - loops-all-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

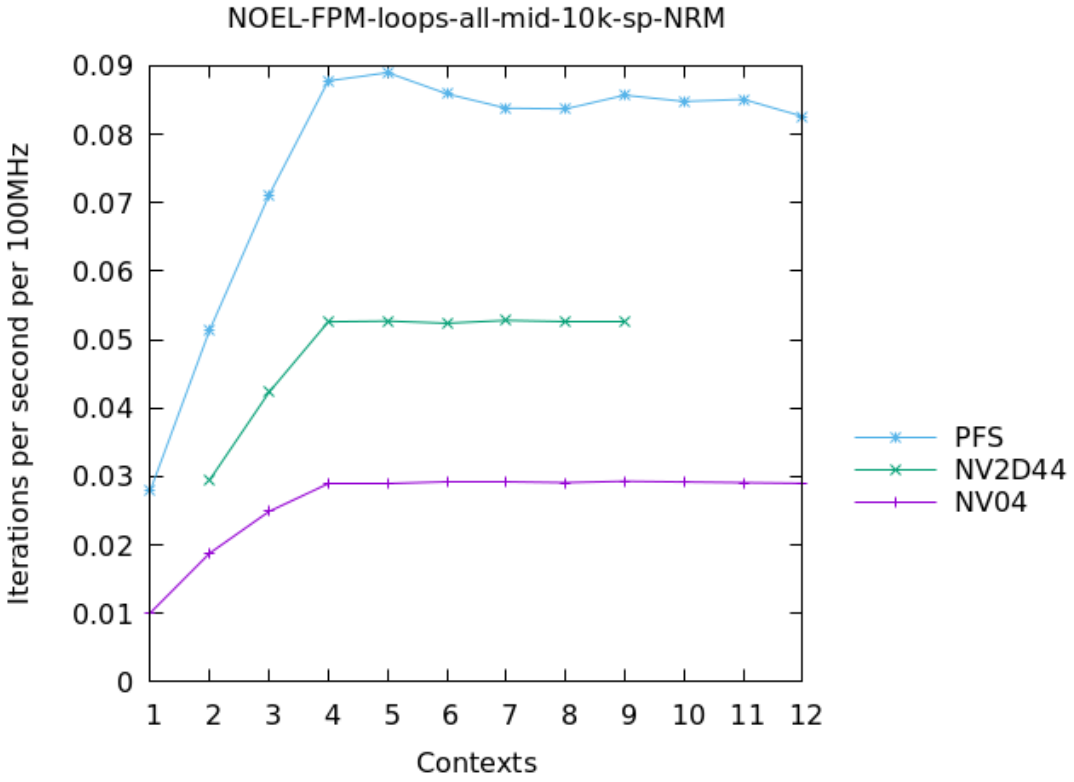

Figure 61: NOEL - FPMark - loops-all-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

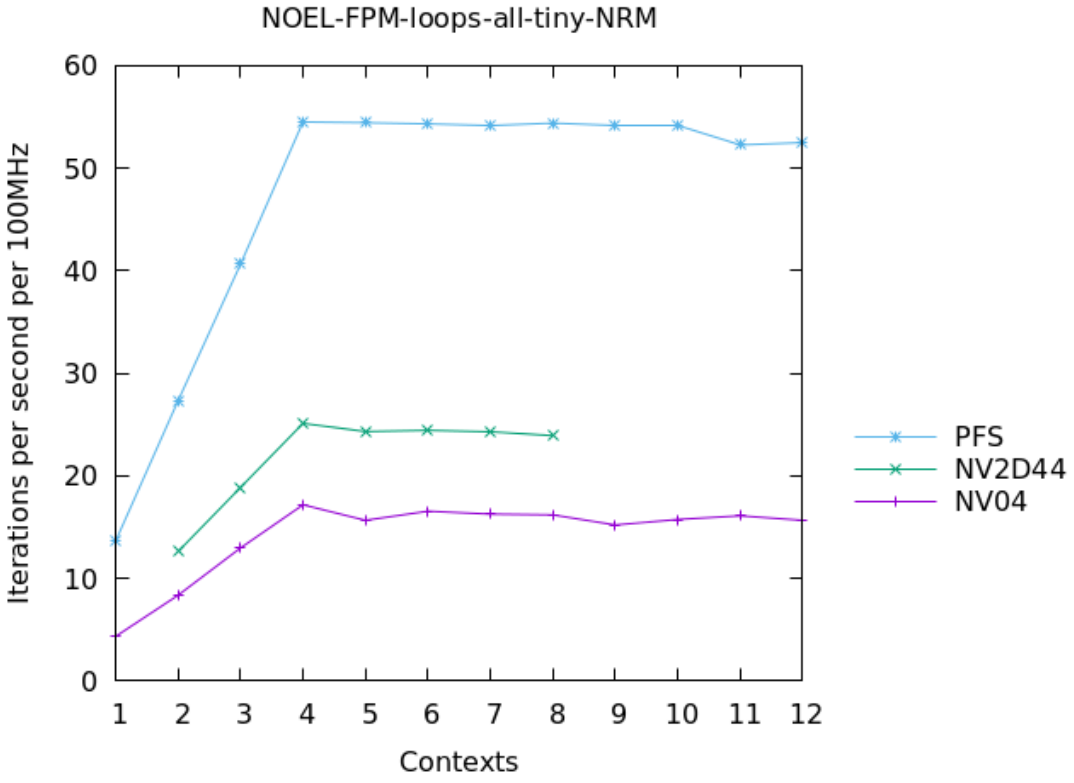

Figure 62: NOEL - FPMark - loops-all-tiny, iterations per second at 100MHz. Higher values denote better performance.

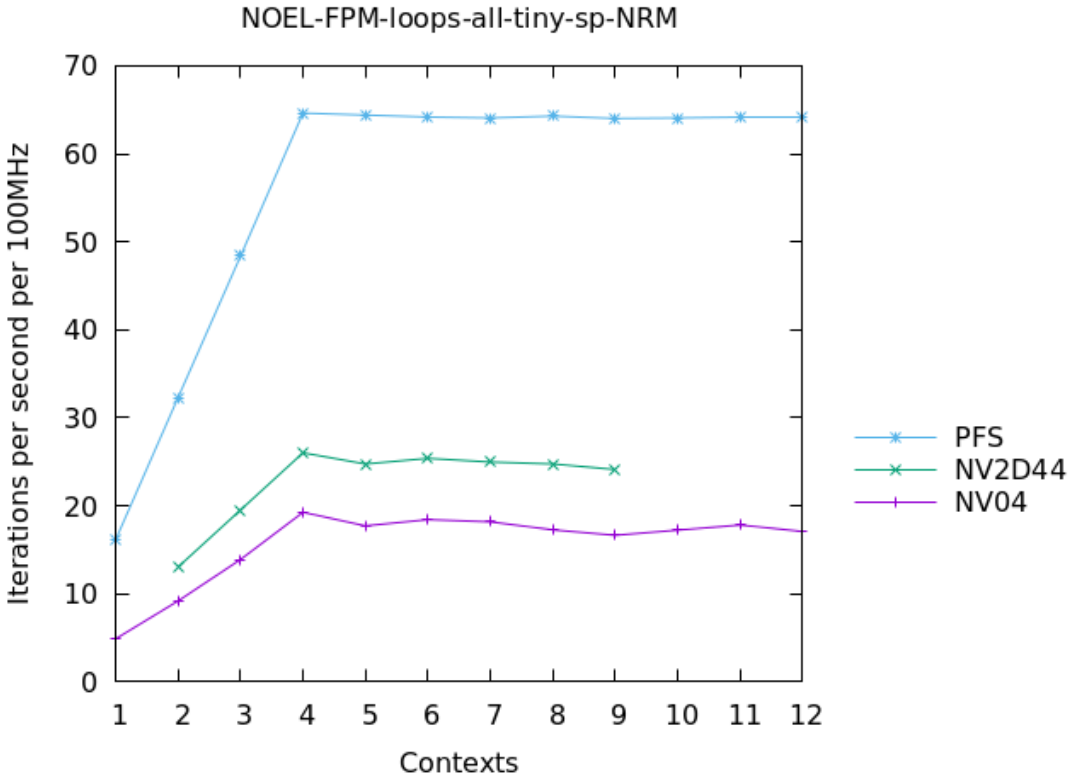

Figure 63: NOEL - FPMark - loops-all-tiny-sp, iterations per second at 100MHz. Higher values denote better performance.

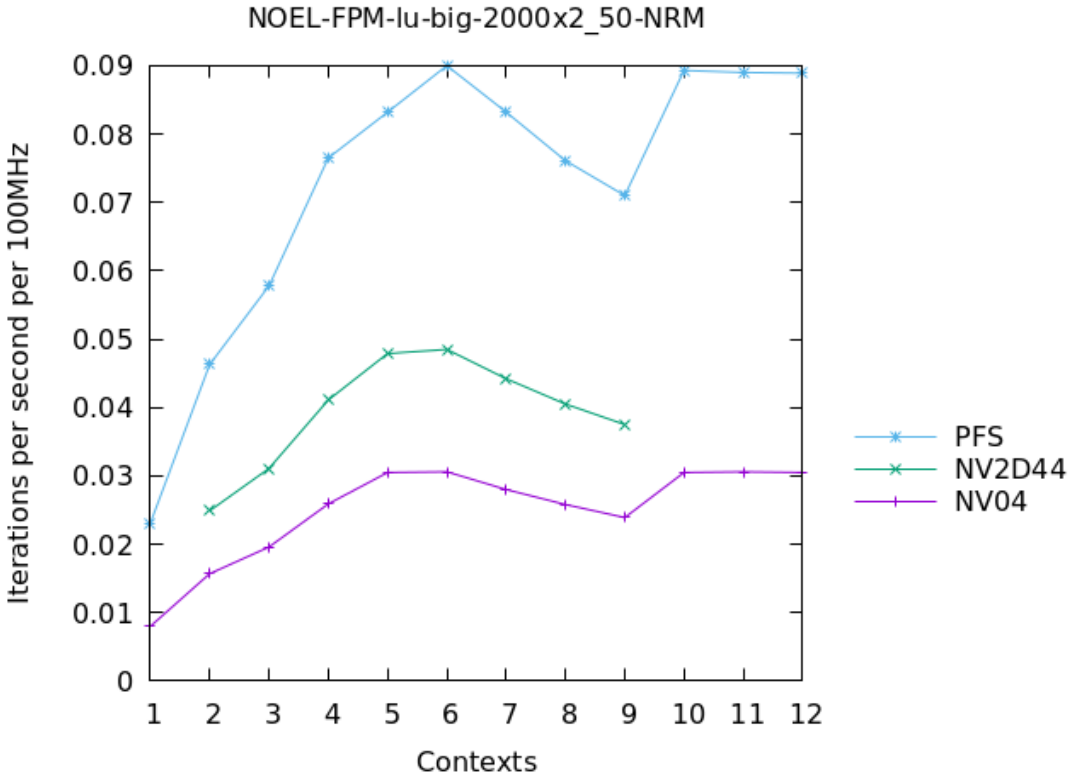

Figure 64: NOEL - FPMark - lu-big-2000x2_50, iterations per second at 100MHz. Higher values denote better performance.

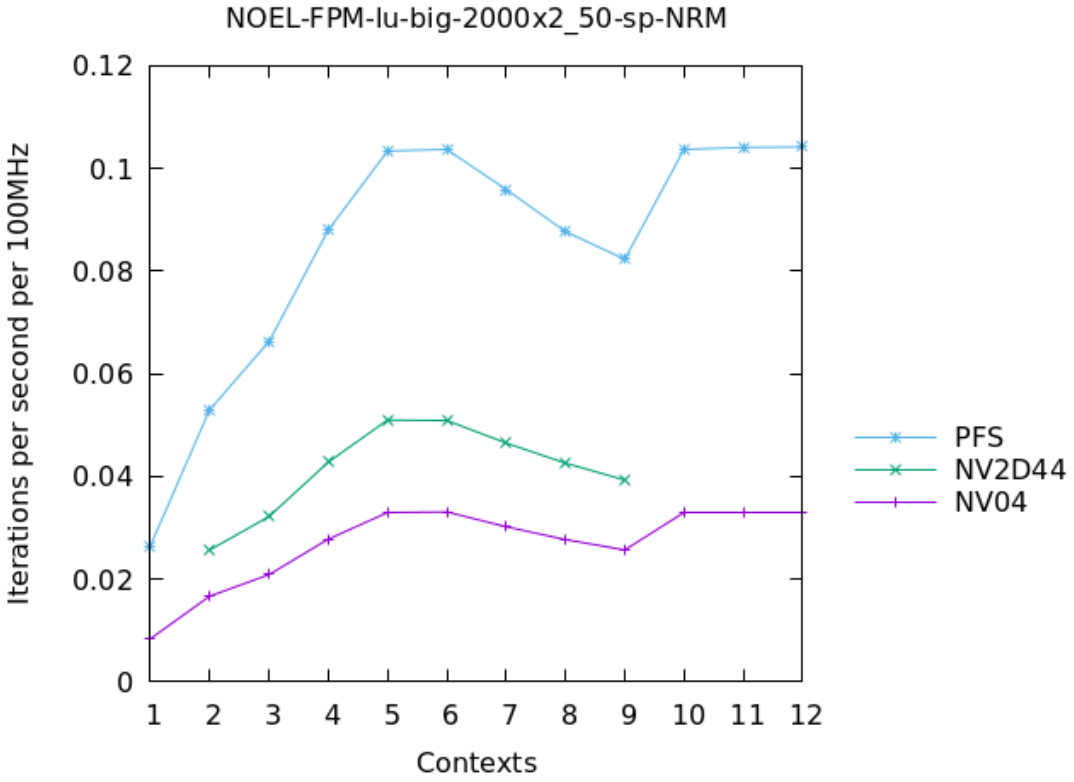

Figure 65: NOEL - FPMark - lu-big-2000x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

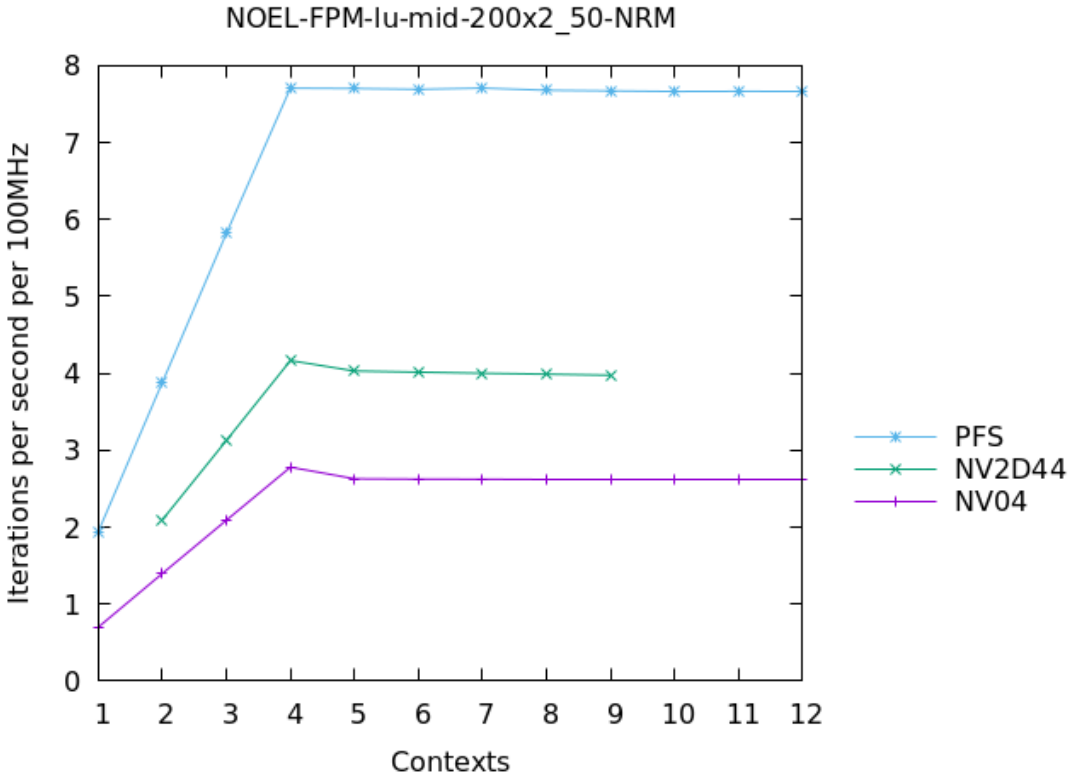

Figure 66: NOEL - FPMark - lu-mid-200x2_50, iterations per second at 100MHz. Higher values denote better performance.

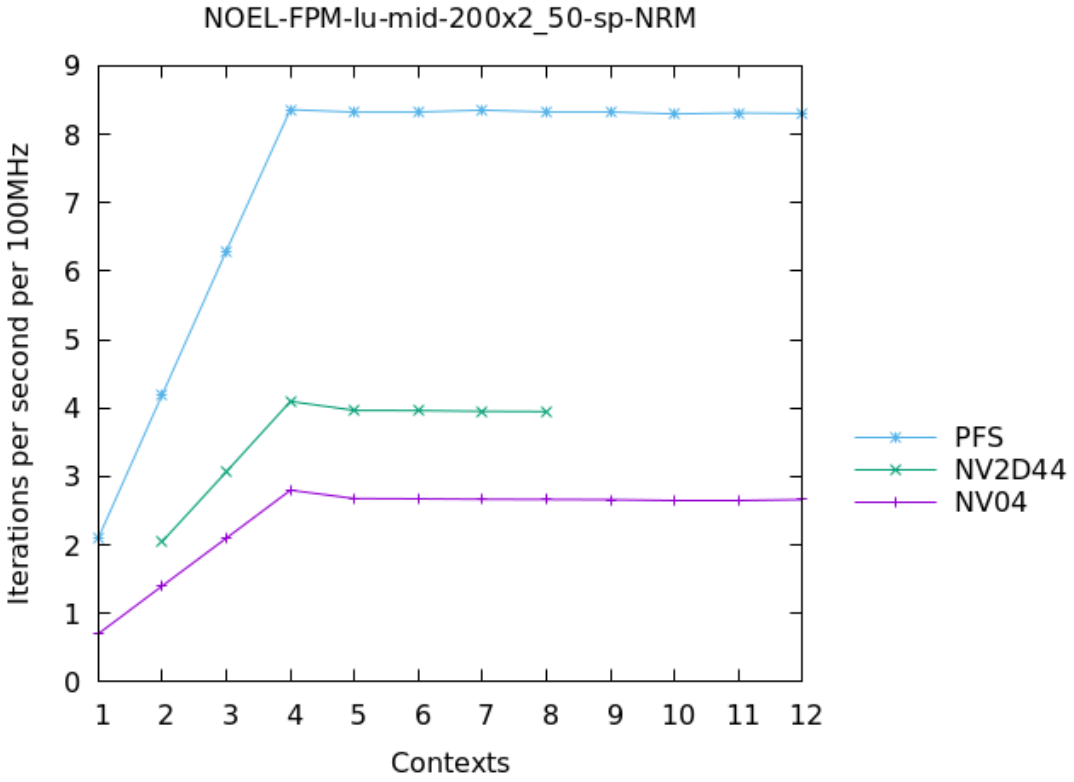

Figure 67: NOEL - FPMark - lu-mid-200x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

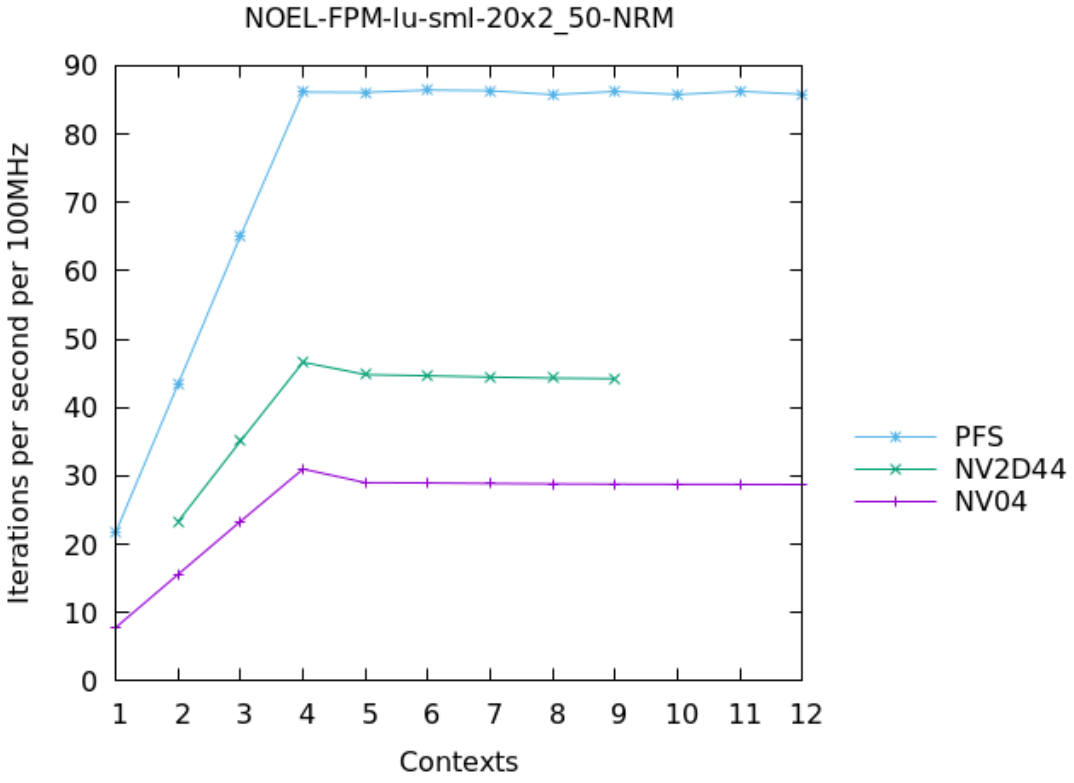

Figure 68: NOEL - FPMark - lu-sml-20x2_50, iterations per second at 100MHz. Higher values denote better performance.

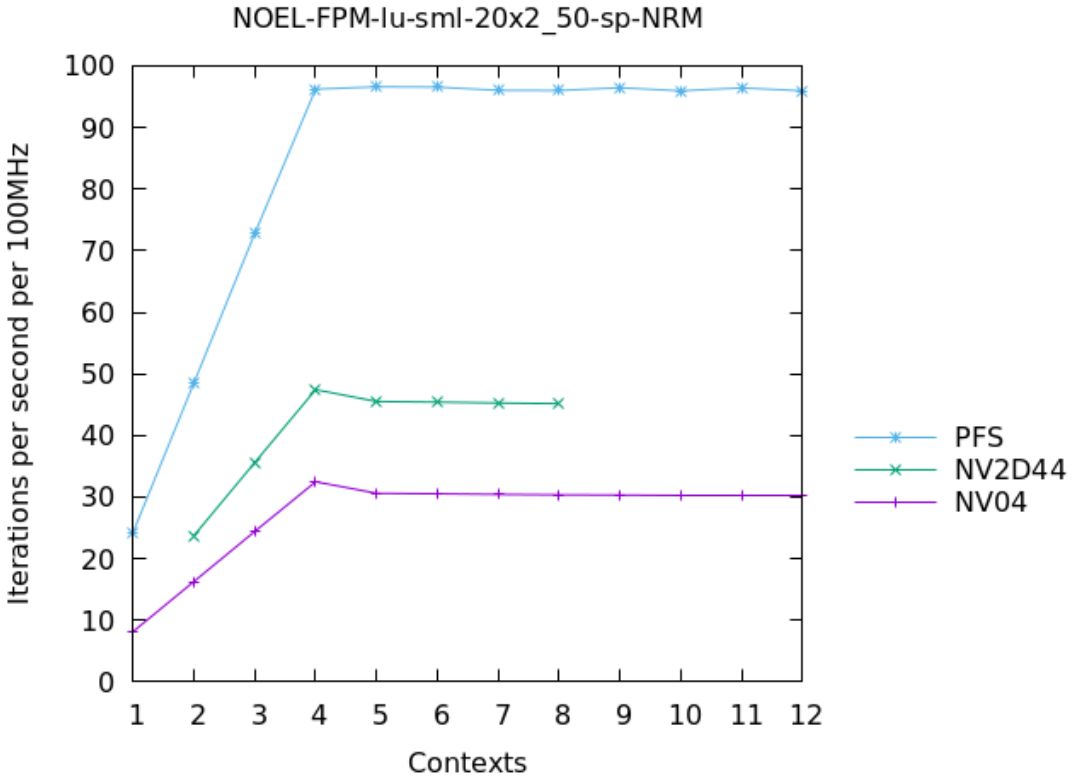

Figure 69: NOEL - FPMark - lu-sml-20x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

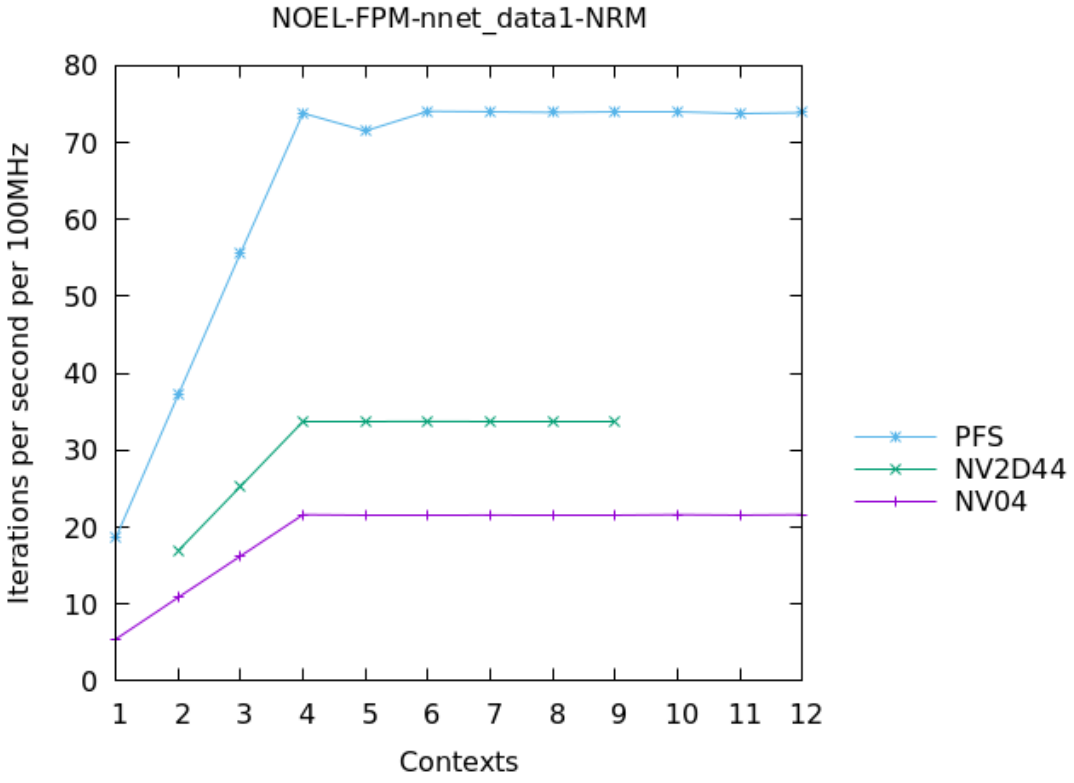

Figure 70: NOEL - FPMark - nnet_data1, iterations per second at 100MHz. Higher values denote better performance.

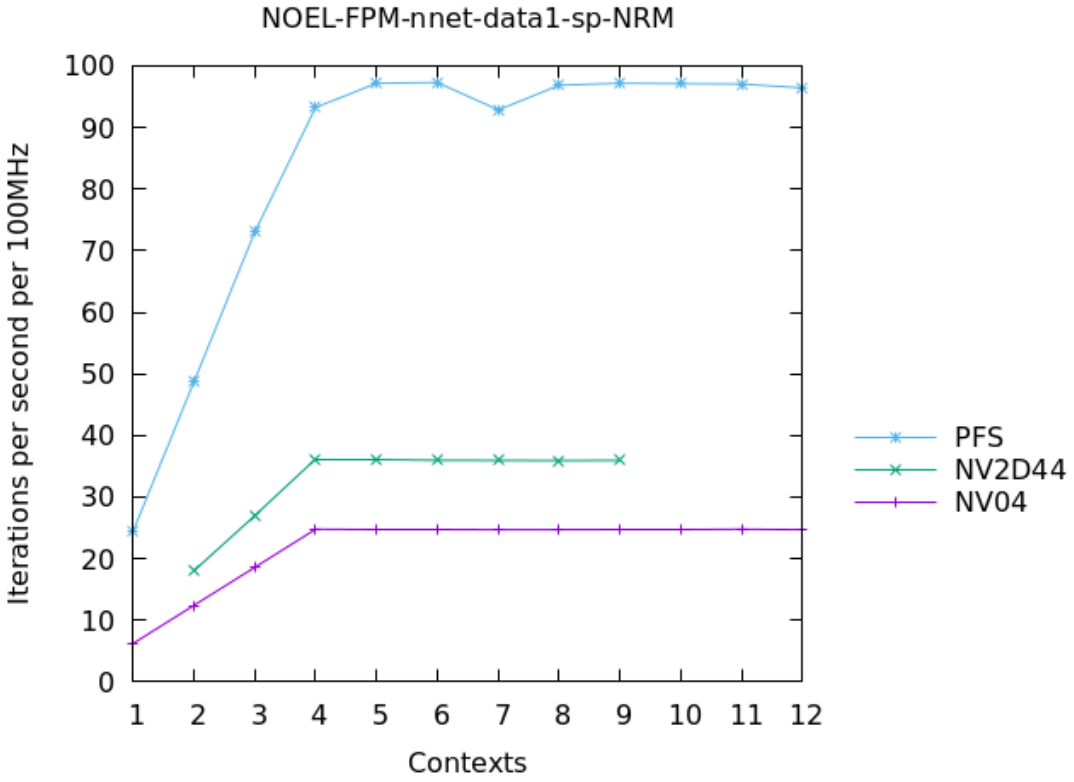

Figure 71: NOEL - FPMark - nnet-data1-sp, iterations per second at 100MHz. Higher values denote better performance.

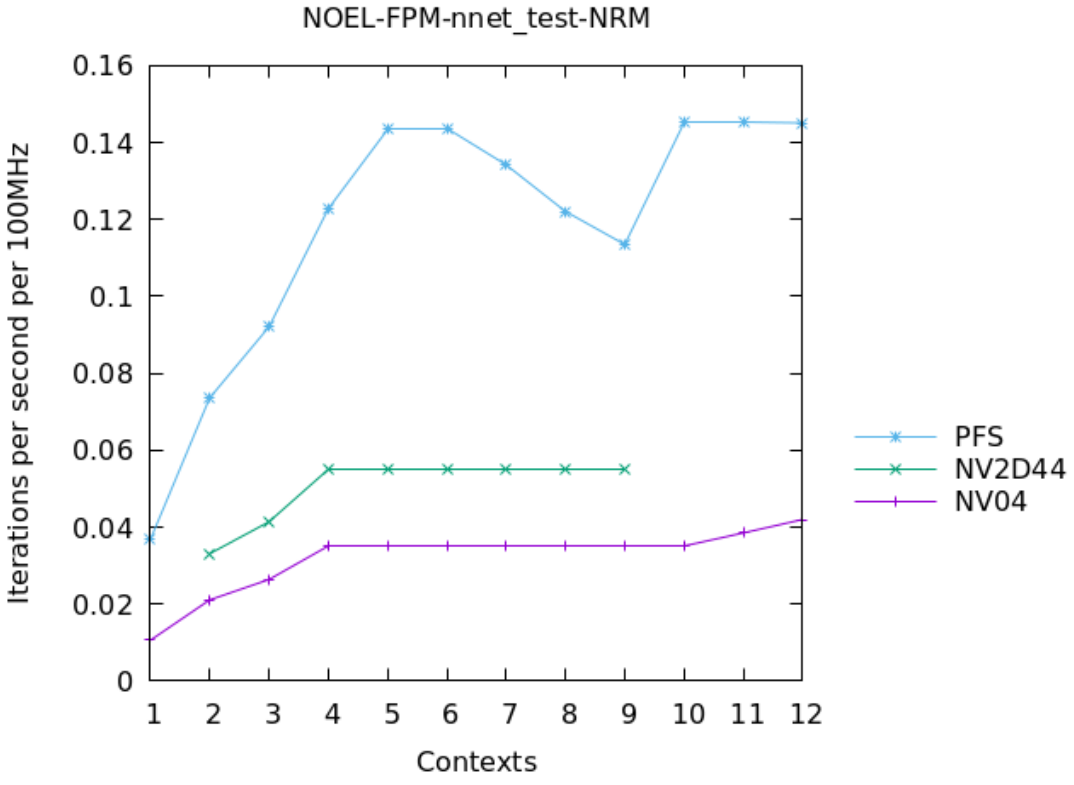

Figure 72: NOEL - FPMark - nnet_test, iterations per second at 100MHz. Higher values denote better performance.

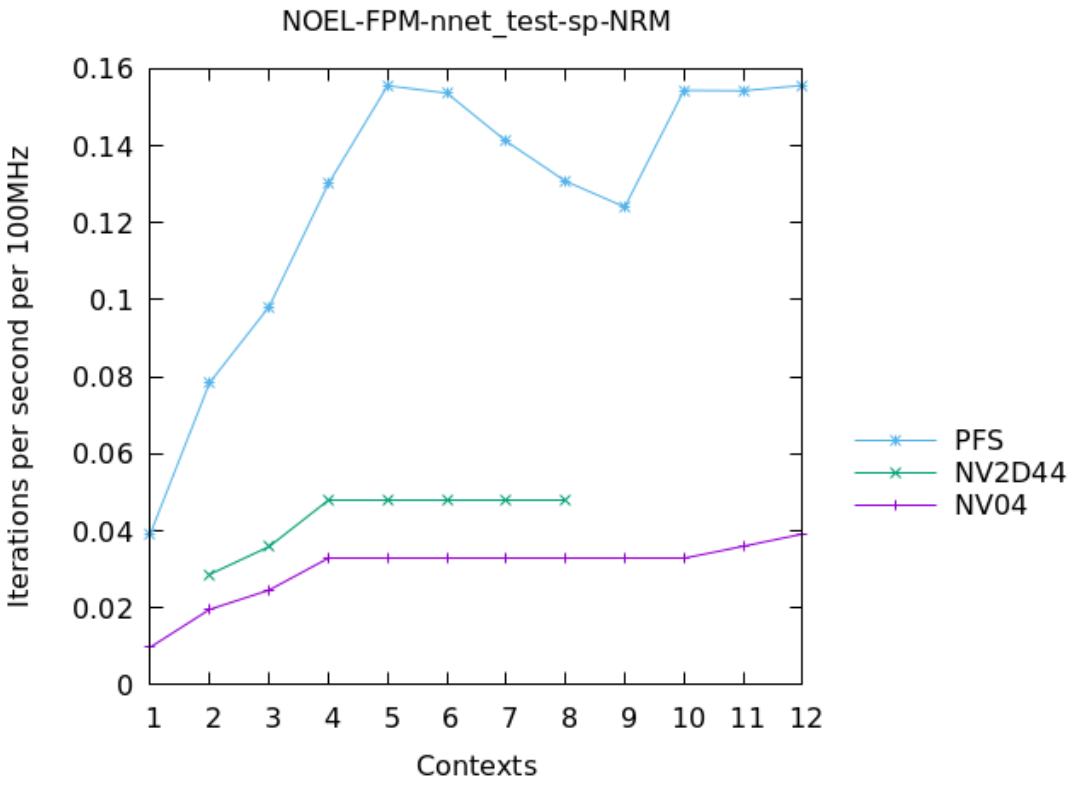

Figure 73: NOEL - FPMark - nnet_test-sp, iterations per second at 100MHz. Higher values denote better performance.

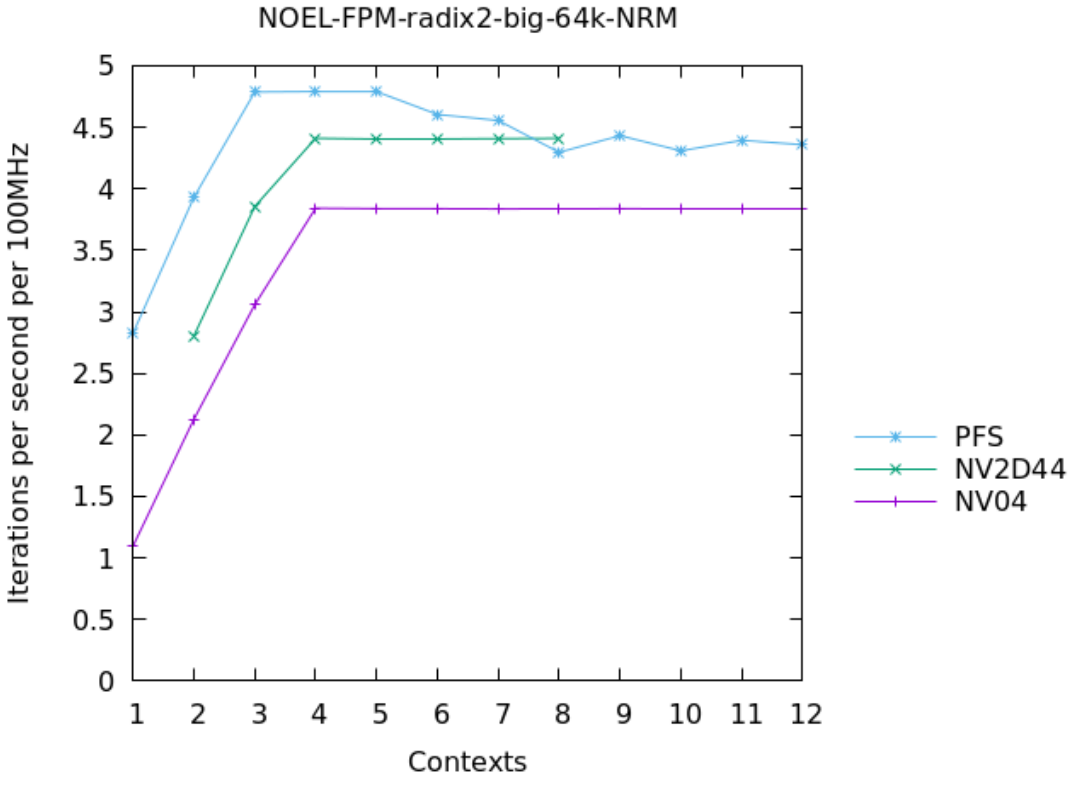

Figure 74: NOEL - FPMark - radix2-big-64k, iterations per second at 100MHz. Higher values denote better performance.

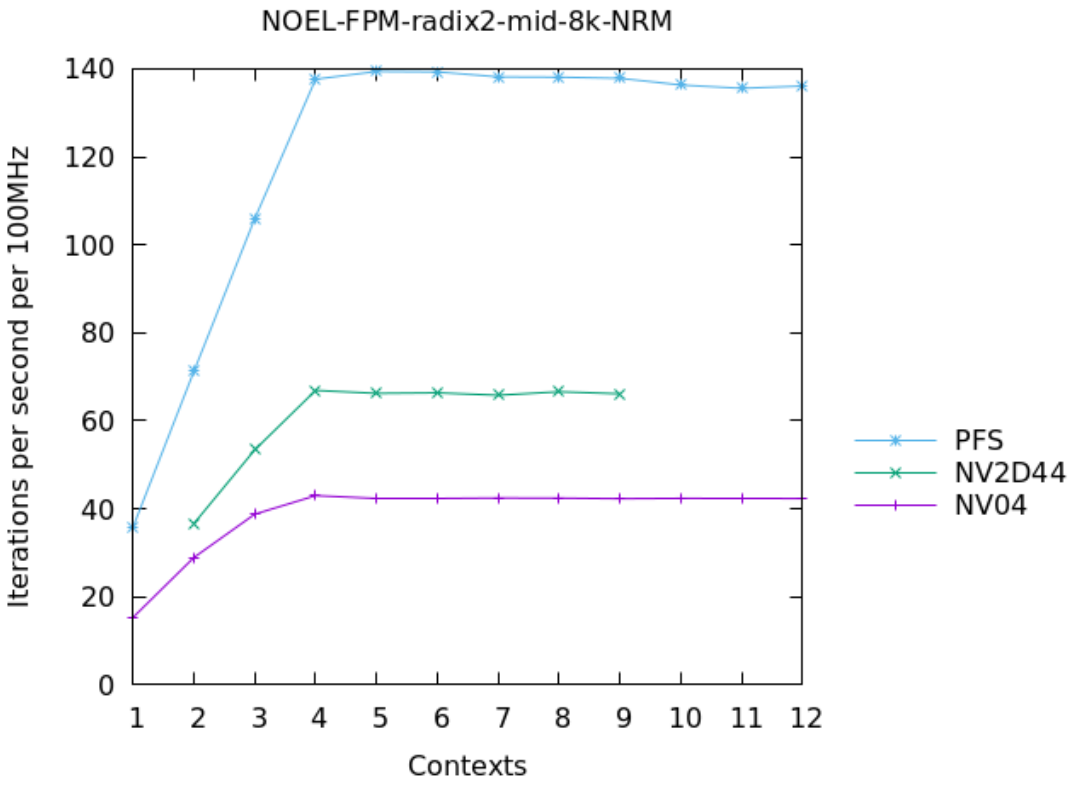

Figure 75: NOEL - FPMark - radix2-mid-8k, iterations per second at 100MHz. Higher values denote better performance.

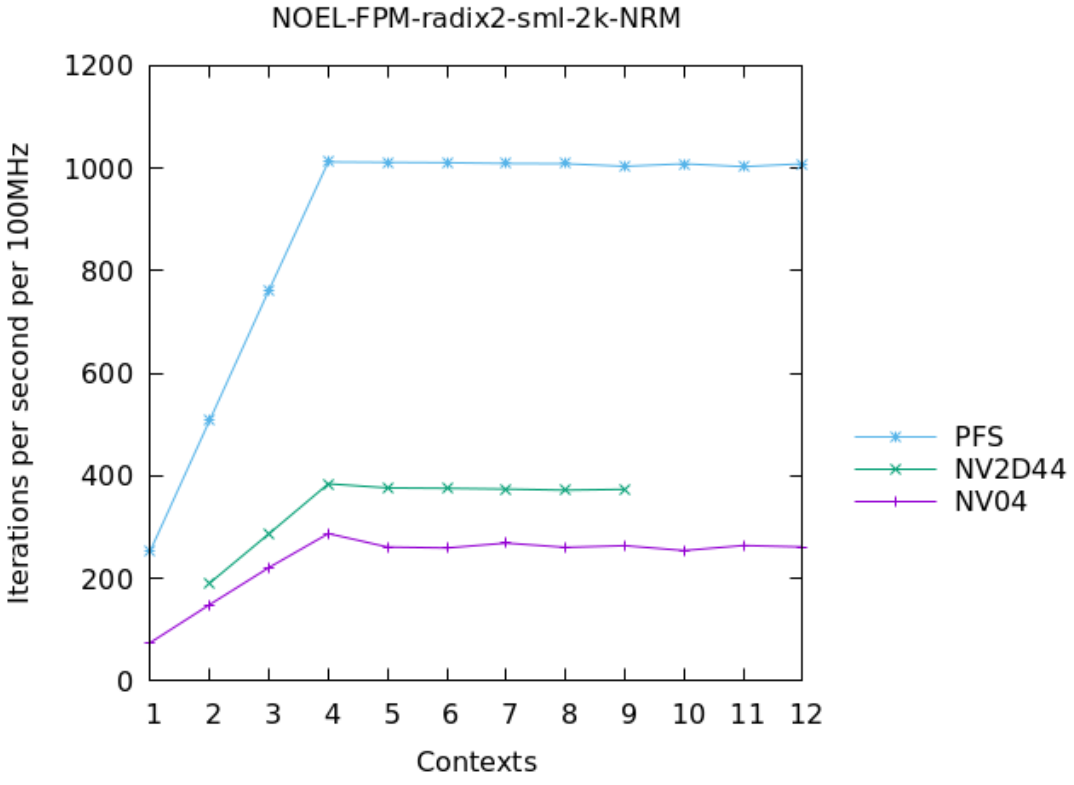

Figure 76: NOEL - FPMark - radix2-sml-2k, iterations per second at 100MHz. Higher values denote better performance.

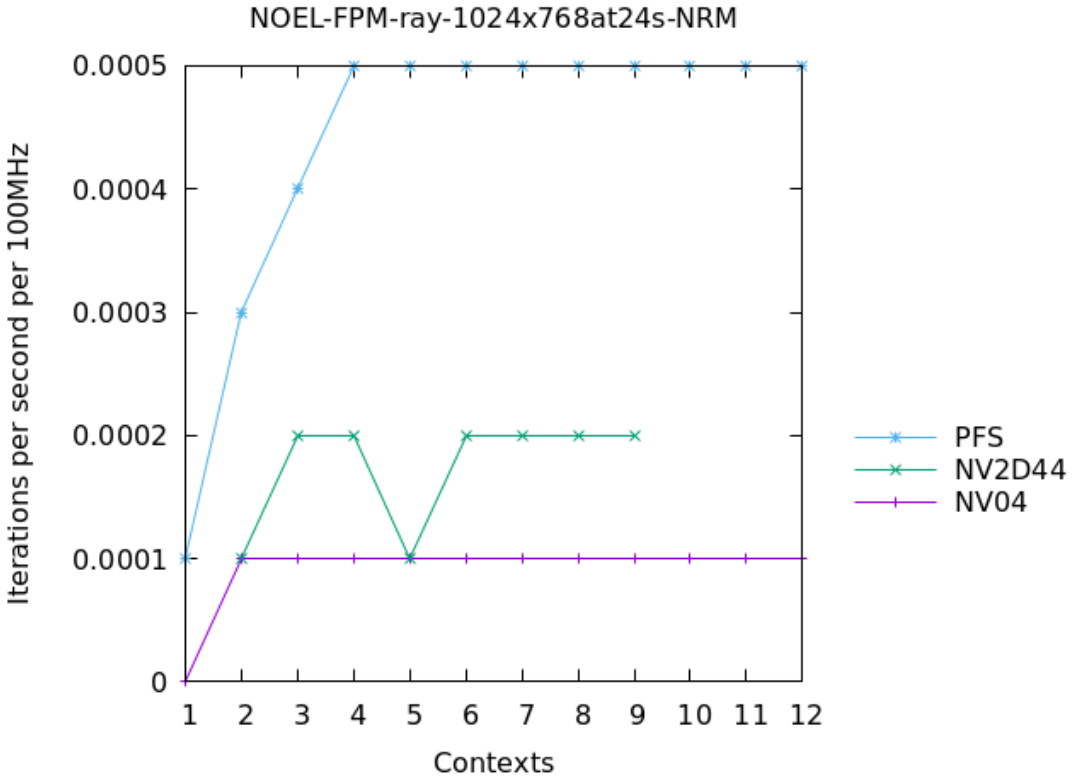

Figure 77: NOEL - FPMark - ray-1024x768at24s, iterations per second at 100MHz. Higher values denote better performance.

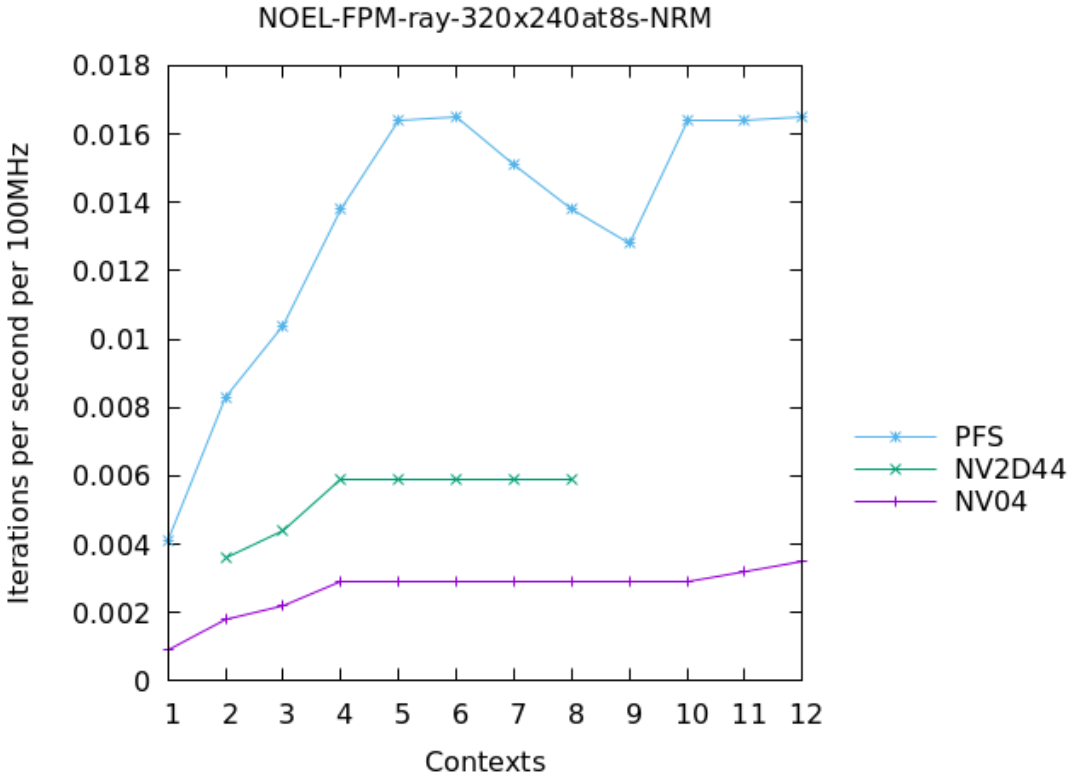

Figure 78: NOEL - FPMark - ray-320x240at8s, iterations per second at 100MHz. Higher values denote better performance.

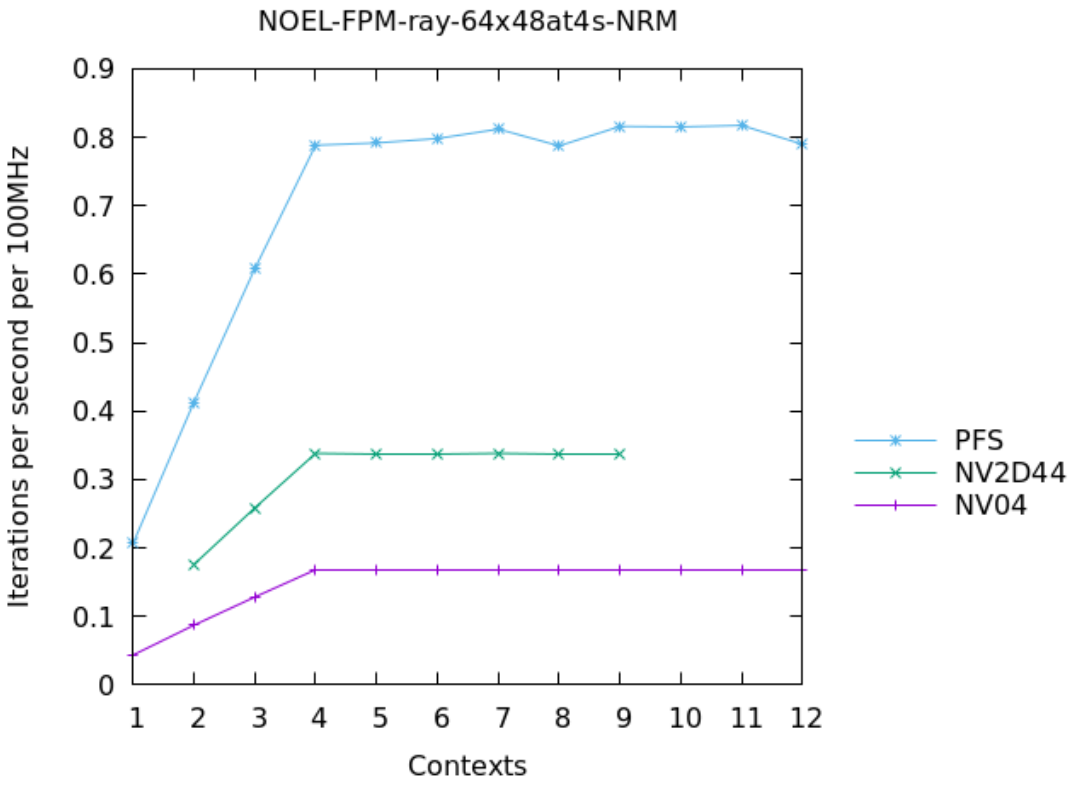

Figure 79: NOEL - FPMark - ray-64x48at4s, iterations per second at 100MHz. Higher values denote better performance.

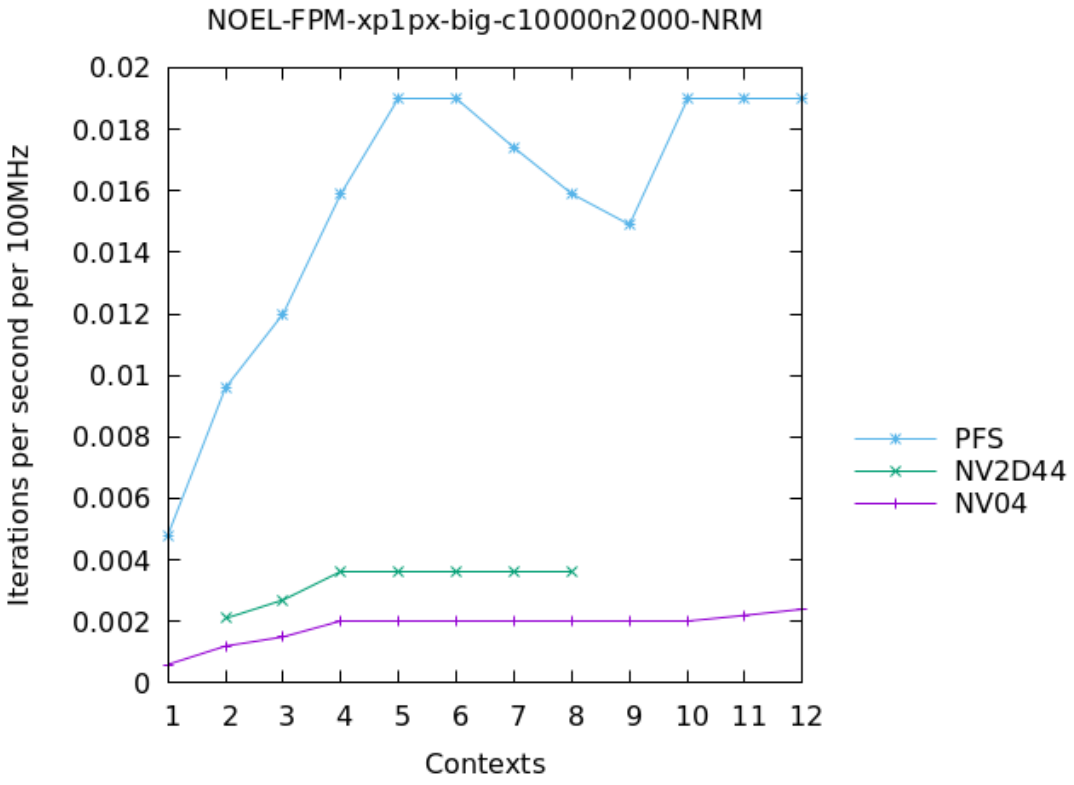

Figure 80: NOEL - FPMark - xp1px-big-c10000n2000, iterations per second at 100MHz. Higher values denote better performance.

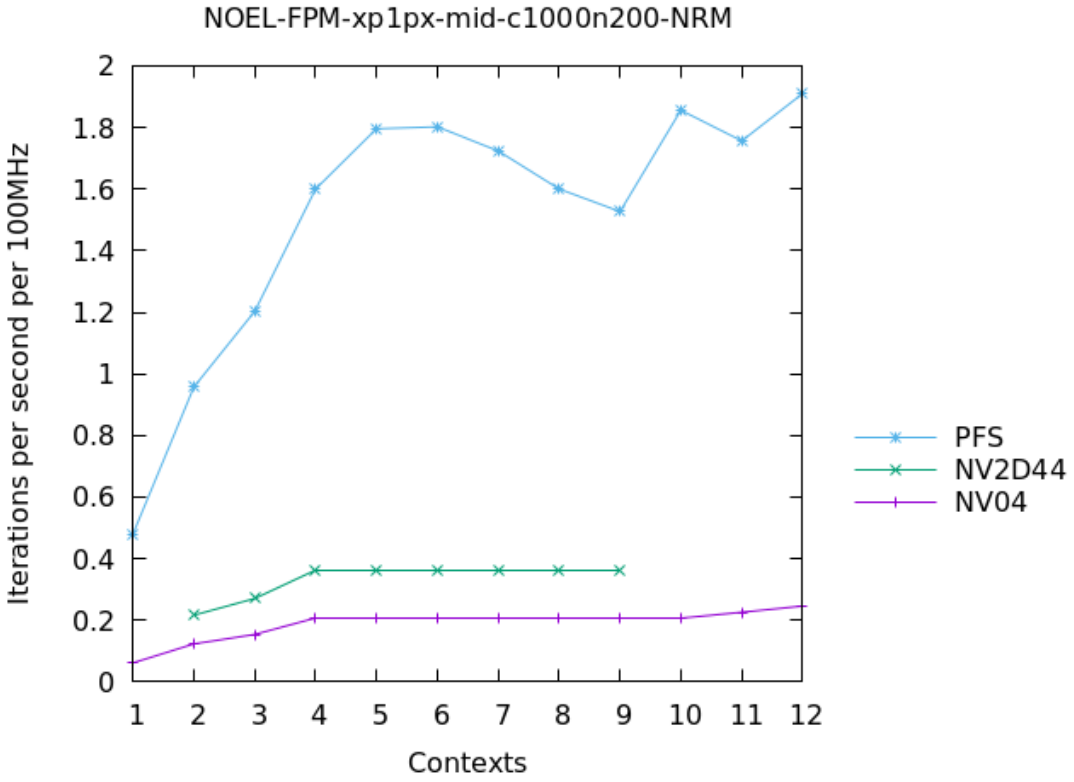

Figure 81: NOEL - FPMark - xp1px-mid-c1000n200, iterations per second at 100MHz. Higher values denote better performance.

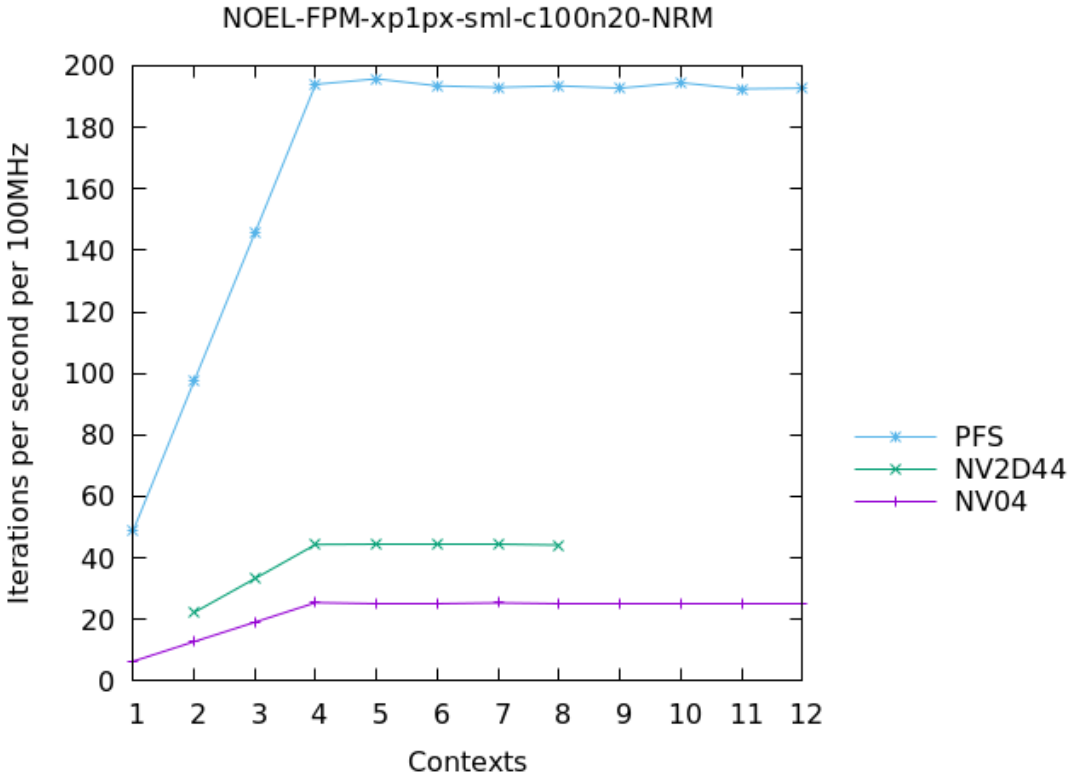

Figure 82: NOEL - FPMark - xp1px-sml-c100n20, iterations per second at 100MHz. Higher values denote better performance.

9 LEON performance

9.1 PWLS benchmarks

9.1.1 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
```

9.1.2 Measurements

For an explanation of the benchmark acronyms see Section 8.1.2.

Table 47 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 47: 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 83 to 85. The figures show the corresponding benchmark

performance at 100MHz. For Figures 83 and 84 higher values denote better processor performance, while for Fig. 85 lower values indicate better performance.

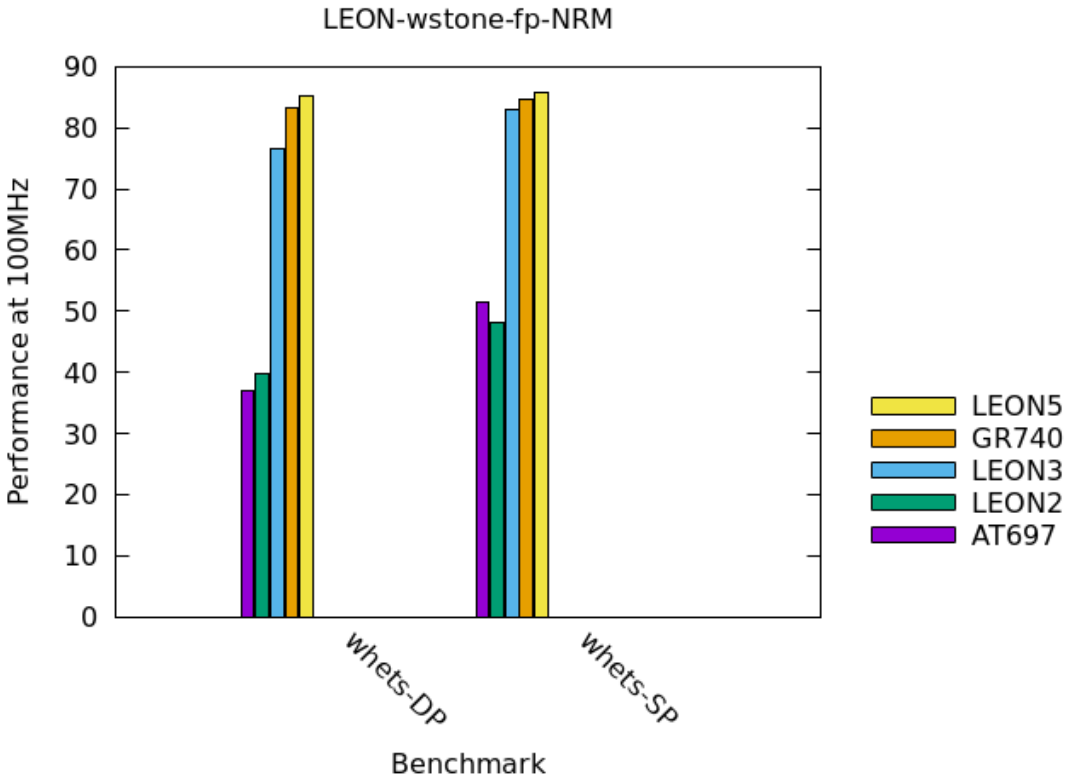

Figure 83: LEON - Whetstone performance for LEON targets, MWIPS at 100MHz. Higher values denote better performance.

9.1.3 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[1] than the previous generation.

[1] both integer and floating-point performance

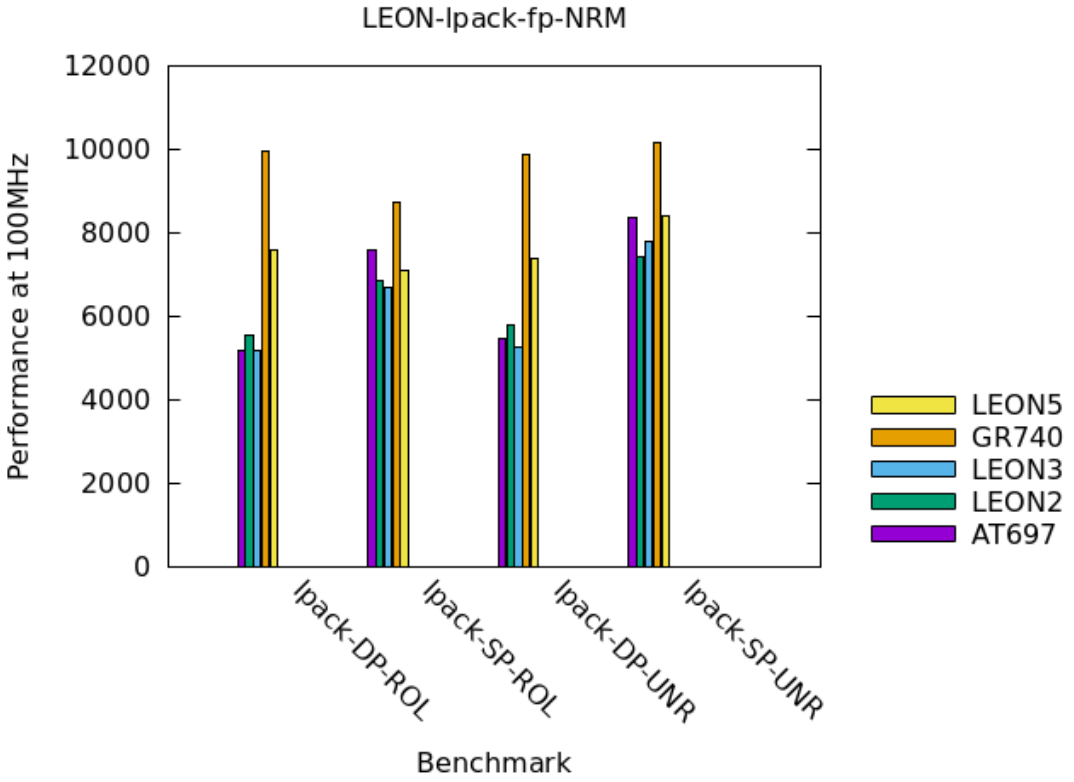

Figure 84: LEON - Linpack performance for LEON targets, Kflops at 100MHz. Higher values denote better performance.

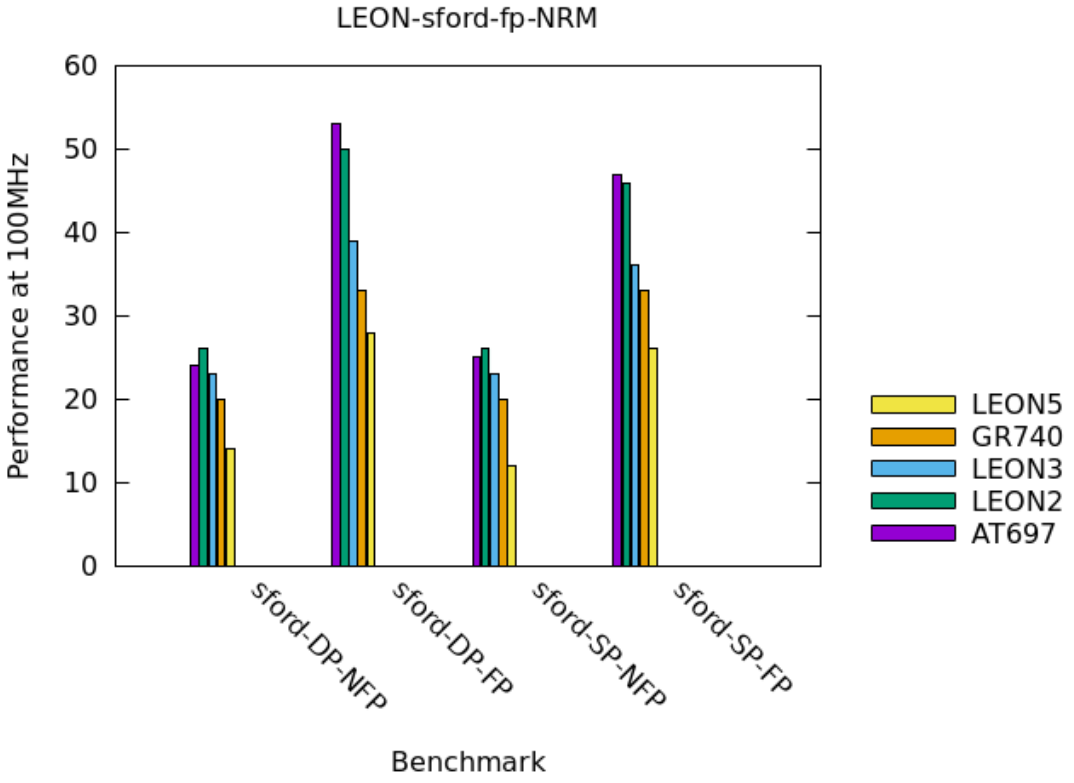

Figure 85: LEON - Stanford performance for LEON targets, milliseconds at 100MHz. Lower values denote better performance.

9.2 CoreMark

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

9.2.1 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
```

9.2.2 Measurements

Table 48 shows results for the *CoreMark* benchmark when `the core_matrix()` function is computed in the floating-point domain. Table 49 shows results for the *CoreMark* benchmark when `the core_matrix()` function is computed in the integer domain.

Table 48: 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 49: 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 86 and 87. The figures show the *CoreMark* iterations per second at 100MHz. Higher values denote better processor performance.

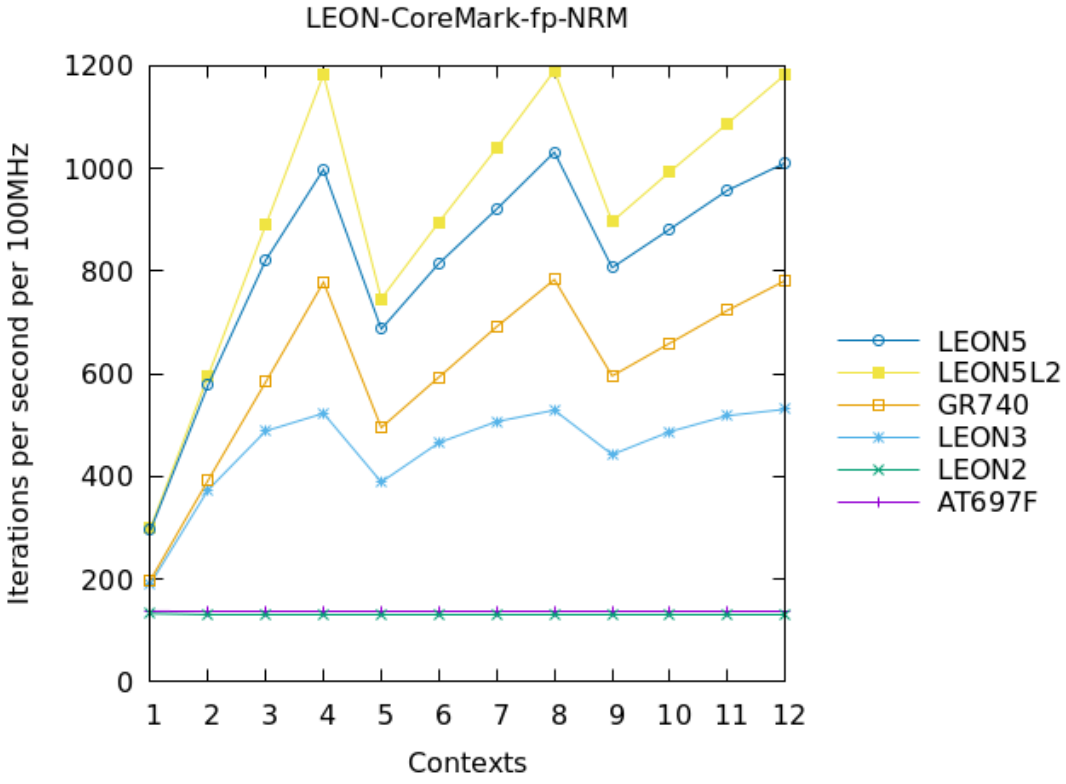

Figure 86: LEON - CoreMark - floating-point performance for LEON targets, CoreMark iterations at 100MHz. Higher values denote better performance.

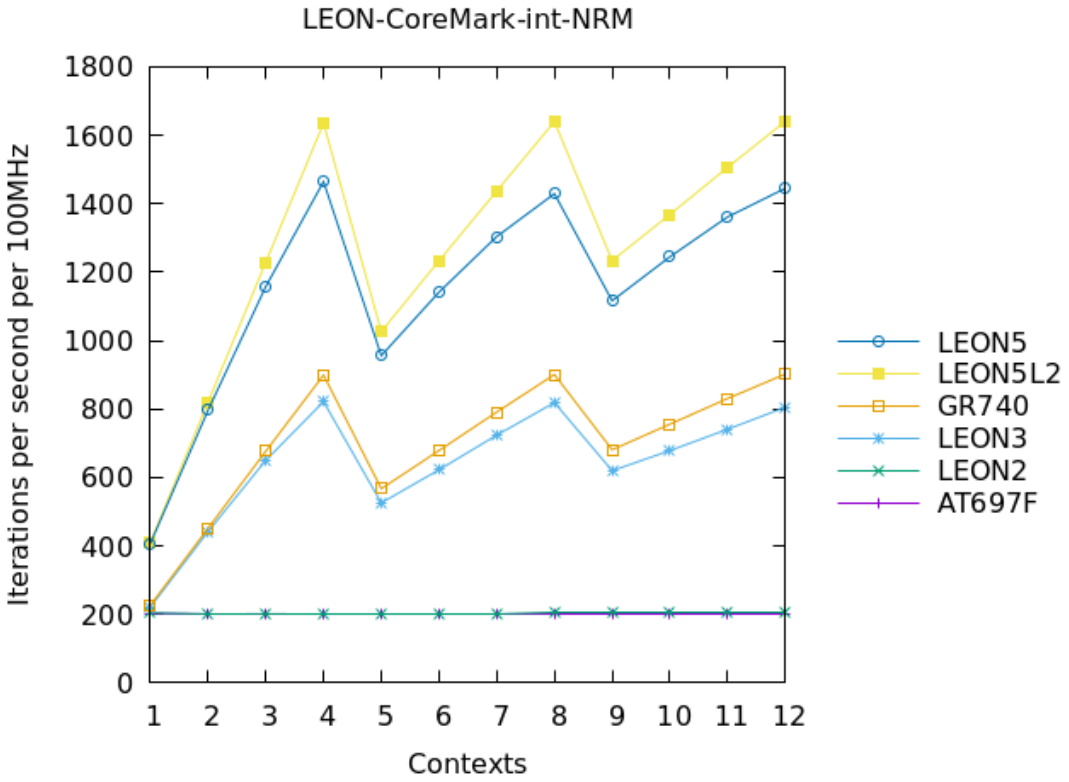

Figure 87: LEON - CoreMark - integer performance for LEON targets, CoreMark iterations at 100MHz. Higher values denote better performance.

9.2.3 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 Sections 8.2.3 and 8.2.5).

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).

9.3 CoreMark-Pro

For a definition of the workloads and specification of the measurements see Section 8.3.1.

9.3.1 Worst-case execution time

An approximate worst-case execution time for the *CoreMark-Pro* suite executed in LEON-based systems is shown in Table 50.

Table 50: LEON - worst-case execution times.

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

9.3.2 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
```

9.3.3 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 51 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 51: 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 52: 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 53: 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 54: LEON - CoreMark-Pro results for configuration LE7 (GR740) - workload iterations per second at 250MHz. *-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 55: 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 56: 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 57: 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 52 to 57 are plotted in Figures 88 to 96.

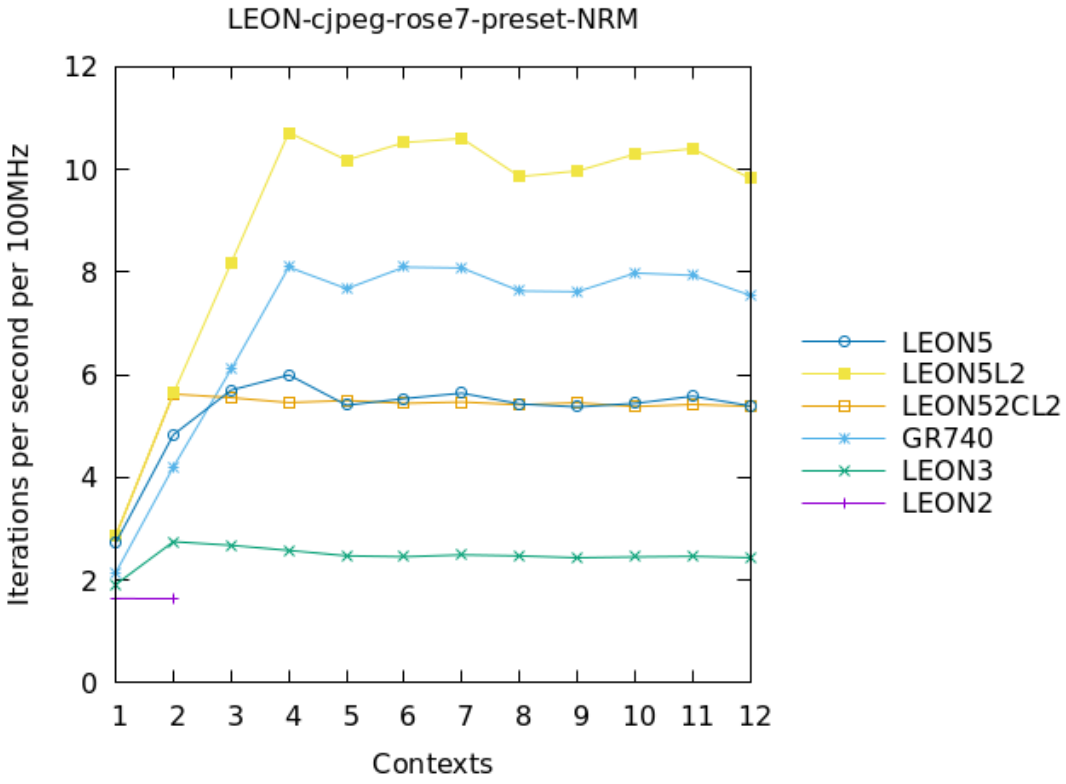

Figure 88: LEON - CoreMark-Pro - cjpeg, iterations per second at 100MHz. Higher values denote better performance.

9.3.4 Analysis

Observations based on Figures 88 to 96 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

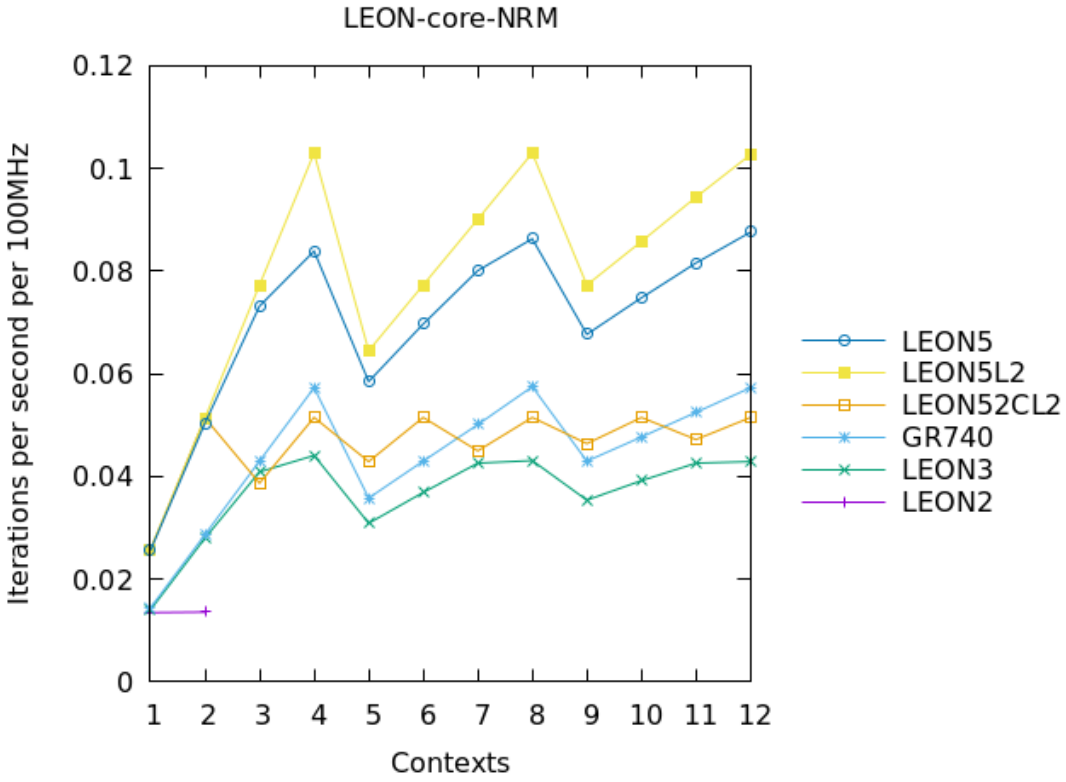

Figure 89: LEON - CoreMark-Pro - core, iterations per second at 100MHz. Higher values denote better performance.

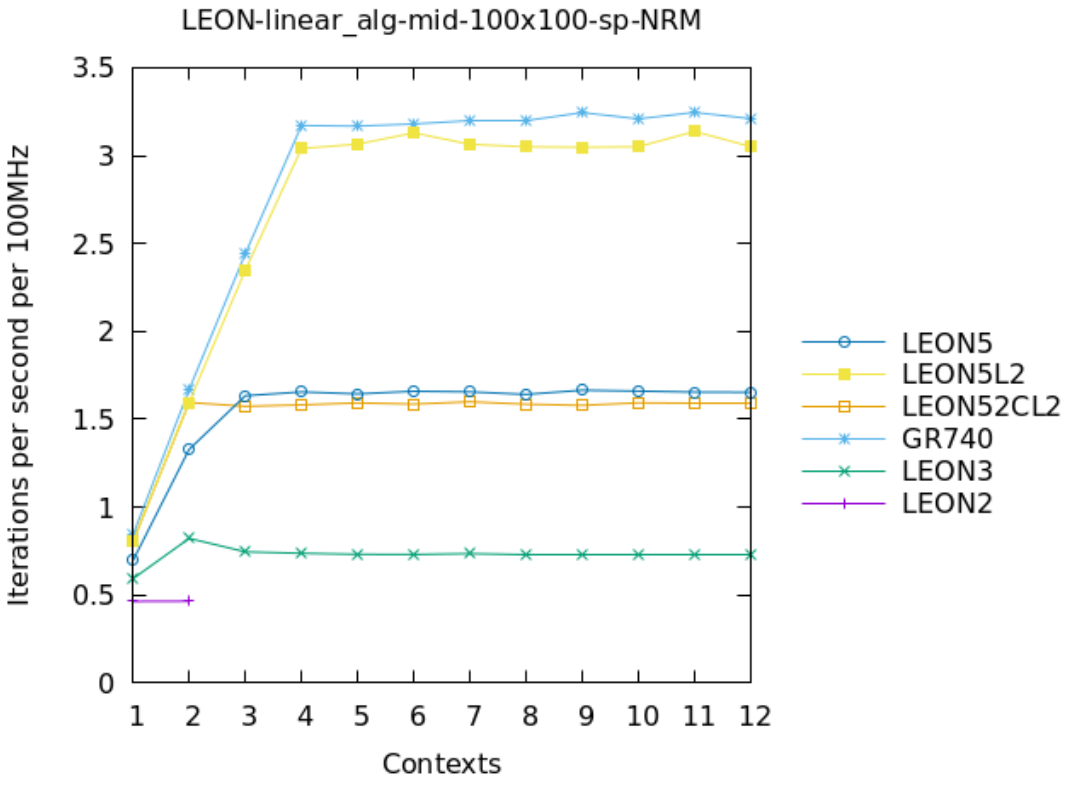

Figure 90: LEON - CoreMark-Pro - linear, iterations per second at 100MHz. Higher values denote better performance.

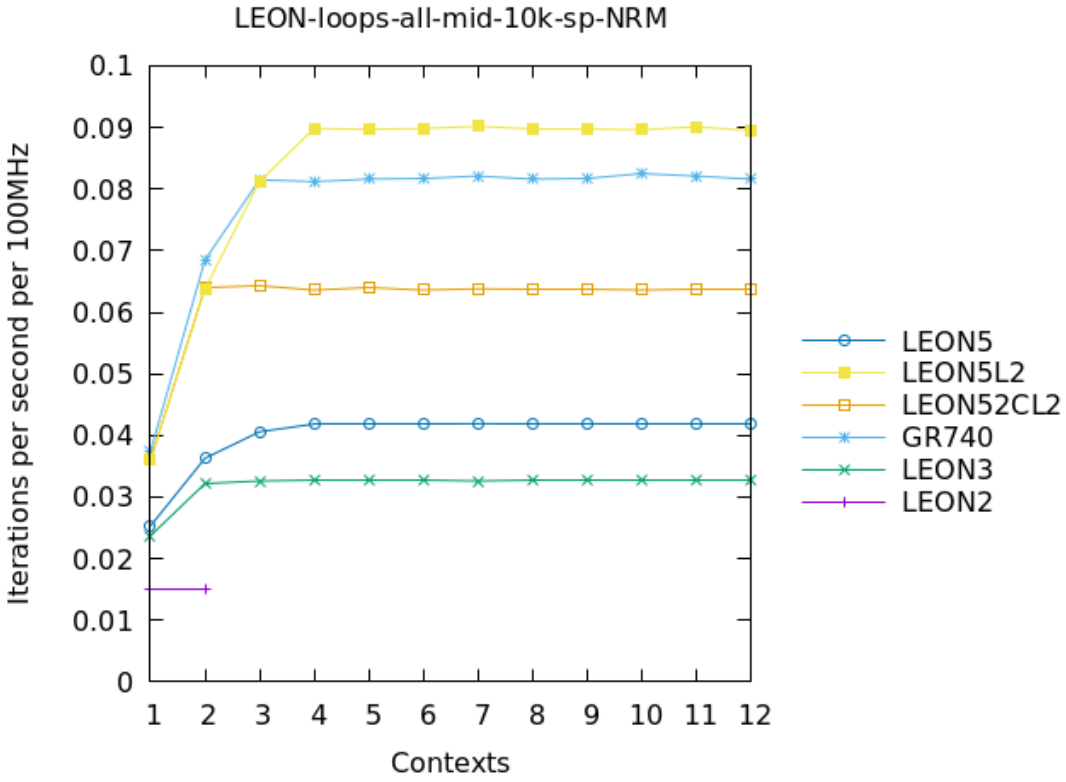

Figure 91: LEON - CoreMark-Pro - loops, iterations per second at 100MHz. Higher values denote better performance.

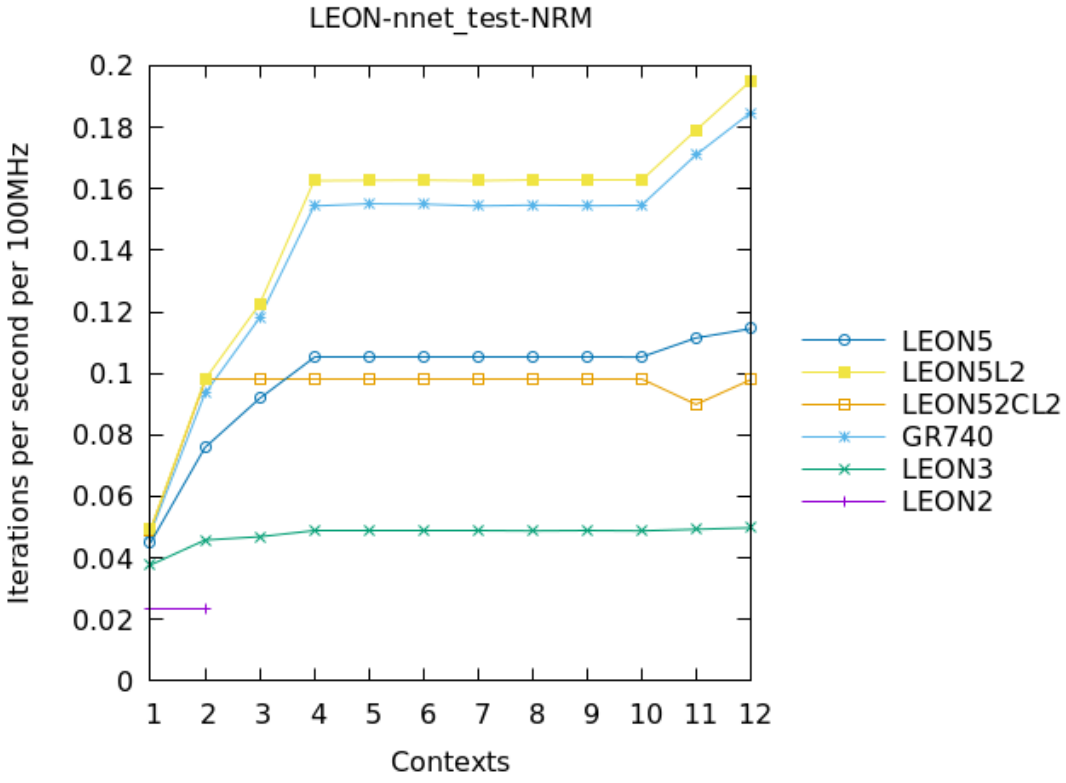

Figure 92: LEON - CoreMark-Pro - nnet, iterations per second at 100MHz. Higher values denote better performance.

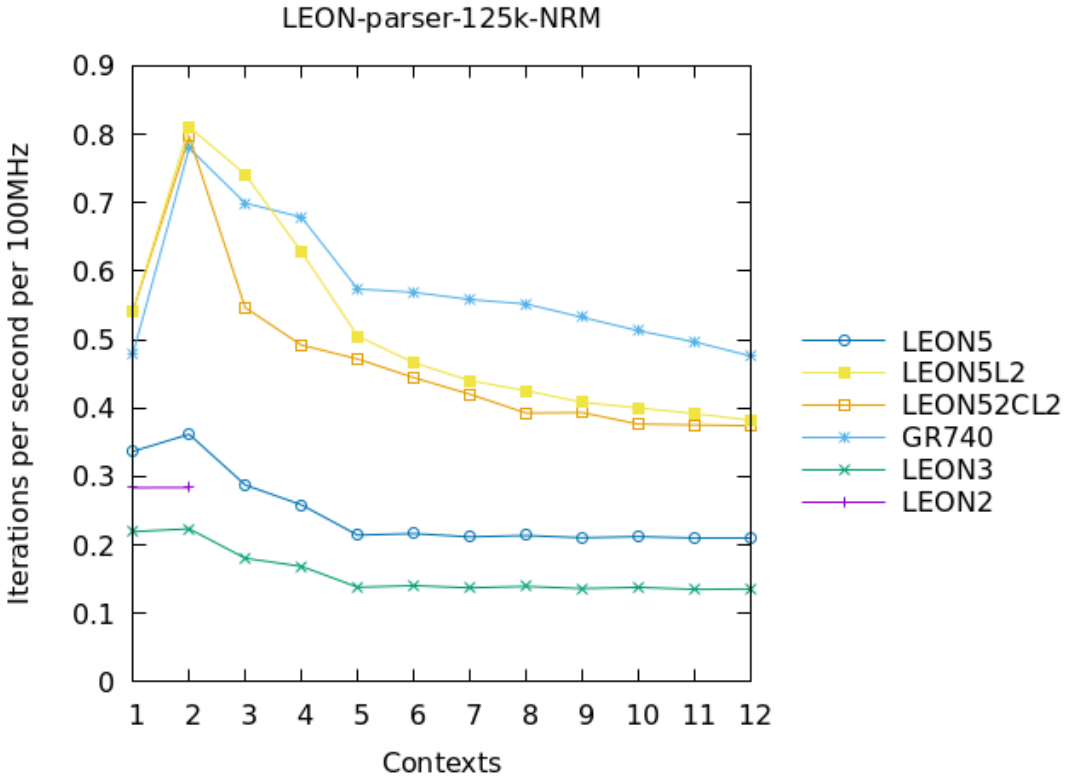

Figure 93: LEON - CoreMark-Pro - parser, iterations per second at 100MHz. Higher values denote better performance.

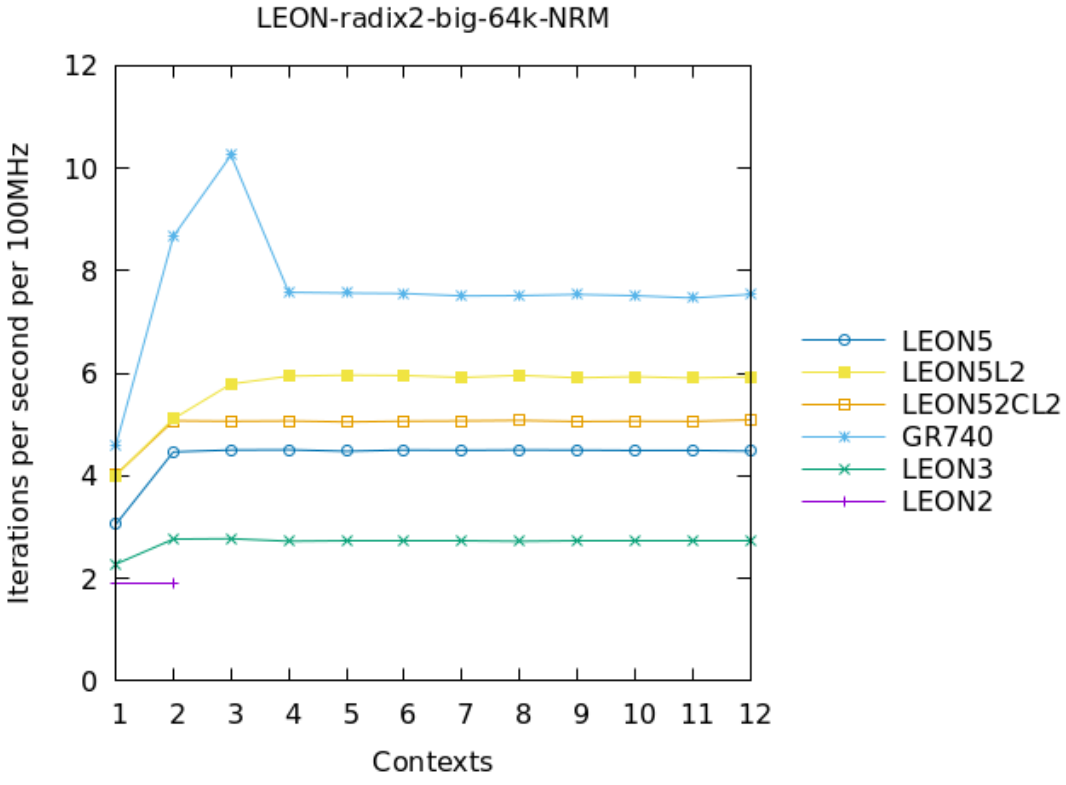

Figure 94: LEON - CoreMark-Pro - radix2, iterations per second at 100MHz. Higher values denote better performance.

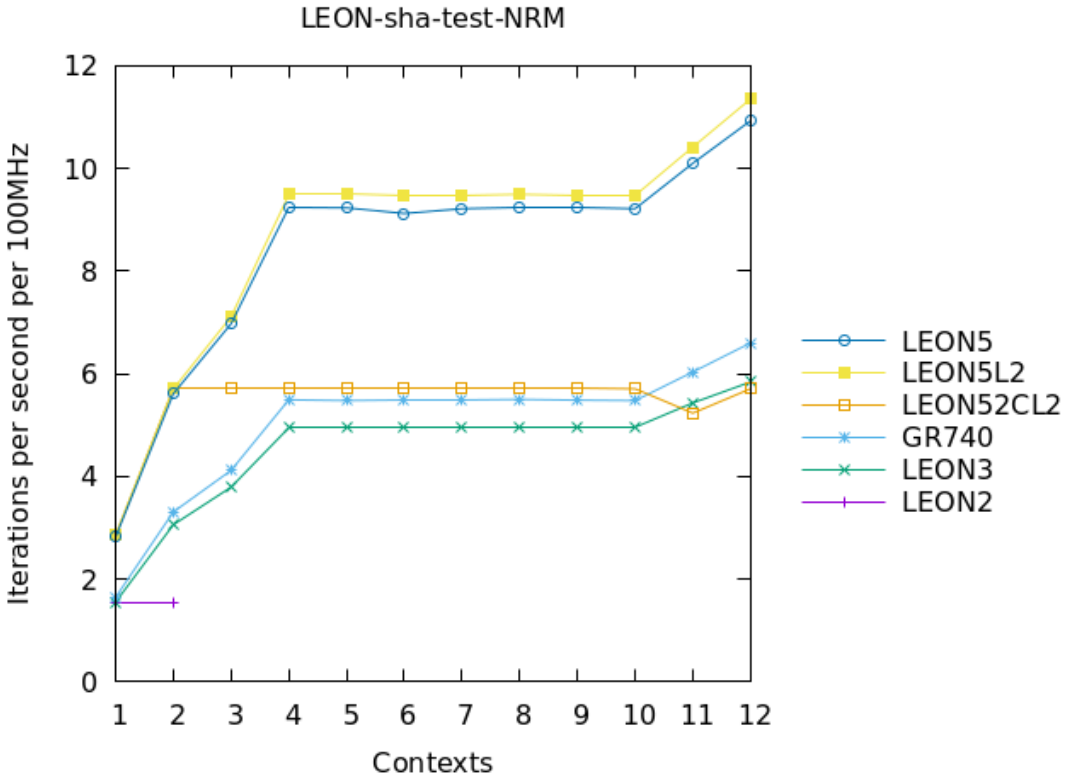

Figure 95: LEON - CoreMark-Pro - sha, iterations per second at 100MHz. Higher values denote better performance.

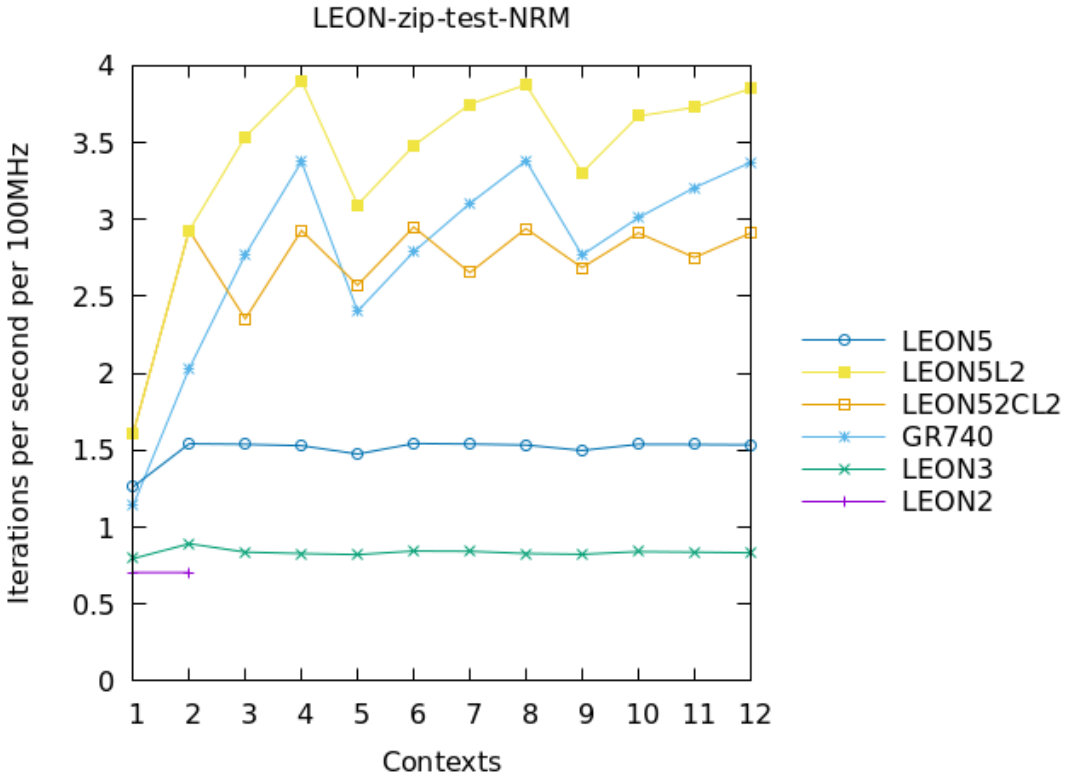

Figure 96: LEON - CoreMark-Pro - zip, iterations per second at 100MHz. Higher values denote better performance.

- achieves the fifth[2] 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*.

9.4 FPMark

For a definition of the workloads and specification of the measurements see Section 8.4.1.

9.4.1 Compiler options

The following options were used for compiling the *FPMark* 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
```

9.4.2 Measurements

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

The *FPMark* data are presented for 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.

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

Table 58: LEON - FPMark results for LEON2 - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second	
Workload	-c1	-c2
atan-1M	0.1741	0.1749
atan-1M-sp	0.2225	0.2239
atan-1k	179.6429	180.6391
atan-1k-sp	227.1591	228.7492
atan-64k	2.8524	2.8661
atan-64k-sp	3.6442	3.6682
blacks-big-n5000v200	0.0050	0.0050
blacks-big-n5000v200-sp	0.0060	0.0060
blacks-mid-n1000v40	0.1256	0.1256
blacks-mid-n1000v40-sp	0.1490	0.1490
blacks-sml-n500v20	0.5020	0.5020
blacks-sml-n500v20-sp	0.5948	0.5948
horner-big-100k	0.8022	0.8022
horner-big-100k-sp	0.9446	0.9446
horner-mid-10k	7.9898	7.9900
horner-mid-10k-sp	9.4036	9.4038
horner-sml-1k	76.8710	76.8971
horner-sml-1k-sp	91.6028	91.6280
inner-product-big-100k	0.1062	0.1062
inner-product-big-100k-sp	0.1172	0.1172
inner-product-mid-10k	1.0603	1.0603
inner-product-mid-10k-sp	1.1703	1.1703
inner-product-sml-1k	34.3808	34.3796
inner-product-sml-1k-sp	43.9194	43.9348
linear_alg-big-1000x1000	0.0018	0.0018
linear_alg-big-1000x1000-sp	0.0021	0.0021
linear_alg-mid-100x100	0.3755	0.3755
linear_alg-mid-100x100-sp	0.4642	0.4649
linear_alg-sml-50x50	3.0537	3.0568
linear_alg-sml-50x50-sp	3.7768	3.7760
loops-all-big-100k	0.0012	0.0012
loops-all-big-100k-sp	0.0014	0.0014
loops-all-mid-10k	0.0129	0.0129
loops-all-mid-10k-sp	0.0151	0.0151
loops-all-tiny	8.8232	8.8250
loops-all-tiny-sp	10.6633	10.6633
lu-big-2000x2_50	0.0158	0.0158
lu-big-2000x2_50-sp	0.0202	0.0202
lu-mid-200x2_50	1.5309	1.5340
lu-mid-200x2_50-sp	1.7634	1.7673
lu-sml-20x2_50	17.0136	17.0435
lu-sml-20x2_50-sp	20.5228	20.5027
nnet-data1-sp	15.3311	15.3320
nnet_data1	11.7447	11.7554

continues on next page

Table 58 – continued from previous page

.	Iterations per second	
Workload	-c1	-c2
nnet_test	0.0232	0.0232
nnet_test-sp	0.0245	0.0245
radix2-big-64k	1.9170	1.9170
radix2-mid-8k	19.3192	19.3195
radix2-sml-2k	123.6756	123.6843
ray-1024x768at24s	0.0001	0.0001
ray-320x240at8s	0.0024	0.0024
ray-64x48at4s	0.1178	0.1177
xp1px-big-c10000n2000	0.0014	0.0015
xp1px-mid-c1000n200	0.1458	0.1472
xp1px-sml-c100n20	15.0360	15.1736

Table 59: LEON - FPMark results for 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
atan-1M	0.4559	0.8998	1.1039	1.4071	1.3982	0.0000
atan-1M-sp	0.5754	1.1163	1.3996	1.7947	1.7879	1.7838
atan-1k	500.5005	986.0960	1453.0660	1866.3680	1060.6703	0.0000
atan-1k-sp	599.2330	1178.9672	1755.3098	2268.0880	1279.0995	0.0000
atan-64k	7.4722	14.7406	21.3904	26.6724	25.9195	25.8645
atan-64k-sp	9.4199	18.3076	26.9658	34.5042	33.4437	33.3812
blacks-big-n5000v200	0.0094	0.0181	0.0217	0.0261	0.0261	0.0261
blacks-big-n5000v200-sp	0.0103	0.0205	0.0254	0.0261	0.0261	0.0261
blacks-mid-n1000v40	0.2339	0.4513	0.5407	0.6506	0.6505	0.6503
blacks-mid-n1000v40-sp	0.2571	0.5121	0.6337	0.6524	0.6522	0.0000
blacks-sml-n500v20	0.9349	1.8031	2.1589	2.5961	2.5934	2.5934
blacks-sml-n500v20-sp	1.0279	2.0458	2.5304	2.6096	2.6062	2.6069
horner-big-100k	1.4270	2.7153	3.9491	4.9003	4.8605	5.0048
horner-big-100k-sp	1.6438	3.2225	4.7297	6.1626	6.0467	6.2445
horner-mid-10k	14.1635	26.5400	38.9302	50.6380	45.8674	48.5343
horner-mid-10k-sp	16.2938	31.3519	46.6832	62.2820	55.8971	58.4932
horner-sml-1k	131.9244	216.3004	301.0869	325.4149	250.2628	266.1344
horner-sml-1k-sp	156.2573	245.7969	370.6312	407.5810	282.2786	0.0000
inner-product-big-100k	0.1938	0.2006	0.2023	0.0000	0.0000	0.0000
inner-product-big-100k-sp	0.3556	0.4019	0.4049	0.4109	0.3921	0.4038
inner-product-mid-10k	1.9330	2.1283	1.9137	1.9416	1.8620	0.0000
inner-product-mid-10k-sp	3.5350	4.2574	3.8097	3.8287	3.6895	3.6817
inner-product-sml-1k	55.3434	105.2964	59.7907	59.3577	42.1763	41.5093
inner-product-sml-1k-sp	81.4664	144.4043	101.1531	90.3098	56.1230	0.0000
linear_alg-big-1000x1000	0.0016	0.0019	0.0019	0.0019	0.0019	0.0000
linear_alg-big-1000x1000-sp	0.0026	0.0033	0.0033	0.0030	0.0031	0.0031
linear_alg-mid-100x100	0.3541	0.4419	0.4280	0.4305	0.4293	0.0000

continues on next page

Table 59 – continued from previous page

.	Iterations per second					
Workload	-c1	-c2	-c3	-c4	-c5	-c6
linear_alg-mid-100x100-sp	0.5935	0.8225	0.7457	0.7357	0.7314	0.7337
linear_alg-sml-50x50	3.5007	4.9363	4.8606	4.8706	4.8473	4.8483
linear_alg-sml-50x50-sp	5.6537	8.9767	8.4671	8.4552	8.3711	0.0000
loops-all-big-100k	0.0015	0.0018	0.0020	0.0000	0.0000	0.0000
loops-all-big-100k-sp	0.0020	0.0025	0.0027	0.0026	0.0026	0.0000
loops-all-mid-10k	0.0181	0.0244	0.0246	0.0247	0.0247	0.0247
loops-all-mid-10k-sp	0.0237	0.0323	0.0325	0.0327	0.0327	0.0327
loops-all-tiny	11.8402	20.3815	26.0268	28.0994	23.6530	0.0000
loops-all-tiny-sp	14.9022	25.9713	36.7836	38.7327	31.3814	32.0842
lu-big-2000x2_50	0.0190	0.0295	0.0294	0.0311	0.0304	0.0305
lu-big-2000x2_50-sp	0.0370	0.0681	0.0795	0.0932	0.0944	0.0947
lu-mid-200x2_50	2.2996	4.2282	5.6183	6.3731	4.1492	4.1442
lu-mid-200x2_50-sp	3.0111	5.7441	7.9313	9.3365	5.8863	0.0000
lu-sml-20x2_50	25.5062	47.0460	62.1373	68.6384	44.2093	44.1517
lu-sml-20x2_50-sp	34.1135	65.0106	88.1725	99.7466	58.6033	0.0000
nnet-data1-sp	26.8356	36.4711	39.8963	40.9299	40.4809	40.4433
nnet_data1	18.9398	22.0907	24.3593	25.0853	24.9582	24.9321
nnet_test	0.0377	0.0458	0.0469	0.0487	0.0488	0.0489
nnet_test-sp	0.0439	0.0612	0.0634	0.0681	0.0678	0.0000
radix2-big-64k	2.2790	2.7673	2.7752	2.7262	2.7219	0.0000
radix2-mid-8k	21.8411	26.5159	26.4128	25.9744	25.7471	25.7781
radix2-sml-2k	154.3277	208.2691	209.1713	206.0258	191.9452	191.7608
ray-1024x768at24s	0.0001	0.0003	0.0004	0.0006	0.0004	0.0004
ray-320x240at8s	0.0043	0.0086	0.0108	0.0143	0.0143	0.0000
ray-64x48at4s	0.2139	0.4259	0.6227	0.8074	0.8063	0.8065
xp1px-big-c10000n2000	0.0026	0.0051	0.0061	0.0068	0.0068	0.0000
xp1px-mid-c1000n200	0.2665	0.5124	0.6143	0.6880	0.6877	0.6877
xp1px-sml-c100n20	27.0563	51.8350	72.4428	77.2499	72.9395	0.0000

Table 60: LEON - FPMark results for configuration GR740 - workload iterations per second at 250MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
atan-1M	1.2835	2.6103	3.0750	4.2212	4.1118	4.2662	4.2463	4.1186	4.2790	4.2900	4.7373	5.1813
atan-1M-sp	1.4952	2.9833	3.7300	4.9505	4.9285	4.9505	4.9383	4.9358	4.9456	4.9480	5.4241	5.9201
atan-1k	1436.7816	2867.7947	4301.0753	5724.0985	5063.2911	4894.7626	5022.6017	4741.5837	4948.0455	4748.3381	4732.6077	4657.6619
atan-1k-sp	1535.8624	3074.0855	4591.3682	6112.4694	6405.4054	6230.1255	5333.3333	5293.8062	5260.3893	5128.2051	5020.0803	4997.5012
atan-64k	23.4412	46.6331	69.5507	83.9842	85.0702	86.0067	86.4454	81.4730	84.0901	82.7472	86.5276	81.6460
atan-64k-sp	25.2787	50.7151	75.8495	100.6847	100.7354	100.2506	100.1101	100.2607	100.0400	100.0600	100.3714	100.0500
blacks-big-n5000v200	0.0264	0.0529	0.0664	0.0883	0.0880	0.0883	0.0880	0.0880	0.0880	0.0883	0.0971	0.1055
blacks-big-n5000v200-sp	0.0263	0.0524	0.0658	0.0876	0.0873	0.0873	0.0873	0.0876	0.0873	0.0873	0.0964	0.1047
blacks-mid-n1000v40	0.6634	1.3210	1.6513	2.1978	2.1978	2.2065	2.2065	2.1978	2.1978	2.1978	2.4160	2.6345
blacks-mid-n1000v40-sp	0.6556	1.3099	1.6439	2.1820	2.1820	2.1820	2.1820	2.1820	2.1820	2.1820	2.3997	2.6172
blacks-sml-n500v20	2.6525	5.2798	6.6007	8.7873	8.7873	8.7873	8.8183	8.7873	8.7873	8.8183	9.6576	10.5263
blacks-sml-n500v20-sp	2.6316	5.2356	6.5445	8.7566	8.7184	8.7566	8.6881	8.7566	8.7184	8.7566	9.5902	10.4530
horner-big-100k	4.0912	7.6441	10.6929	13.7363	14.1004	13.8293	13.7798	13.7250	13.7931	13.8179	13.7798	13.7231
horner-big-100k-sp	4.2952	8.2802	11.6455	15.3421	15.6152	15.4416	15.4107	15.3092	15.6372	15.3563	15.4012	15.3069
horner-mid-10k	42.2744	84.1043	126.8553	168.6625	164.2306	166.1958	166.5556	162.4695	163.8538	165.4260	166.8057	164.6091
horner-mid-10k-sp	42.5170	84.8752	127.0487	169.7217	165.8100	166.6667	167.6165	164.0689	165.9751	166.4170	167.5884	165.9751
horner-sml-1k	398.8831	795.7349	1190.6179	1585.2885	1438.6419	1382.9346	1406.6676	1288.1618	1274.3724	1321.3531	1336.7197	1338.1507
horner-sml-1k-sp	403.3722	806.9071	1212.7092	1603.3349	1450.3263	1415.4282	1424.7044	1369.4878	1329.7872	1347.7089	1356.3000	1365.9336
inner-product-big-100k	0.9208	0.5949	0.5959	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
inner-product-big-100k-sp	1.4533	1.8290	1.4496	1.2570	1.4784	1.5939	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
inner-product-mid-10k	9.1360	10.6818	12.8041	13.9039	14.1623	14.5423	14.2893	14.4645	14.1965	14.4217	14.2816	14.4603

continues on next page

Table 60 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
inner-product-mid-10k-sp	14.3926	19.9611	28.7522	27.4876	28.4698	26.8061	26.6898	27.2424	27.1942	26.7935	26.8330	26.7899
inner-product-sml-1k	210.2607	249.6256	367.3769	211.9992	185.3568	156.1280	156.6416	153.6570	150.9662	149.0091	144.5087	142.8367
inner-product-sml-1k-sp	244.8580	283.3664	623.4414	746.2687	469.9248	460.4052	421.2300	408.8307	358.8088	322.4766	300.0300	288.0184
linear_alg-big-1000x1000	0.0061	0.0072	0.0075	0.0072	0.0074	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075	0.0075
linear_alg-big-1000x1000-sp	0.0086	0.0115	0.0137	0.0131	0.0142	0.0148	0.0145	0.0147	0.0148	0.0146	0.0147	0.0148
linear_alg-mid-100x100	1.9059	3.7425	5.3746	6.8306	6.8521	6.8729	6.8148	6.8975	6.8738	6.9416	6.9156	6.9128
linear_alg-mid-100x100-sp	2.1173	4.1733	6.0909	7.9026	8.1182	7.9227	7.9745	8.0841	8.1433	8.0257	7.9390	7.9962
linear_alg-sml-50x50	13.8769	27.4348	40.8764	53.8445	53.5791	53.6251	53.6251	53.6481	53.6251	53.5504	53.5275	53.7577
linear_alg-sml-50x50-sp	15.0173	29.8418	44.6229	59.5451	59.3401	59.0807	59.0458	59.3190	59.4955	58.9275	59.1017	59.0528
loops-all-big-100k	0.0063	0.0077	0.0081	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
loops-all-big-100k-sp	0.0084	0.0115	0.0135	0.0122	0.0122	0.0134	0.0130	0.0000	0.0000	0.0000	0.0000	0.0000
loops-all-mid-10k	0.0858	0.1176	0.1240	0.1242	0.1242	0.1241	0.1274	0.1231	0.1244	0.1247	0.1269	0.1250
loops-all-mid-10k-sp	0.0936	0.1706	0.2030	0.2013	0.2068	0.2038	0.2059	0.2021	0.2070	0.2066	0.2058	0.2026
loops-all-tiny	38.2205	75.5744	112.5619	149.5215	137.8170	135.3180	137.2119	132.3452	127.3237	131.6829	133.0495	123.6400
loops-all-tiny-sp	39.6385	78.0884	117.3158	156.4945	134.1202	139.2758	142.1666	136.8363	131.2680	135.1717	136.5001	126.0398
lu-big-2000x2_50	0.0969	0.1895	0.2381	0.3071	0.3566	0.3550	0.3278	0.3047	0.2845	0.3544	0.3517	0.3512
lu-big-2000x2_50-sp	0.1166	0.2327	0.2900	0.3852	0.4535	0.4561	0.4152	0.3805	0.3522	0.4506	0.4493	0.4517
lu-mid-200x2_50	7.8225	15.6196	23.2840	30.8366	28.0238	27.8118	27.7093	27.6602	27.3838	27.2509	27.0731	26.9731
lu-mid-200x2_50-sp	8.1840	16.3629	24.3582	32.4486	29.6042	29.5142	29.4066	29.3979	29.1656	28.9151	28.7455	28.6107
lu-sml-20x2_50	88.0390	175.1712	260.0172	343.8671	306.2693	305.1851	303.2049	302.3797	299.3474	297.2563	294.3947	292.4233

continues on next page

Table 60 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
lu-sml-20x2_50-sp	93.1862	185.3671	277.2156	368.8132	325.4149	323.6770	322.1027	320.9861	319.3664	315.9957	312.6661	310.1448
nnet-data1-sp	71.9632	142.0858	209.0738	276.0144	273.8226	274.0477	276.3958	274.6498	274.5744	273.4482	274.1228	273.5230
nnet_data1	60.1251	116.8907	171.7033	224.5173	224.4669	224.4165	224.3158	224.5677	224.5173	223.6636	224.9213	223.2641
nnet_test	0.1197	0.2333	0.2945	0.3863	0.3863	0.3864	0.3867	0.3867	0.3877	0.3868	0.4255	0.4633
nnet_test-sp	0.1171	0.2316	0.2898	0.3824	0.3803	0.3831	0.3815	0.3819	0.3817	0.3831	0.4192	0.4573
radix2-big-64k	11.4587	21.6887	25.5304	19.1395	18.8658	19.0375	18.7568	18.8583	18.8587	18.7709	18.8619	18.7491
radix2-mid-8k	110.0146	190.2913	279.3218	328.2132	323.7923	325.8921	325.4785	327.1181	323.5094	317.0678	319.6522	316.8467
radix2-sml-2k	626.4369	1204.8773	1569.2428	1938.5104	1926.9679	1836.0078	1925.7433	1843.4879	2023.9228	1801.3474	1890.9311	1866.9255
ray-1024x768at24s	0.0004	0.0008	0.0012	0.0016	0.0010	0.0012	0.0014	0.0016	0.0012	0.0013	0.0015	0.0016
ray-320x240at8s	0.0123	0.0247	0.0309	0.0411	0.0411	0.0411	0.0411	0.0410	0.0409	0.0410	0.0452	0.0491
ray-64x48at4s	0.6115	1.2170	1.7894	2.3399	2.3481	2.3384	2.3477	2.3383	2.3481	2.3484	2.3482	2.3475
xp1px-big-c10000n2000	0.0075	0.0150	0.0187	0.0251	0.0250	0.0251	0.0250	0.0250	0.0251	0.0250	0.0275	0.0300
xp1px-mid-c1000n200	0.7630	1.5195	1.8997	2.5284	2.5284	2.5387	2.5284	2.5284	2.5284	2.5284	2.7799	3.0311
xp1px-sml-c100n20	77.3276	154.5834	230.2026	307.0310	302.2061	303.4901	303.7667	303.3981	299.7602	301.5682	303.6745	302.7551

Table 61: LEON - FPMark results for configuration 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
atan-1M	0.6197	1.2577	1.5511	2.0881	2.0555	2.0982	2.1017	2.0496	2.0995	2.0872	2.3168	2.5010
atan-1M-sp	0.7239	1.4474	1.8021	2.4073	2.4096	2.4038	2.4067	2.4062	2.4067	2.4085	2.6430	2.8867
atan-1k	676.5442	1349.5277	2024.2915	2693.9655	2361.2751	2257.3363	2290.9507	2196.3541	2287.2827	2176.7523	2242.6553	2212.3894
atan-1k-sp	740.3569	1477.7597	2215.8210	2948.9826	2574.0026	2482.6216	2497.5025	2450.9804	2443.7928	2480.1587	2466.6996	2465.4832
atan-64k	10.1575	20.4152	30.2435	40.6719	41.5610	40.7232	41.2201	40.0288	41.5749	40.6587	40.5088	40.4253
atan-64k-sp	12.1137	23.6429	35.3482	47.2054	46.9351	46.9792	46.9859	46.8033	46.9153	47.0234	47.0278	46.8318
blacks-big-n5000v200	0.0112	0.0224	0.0280	0.0373	0.0373	0.0373	0.0373	0.0373	0.0373	0.0373	0.0410	0.0447

continues on next page

Table 61 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
blacks-big-n5000v200-sp	0.0111	0.0223	0.0279	0.0371	0.0371	0.0371	0.0371	0.0371	0.0371	0.0371	0.0408	0.0445
blacks-mid-n1000v40	0.2796	0.5591	0.6988	0.9321	0.9318	0.9324	0.9321	0.9322	0.9321	0.9322	1.0249	1.1178
blacks-mid-n1000v40-sp	0.2789	0.5564	0.6965	0.9282	0.9281	0.9282	0.9282	0.9283	0.9189	0.9284	1.0208	1.1135
blacks-sml-n500v20	1.1179	2.2361	2.7949	3.7272	3.7272	3.7258	3.6873	3.7258	3.6873	3.7244	4.0968	4.4676
blacks-sml-n500v20-sp	1.1148	2.2277	2.7871	3.7106	3.7092	3.7106	3.7092	3.7092	3.7092	3.7092	4.0771	4.4494
horner-big-100k	1.8772	3.7038	5.3014	6.8852	7.0492	6.9070	6.9171	6.8050	7.0437	6.9023	6.9104	6.8710
horner-big-100k-sp	1.8975	3.7998	5.5748	7.4405	7.5126	7.4499	7.4918	7.4189	7.4294	7.4946	7.4923	7.4189
horner-mid-10k	19.2779	38.0373	54.1126	66.9478	68.2175	70.4523	68.8042	65.1126	67.8887	70.1557	68.9180	65.0153
horner-mid-10k-sp	19.2079	38.4956	57.5971	76.6225	74.8111	75.3807	75.6086	75.0525	74.5657	74.9963	75.3580	74.8559
horner-sml-1k	182.6017	364.1926	546.0900	726.3219	646.6214	629.2474	646.4960	539.9860	575.2416	610.0165	617.7034	582.9884
horner-sml-1k-sp	183.4492	366.1931	548.3659	728.1200	647.4587	653.5948	645.2445	607.0908	582.9204	615.5740	624.0639	618.8119
inner-product-big-100k	0.2893	0.3633	0.3618	0.4012	0.4176	0.4135	0.4157	0.4146	0.4198	0.4154	0.4173	0.4221
inner-product-big-100k-sp	0.4952	0.6839	0.7118	0.7860	0.8007	0.8253	0.8227	0.8234	0.8184	0.8118	0.8224	0.8224
inner-product-mid-10k	4.1810	5.3868	6.9471	5.7410	6.2201	6.4224	6.1471	6.3991	6.3431	6.2483	6.1763	6.2547
inner-product-mid-10k-sp	6.6686	9.1422	14.6681	16.0533	15.4452	15.0750	14.3554	14.5932	14.4373	14.4378	14.5836	14.5927
inner-product-sml-1k	57.2115	57.7201	75.8956	92.3020	86.3782	87.5503	85.4336	83.6960	82.1625	82.0883	82.8638	82.4606
inner-product-sml-1k-sp	125.6124	131.4406	151.3317	151.5611	148.2140	144.6550	139.8015	131.9435	133.9944	129.8533	132.9434	129.5840
linear_alg-big-1000x1000	0.0025	0.0040	0.0052	0.0045	0.0053	0.0056	0.0055	0.0055	0.0057	0.0055	0.0055	0.0056
linear_alg-big-1000x1000-sp	0.0032	0.0052	0.0076	0.0071	0.0086	0.0098	0.0093	0.0096	0.0099	0.0095	0.0097	0.0098

continues on next page

Table 61 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
linear_alg-mid-100x100	0.7146	1.4001	2.0286	2.6100	2.6137	2.6129	2.6898	2.6898	2.6201	2.6254	2.5913	2.6214
linear_alg-mid-100x100-sp	0.8059	1.5980	2.3429	3.0464	3.0419	3.1415	3.0531	3.0423	3.0527	3.0421	3.0647	3.0410
linear_alg-sml-50x50	5.4796	10.8479	16.2301	21.4860	21.3602	21.4326	21.3547	21.4105	21.4087	21.3913	21.3831	21.3648
linear_alg-sml-50x50-sp	6.0242	12.0366	18.0109	24.0292	23.8834	23.8846	23.8641	23.8777	23.9040	23.6328	23.9063	23.8880
loops-all-big-100k	0.0026	0.0033	0.0035	0.0032	0.0032	0.0035	0.0035	0.0034	0.0033	0.0035	0.0034	0.0034
loops-all-big-100k-sp	0.0034	0.0043	0.0043	0.0040	0.0040	0.0043	0.0041	0.0041	0.0041	0.0043	0.0042	0.0042
loops-all-mid-10k	0.0298	0.0485	0.0574	0.0593	0.0597	0.0594	0.0602	0.0595	0.0595	0.0598	0.0601	0.0597
loops-all-mid-10k-sp	0.0360	0.0637	0.0812	0.0898	0.0894	0.0896	0.0904	0.0890	0.0888	0.0898	0.0894	0.0897
loops-all-tiny	16.7639	33.2469	49.3925	65.8328	60.2482	61.3497	60.6428	58.2683	56.2177	57.4449	57.9845	55.9722
loops-all-tiny-sp	18.3009	36.4299	54.3301	72.2335	65.7376	64.4496	65.2827	61.1845	60.2627	61.9886	63.1154	59.3895
lu-big-2000x2_50	0.0461	0.0885	0.1099	0.1404	0.1596	0.1595	0.1474	0.1356	0.1228	0.1529	0.1481	0.1471
lu-big-2000x2_50-sp	0.0539	0.1067	0.1331	0.1764	0.2064	0.2076	0.1903	0.1741	0.1613	0.2060	0.2052	0.2043
lu-mid-200x2_50	3.8803	7.7284	11.5215	15.2954	13.4958	13.3760	13.2235	13.1666	13.0943	13.0789	13.0581	12.9057
lu-mid-200x2_50-sp	3.9406	7.8687	11.7547	15.6546	14.0768	13.9667	13.8334	13.7518	13.5956	13.7146	13.6930	13.6876
lu-sml-20x2_50	43.6721	86.3275	129.0706	170.4216	147.9027	146.0366	143.6472	142.7022	142.1343	141.6611	139.7858	140.9980
lu-sml-20x2_50-sp	45.0138	89.6917	133.8921	177.3553	155.0171	153.2802	151.1693	149.8487	149.4478	149.0891	148.8073	148.6215
nnet-data1-sp	30.0824	59.5557	89.1583	118.1893	117.1646	117.7440	117.8412	117.0960	116.8088	117.7579	117.5779	116.5501
nnet_data1	24.8102	48.9452	72.4900	95.5019	95.3925	95.0661	95.2290	94.2507	95.3743	95.3925	95.2200	95.0661
nnet_test	0.0492	0.0979	0.1224	0.1626	0.1626	0.1627	0.1625	0.1627	0.1625	0.1626	0.1787	0.1946
nnet_test-sp	0.0486	0.0969	0.1207	0.1594	0.1593	0.1594	0.1594	0.1592	0.1590	0.1591	0.1750	0.1892
radix2-big-64k	4.0106	5.1148	5.9385	5.9608	5.9461	5.9264	5.9288	5.9493	5.9634	5.9327	5.9071	5.9561
radix2-mid-8k	43.4511	79.4439	108.6874	132.5469	129.8718	133.7542	134.3183	128.2298	129.3193	129.8179	133.8527	127.9607
radix2-sml-2k	262.1184	483.9615	683.1161	900.2926	819.3699	826.4190	857.9272	842.9287	814.9096	804.1106	839.3205	824.3411
ray-1024x768at24s	0.0002	0.0004	0.0006	0.0008	0.0005	0.0006	0.0007	0.0008	0.0006	0.0006	0.0007	0.0008
ray-320x240at8s	0.0060	0.0120	0.0150	0.0200	0.0200	0.0200	0.0198	0.0200	0.0200	0.0200	0.0220	0.0237

continues on next page

Table 61 – continued from previous page

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
ray-64x48at4s	0.2971	0.5937	0.8729	1.1408	1.1384	1.1399	1.1405	1.1399	1.1392	1.1395	1.1404	1.1388
xp1px-big-c10000n2000	0.0031	0.0063	0.0079	0.0105	0.0105	0.0105	0.0105	0.0104	0.0105	0.0105	0.0114	0.0125
xp1px-mid-c1000n200	0.3180	0.6363	0.7955	1.0604	1.0602	1.0603	1.0602	1.0602	1.0600	1.0495	1.1662	1.2709
xp1px-sml-c100n20	32.3562	64.6705	96.7867	129.1322	127.2265	127.4048	128.0082	127.3723	126.5022	127.0487	127.7139	127.0971

The values shown in Tables 58 to 61 are plotted in Figures 97 to 149.

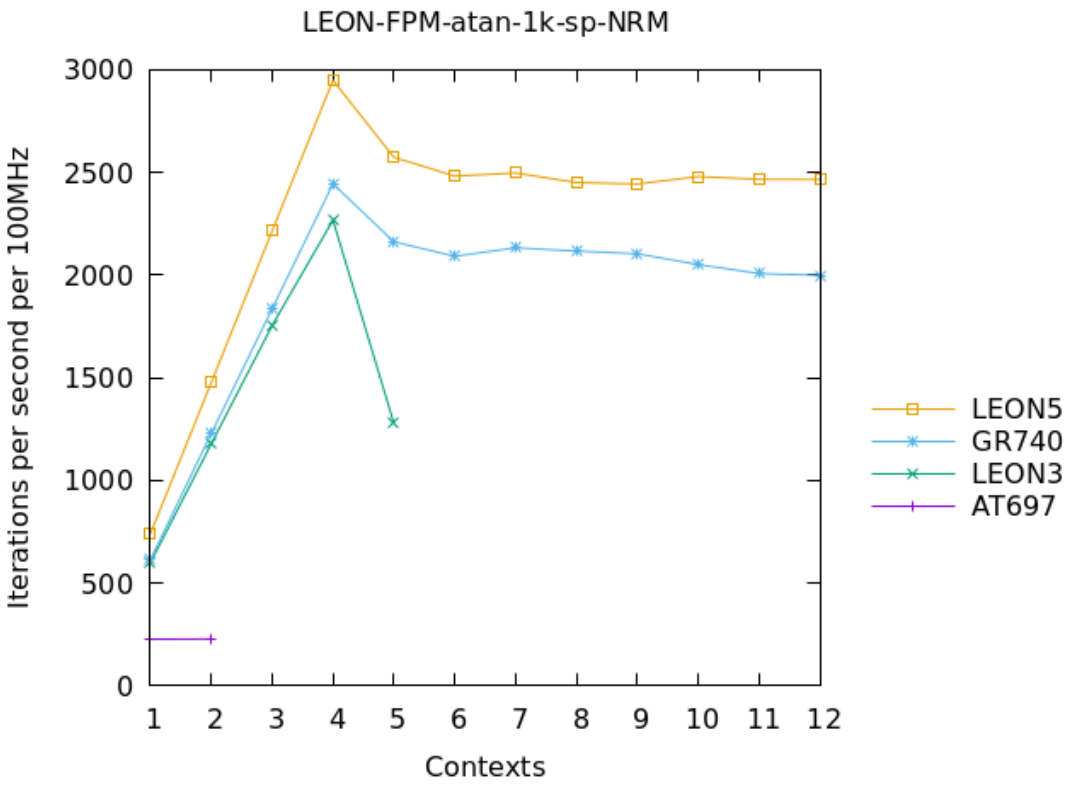

Figure 97: LEON - FPMark - atan-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

9.4.3 Analysis

The workloads can be grouped to categories according to the observed monotonicity of the performance curves:

- Memory system fully saturated for higher number of contexts, examples are *atan-1k, horner-sml, inner-product, loops-all-tiny, lu, radix2*.
- For other workloads the performance curves are monotonous and flat for higher number of contexts.
- For some workloads an additional speedup can be observed for very high number of contexts, examples are *atan-1M, blacks, nnet_test, ray-320x240at8s, xp1px-big, xp1px-med*.
- In general, the configurations that use L2Cache (GR740, LEON5) achieve higher performance.

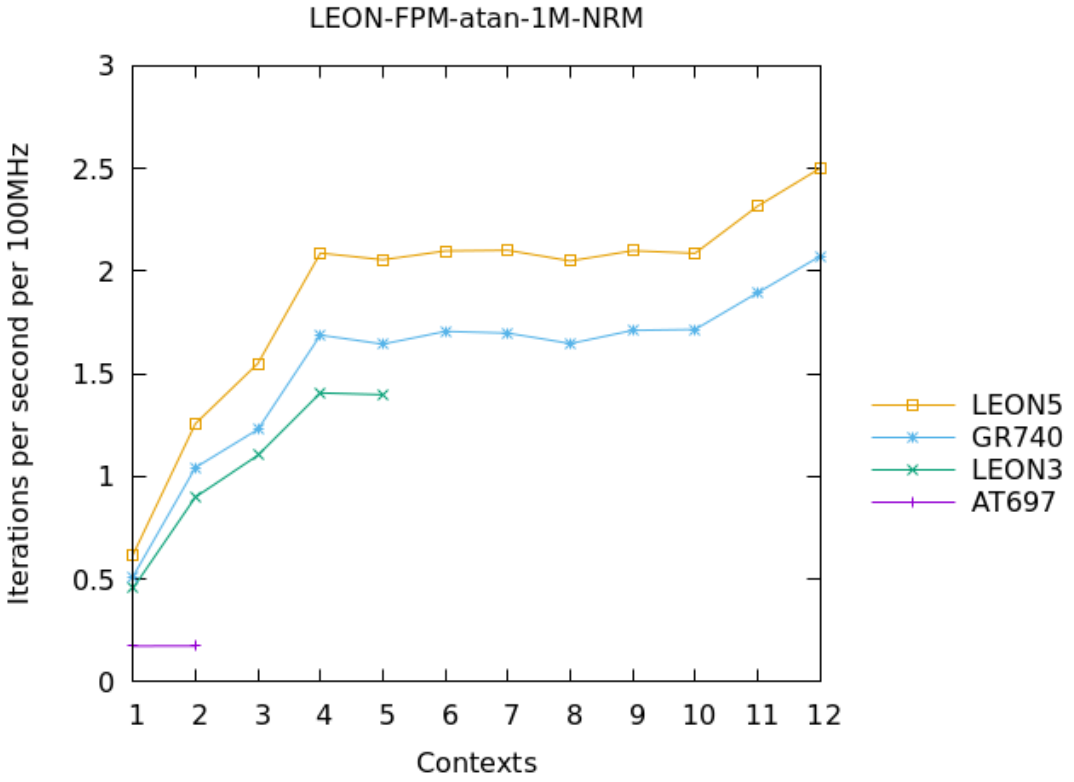

Figure 98: LEON - FPMark - atan-1M, iterations per second at 100MHz. Higher values denote better performance.

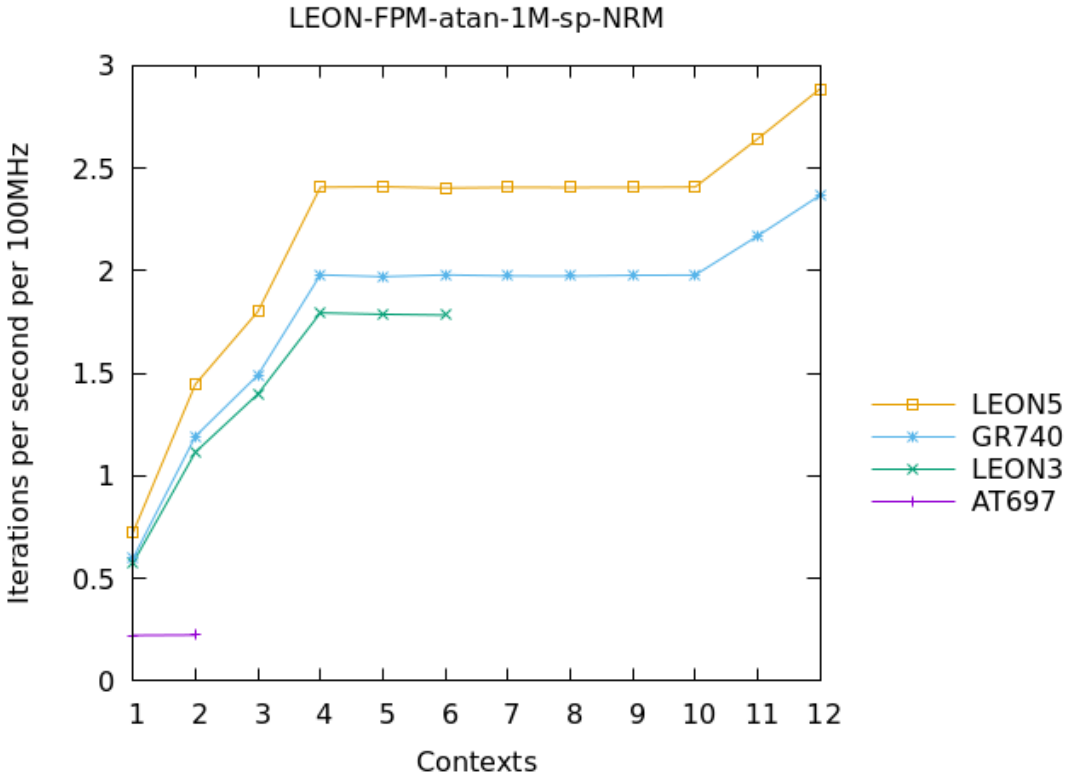

Figure 99: LEON - FPMark - atan-1M-sp, iterations per second at 100MHz. Higher values denote better performance.

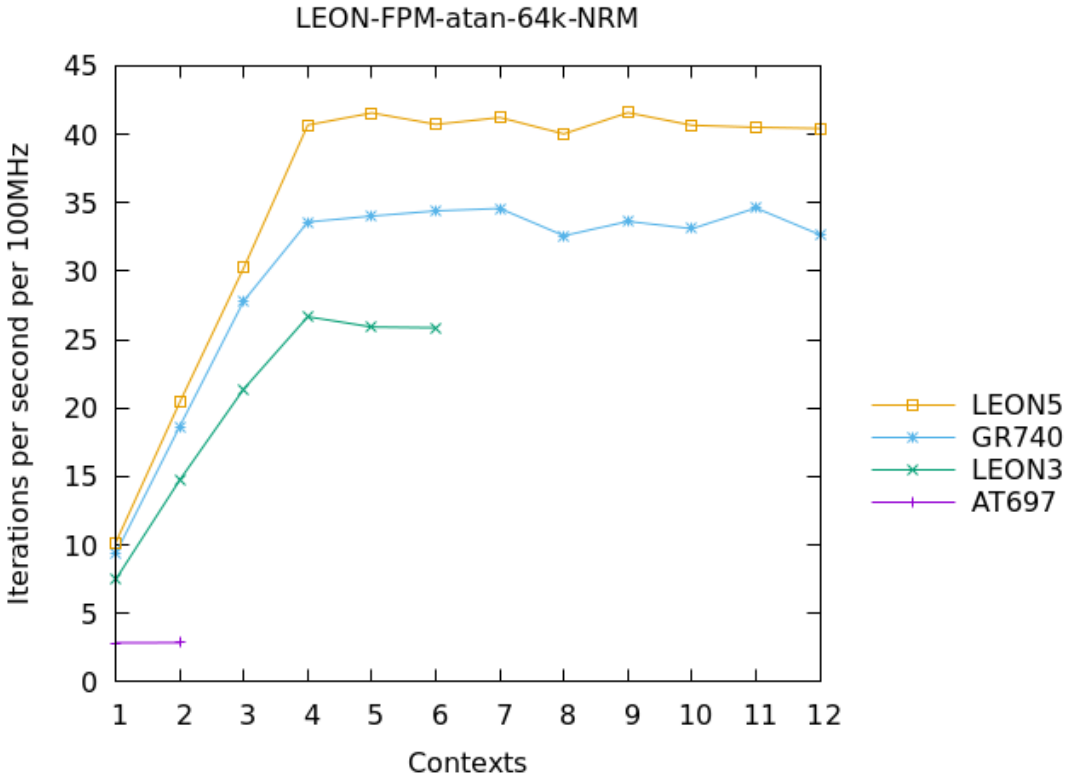

Figure 100: LEON - FPMark - atan-64k, iterations per second at 100MHz. Higher values denote better performance.

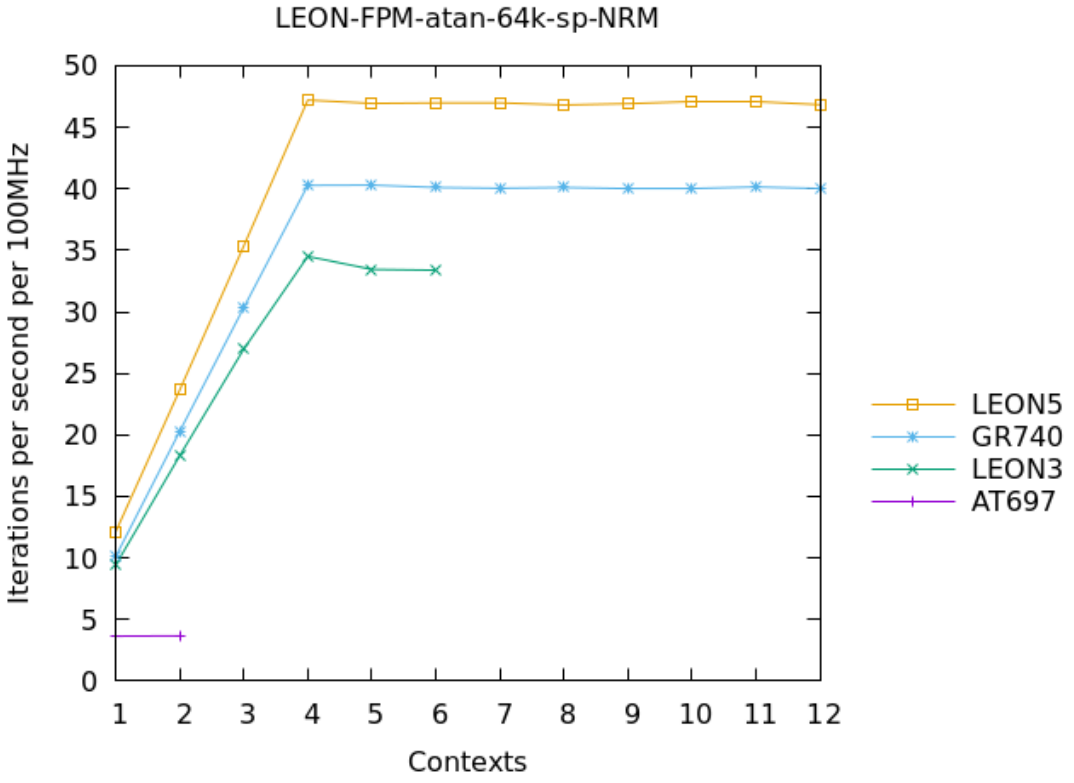

Figure 101: LEON - FPMark - atan-64k-sp, iterations per second at 100MHz. Higher values denote better performance.

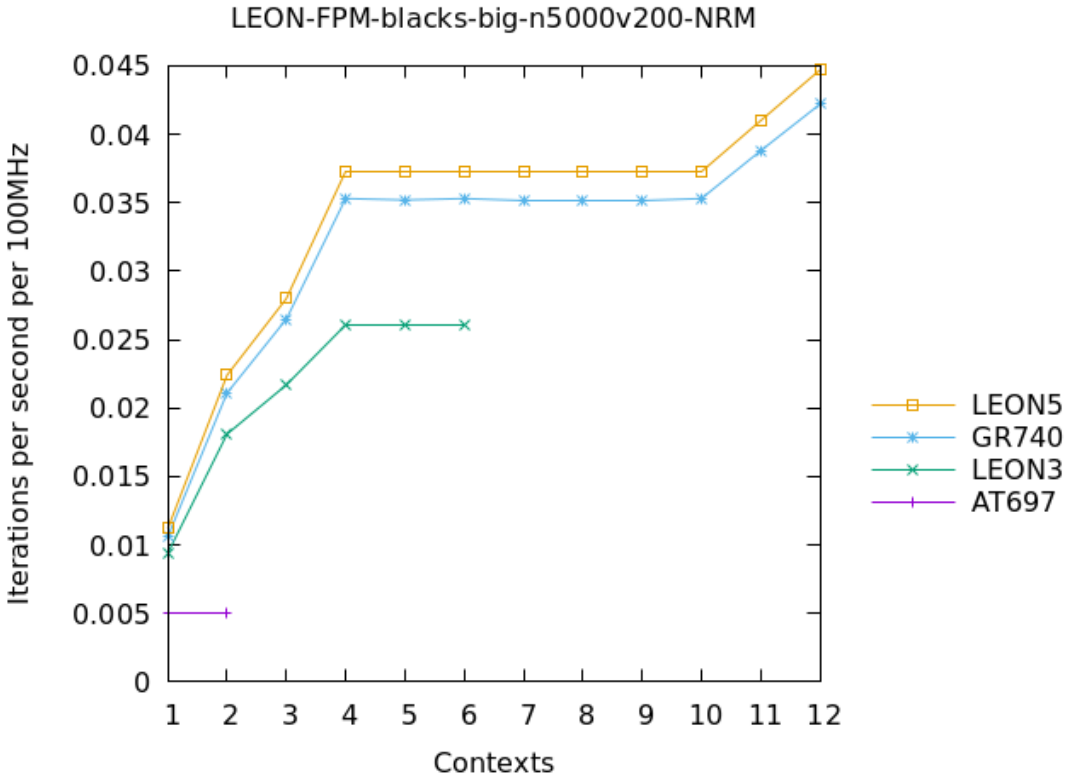

Figure 102: LEON - FPMark - blacks-big-n5000v200, iterations per second at 100MHz. Higher values denote better performance.

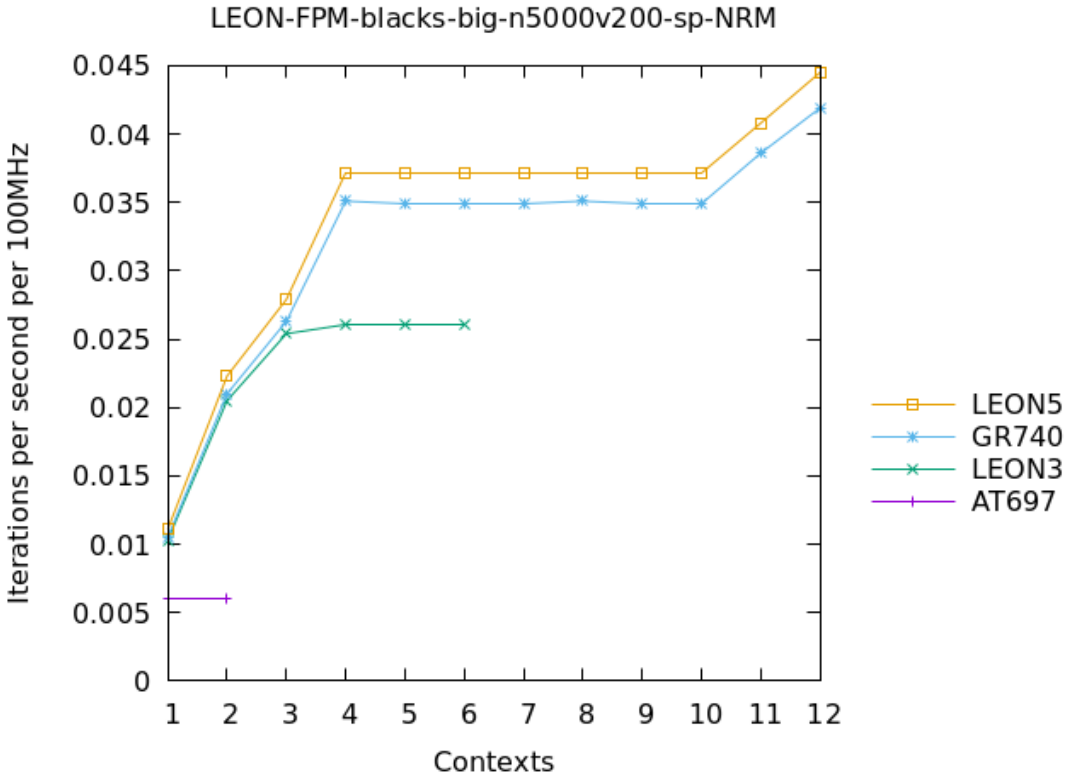

Figure 103: LEON - FPMark - blacks-big-n5000v200-sp, iterations per second at 100MHz. Higher values denote better performance.

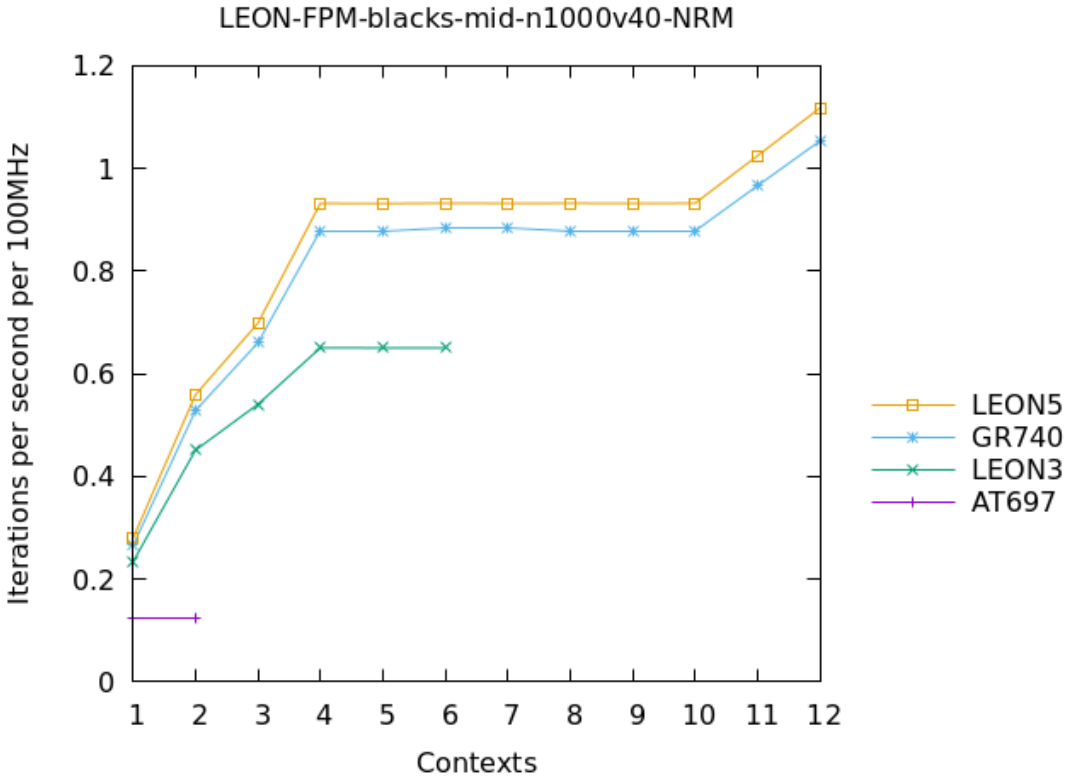

Figure 104: LEON - FPMark - blacks-mid-n1000v40, iterations per second at 100MHz. Higher values denote better performance.

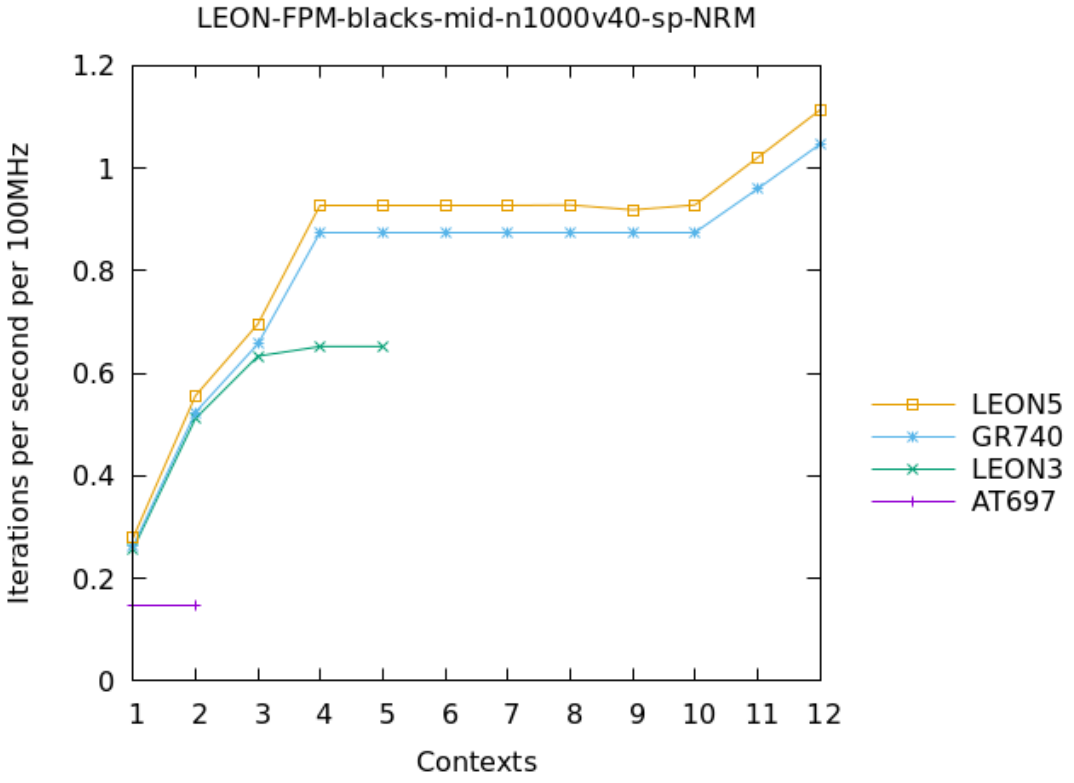

Figure 105: LEON - FPMark - blacks-mid-n1000v40-sp, iterations per second at 100MHz. Higher values denote better performance.

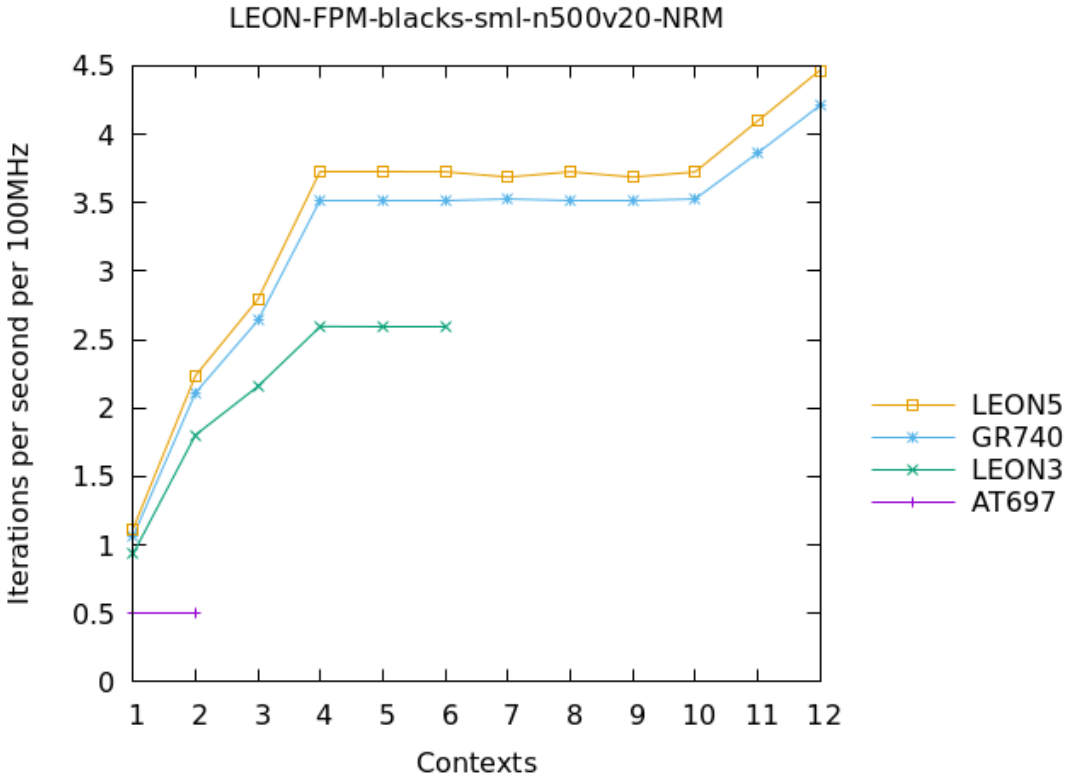

Figure 106: LEON - FPMark - blacks-sml-n500v20, iterations per second at 100MHz. Higher values denote better performance.

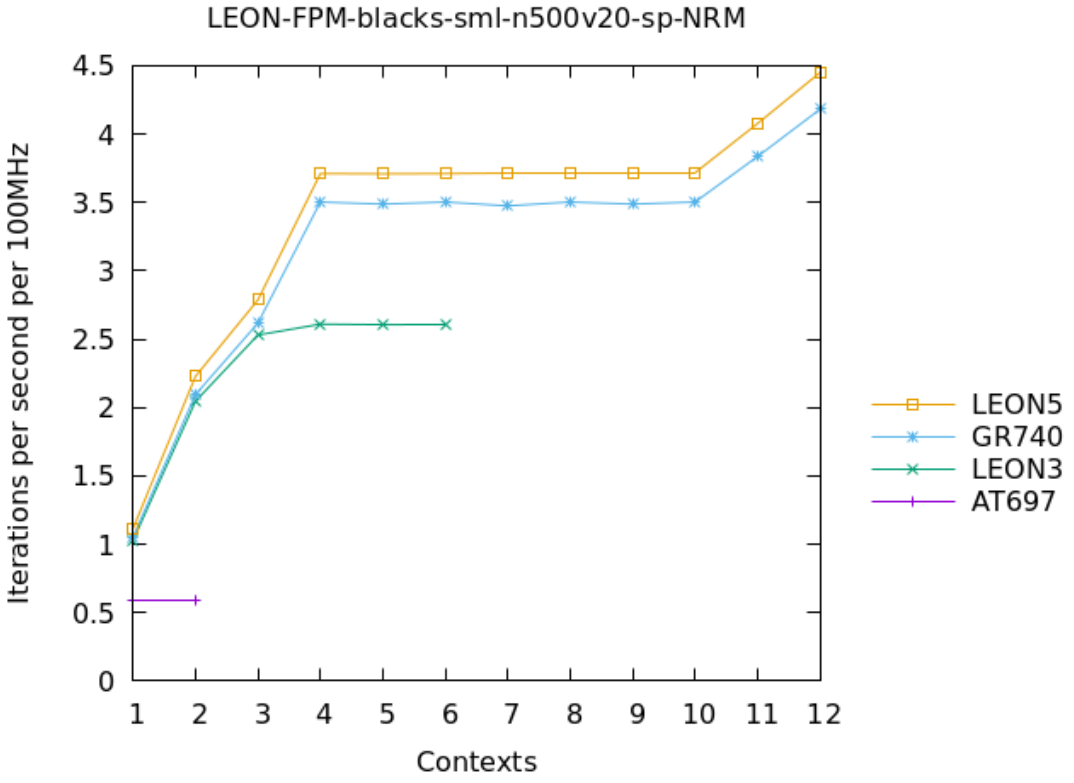

Figure 107: LEON - FPMark - blacks-sml-n500v20-sp, iterations per second at 100MHz. Higher values denote better performance.

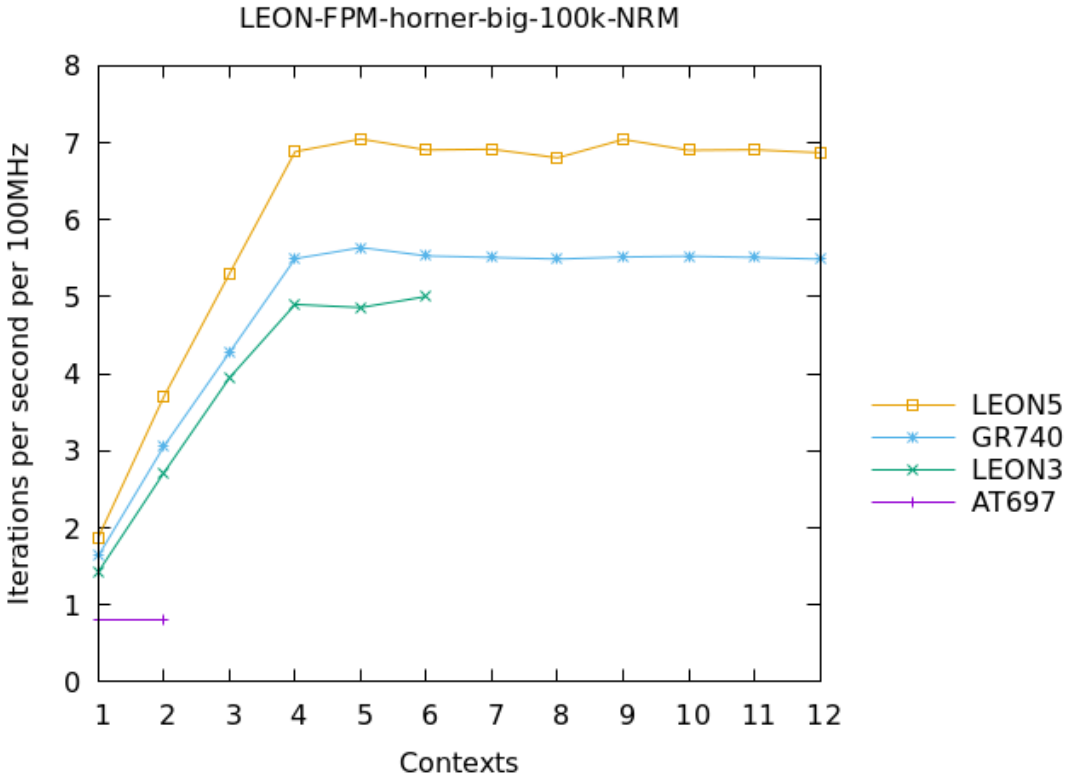

Figure 108: LEON - FPMark - horner-big-100k, iterations per second at 100MHz. Higher values denote better performance.

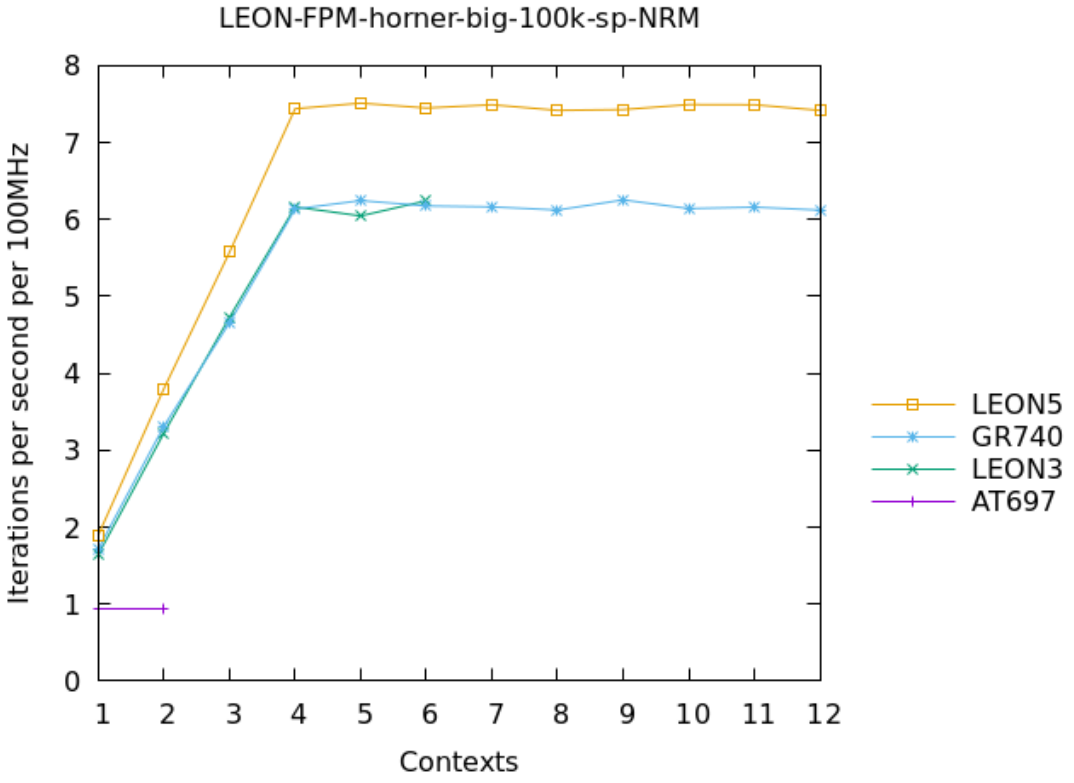

Figure 109: LEON - FPMark - horner-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

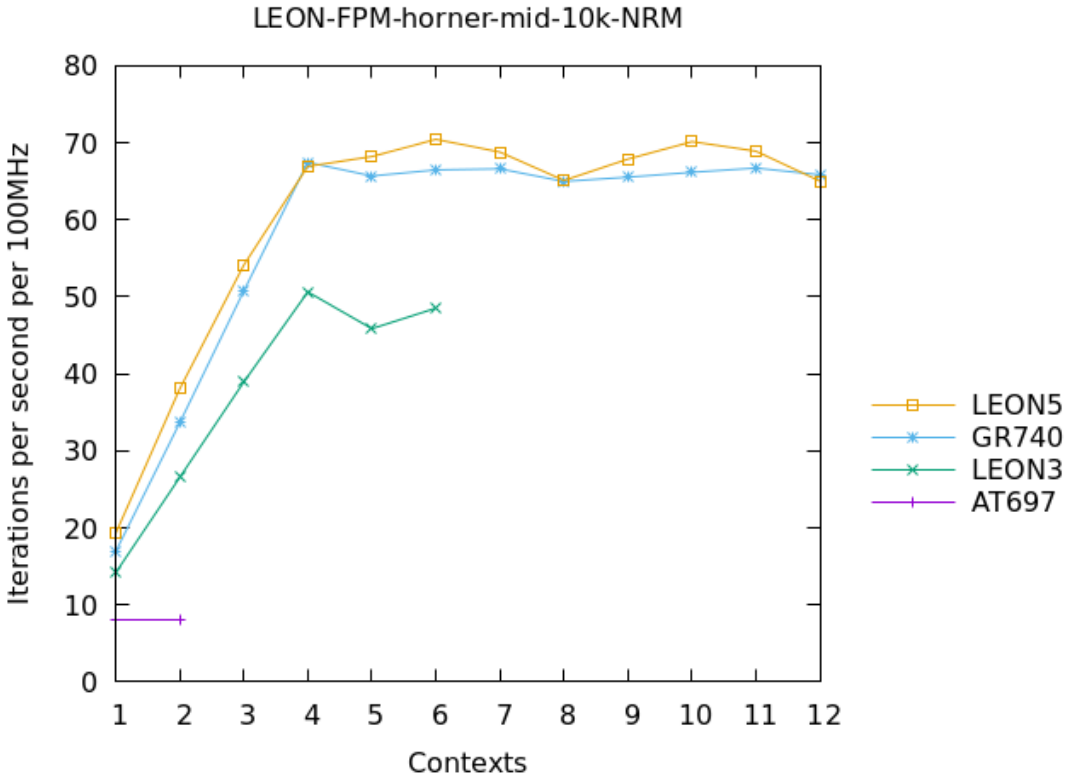

Figure 110: LEON - FPMark - horner-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

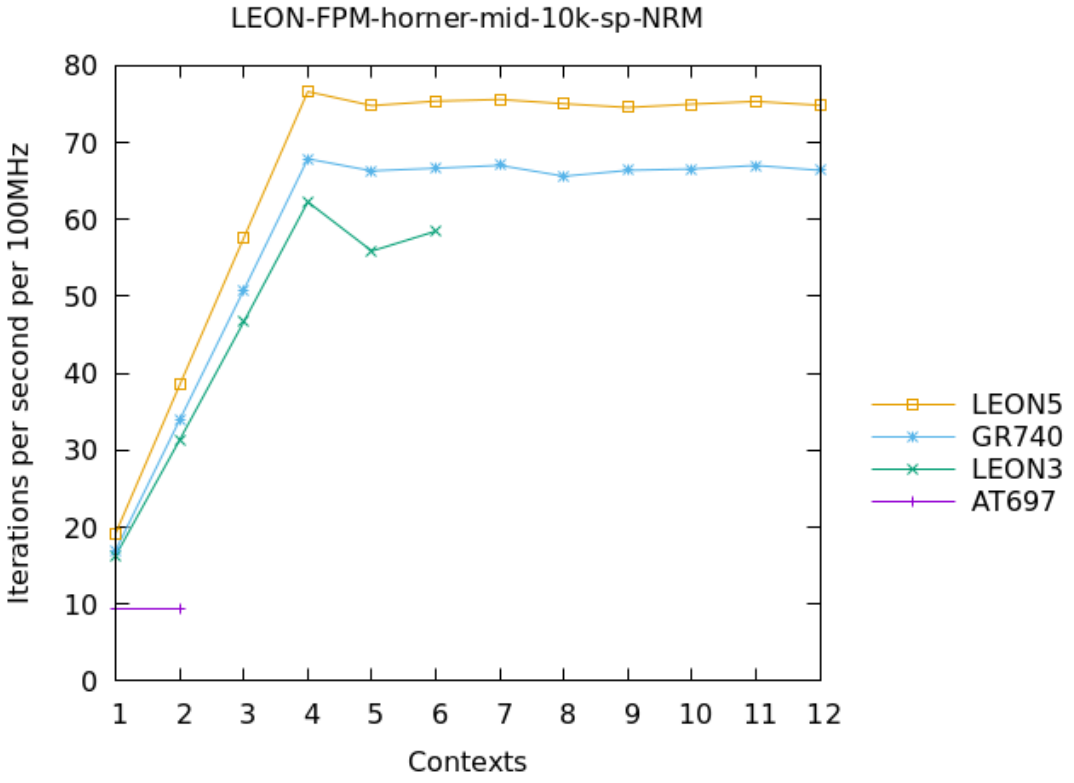

Figure 111: LEON - FPMark - horner-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

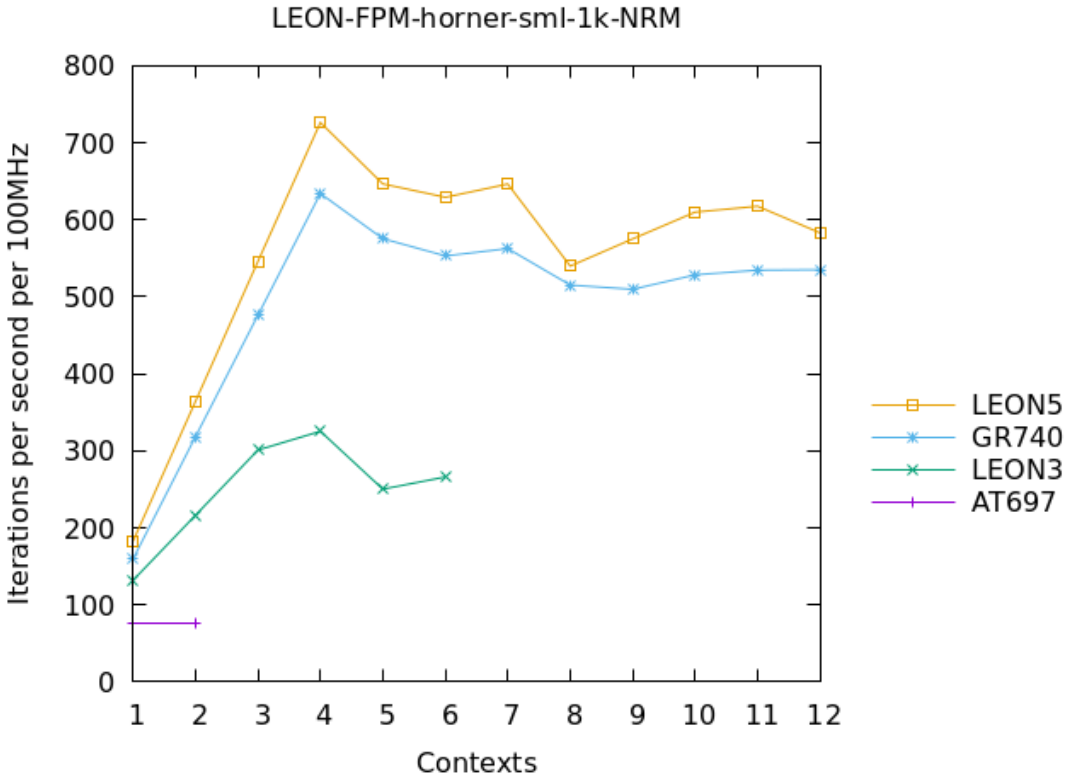

Figure 112: LEON - FPMark - horner-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

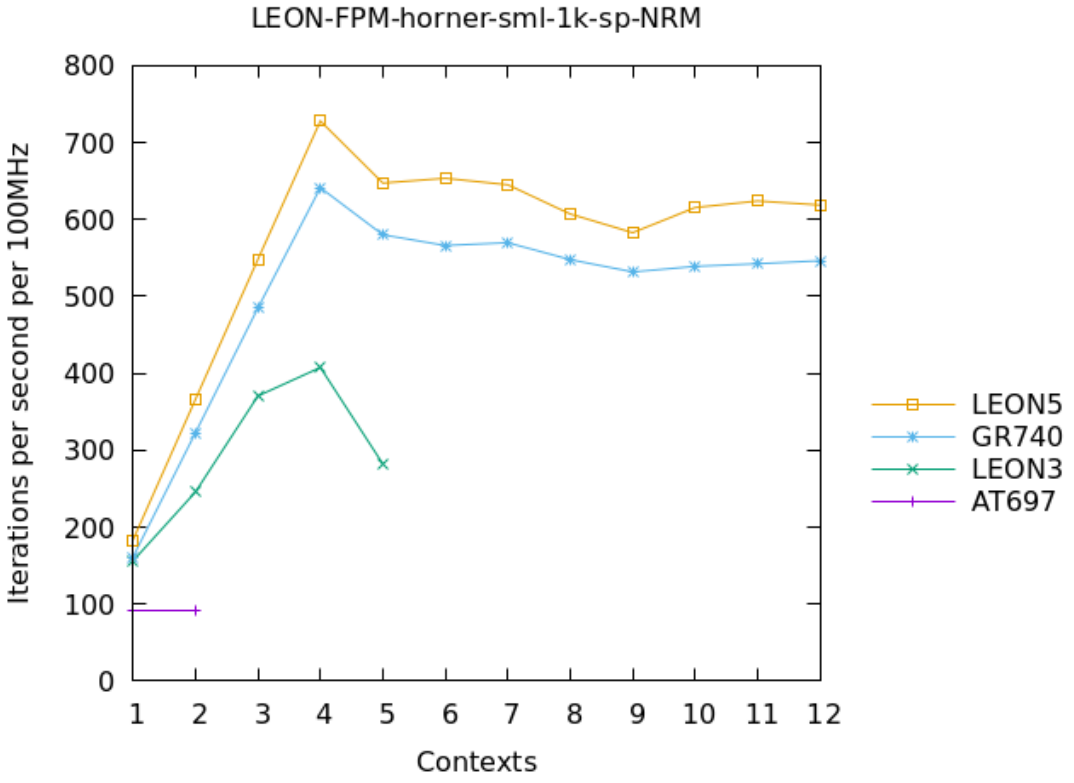

Figure 113: LEON - FPMark - horner-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

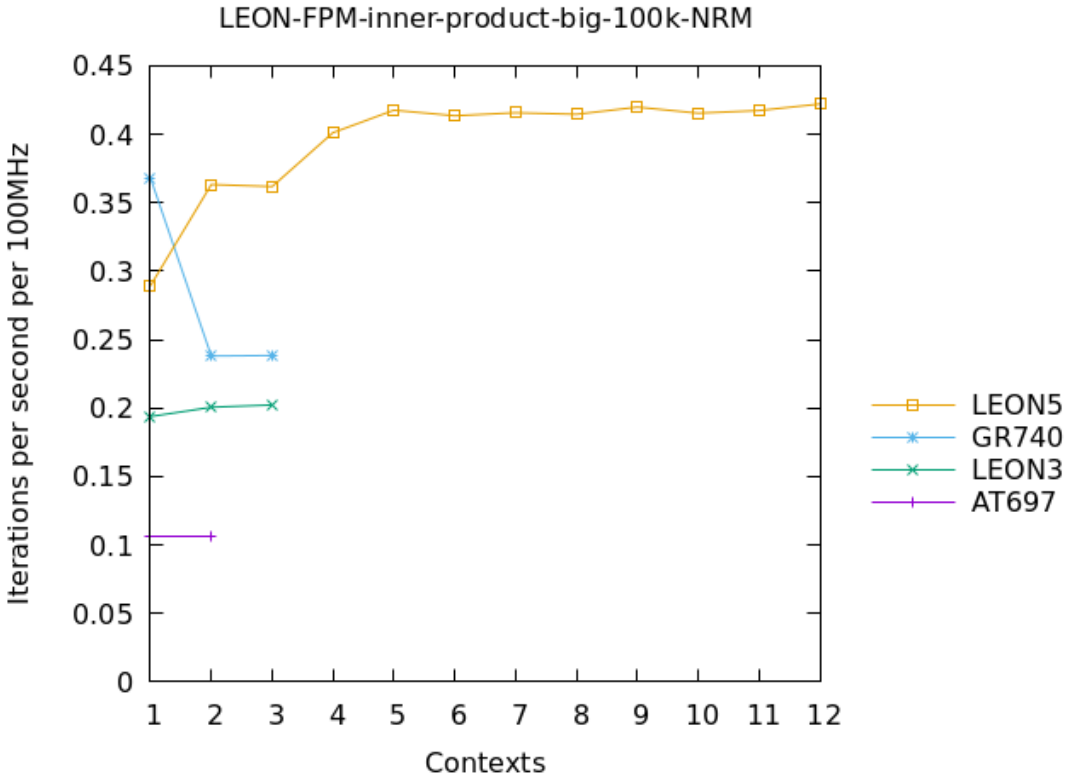

Figure 114: LEON - FPMark - inner-product-big-100k, iterations per second at 100MHz. Higher values denote better performance.

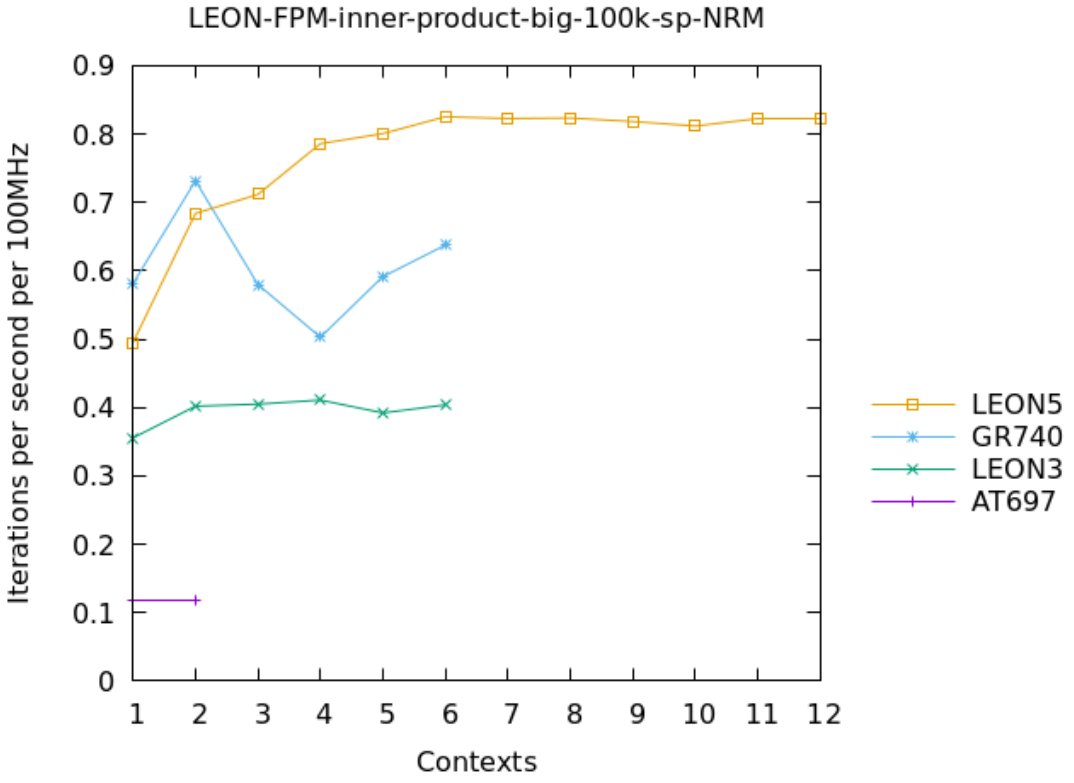

Figure 115: LEON - FPMark - inner-product-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

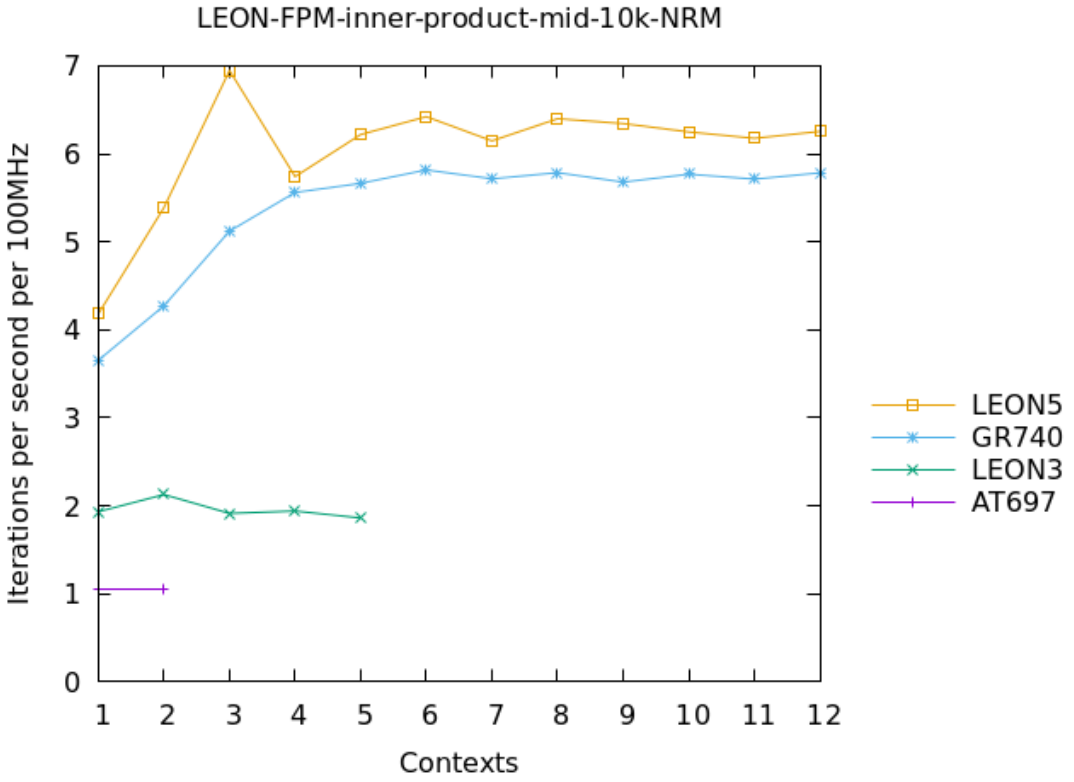

Figure 116: LEON - FPMark - inner-product-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

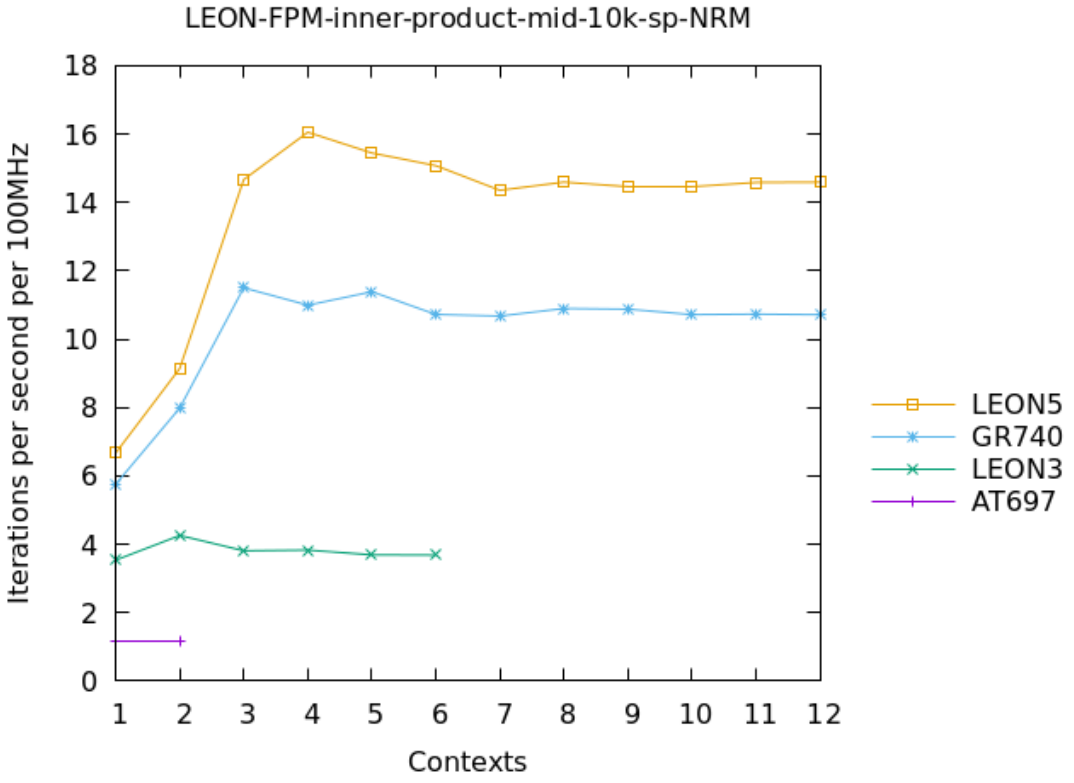

Figure 117: LEON - FPMark - inner-product-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

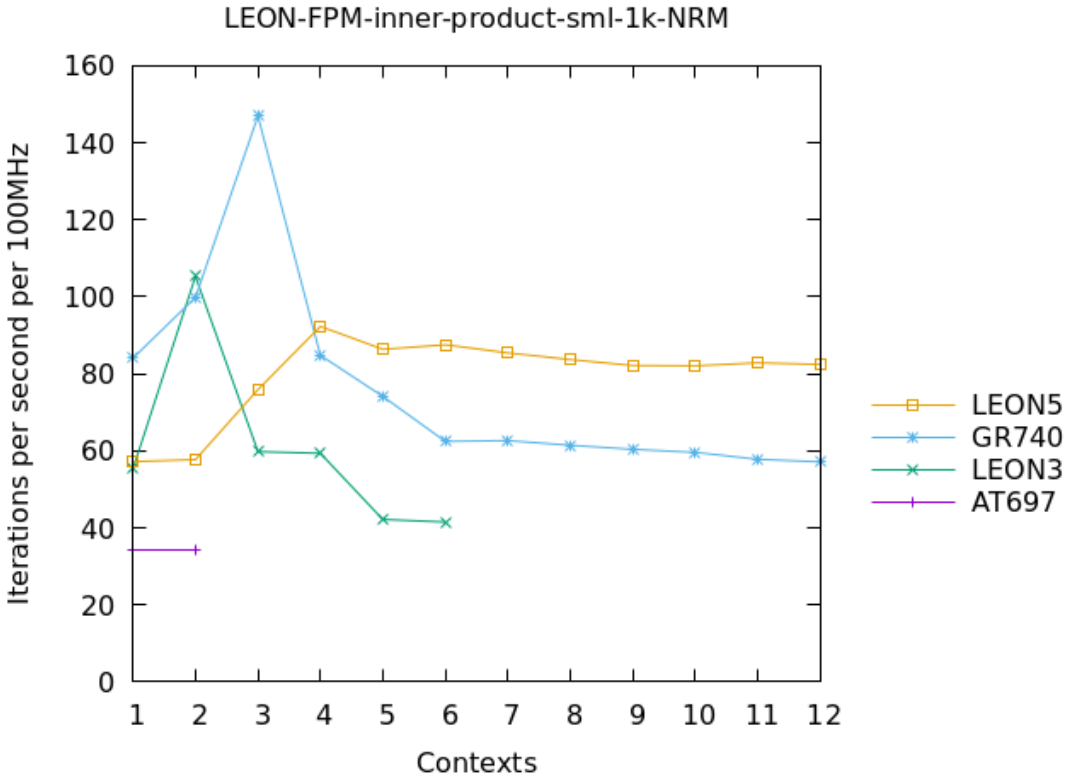

Figure 118: LEON - FPMark - inner-product-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

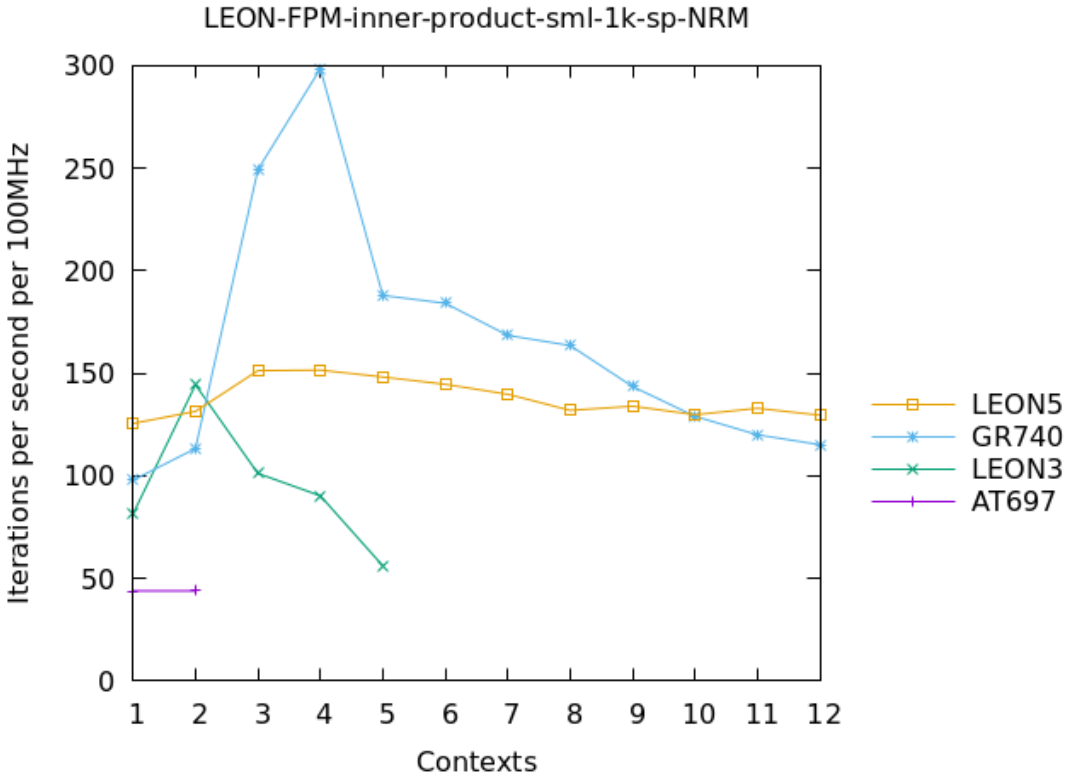

Figure 119: LEON - FPMark - inner-product-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

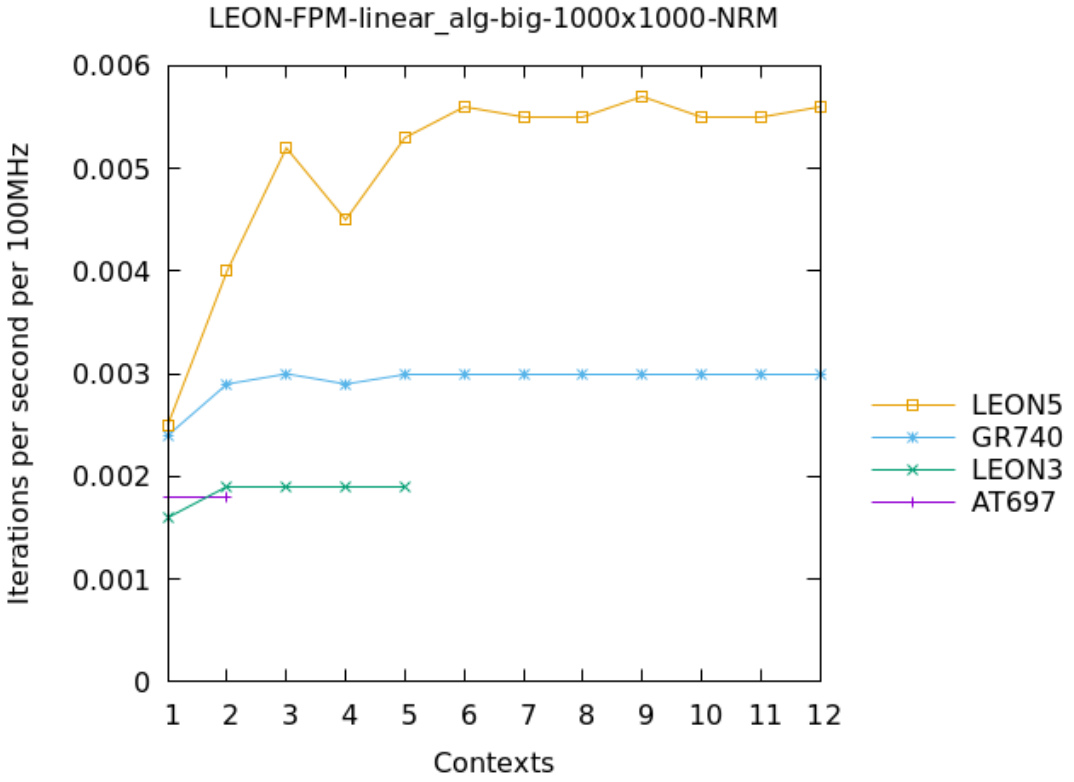

Figure 120: LEON - FPMark - linear_alg-big-1000x1000, iterations per second at 100MHz. Higher values denote better performance.

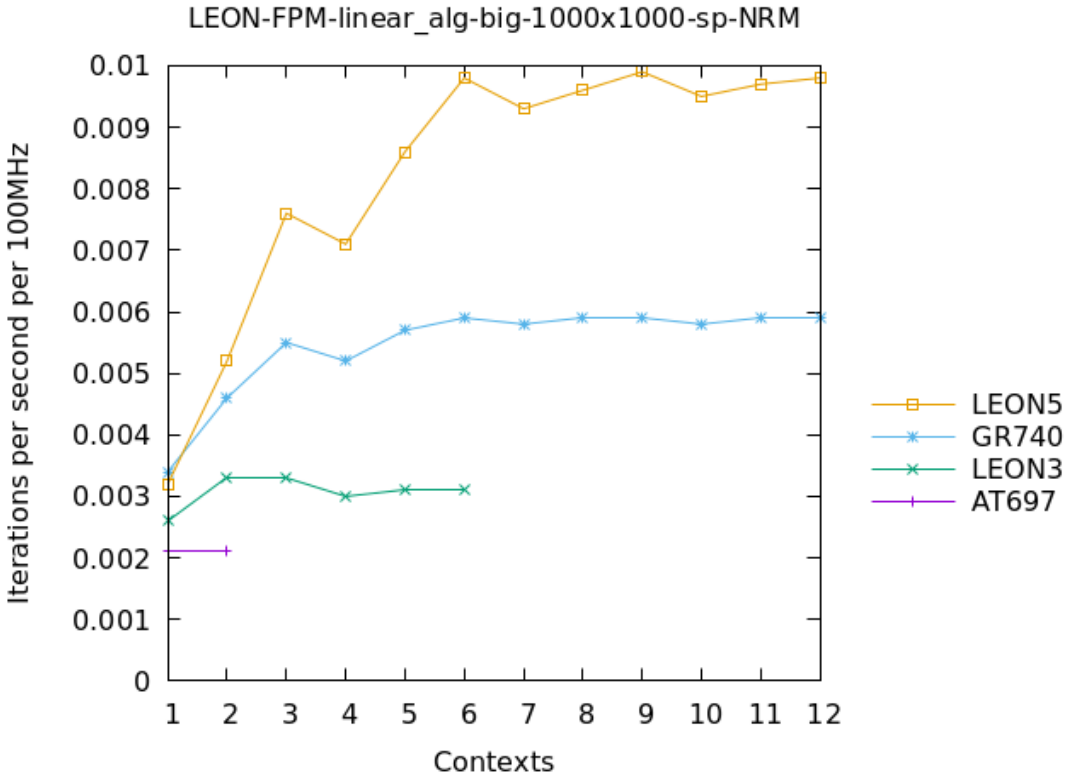

Figure 121: LEON - FPMark - linear_alg-big-1000x1000-sp, iterations per second at 100MHz. Higher values denote better performance.

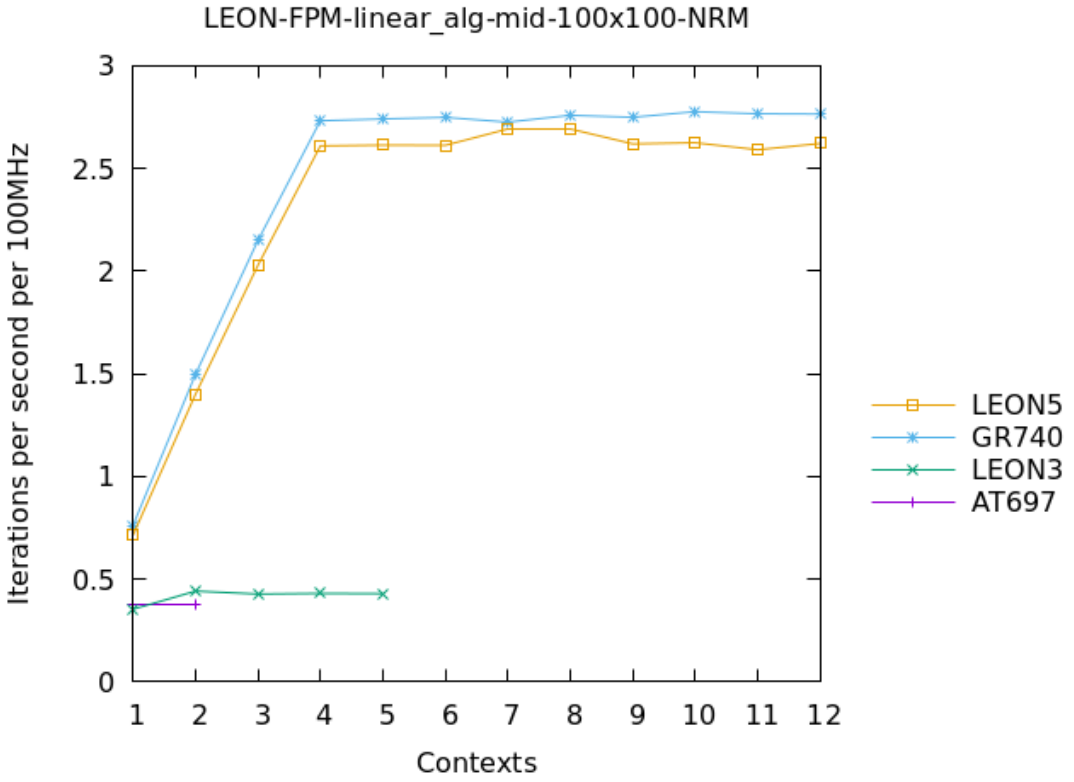

Figure 122: LEON - FPMark - linear_alg-mid-100x100, iterations per second at 100MHz. Higher values denote better performance.

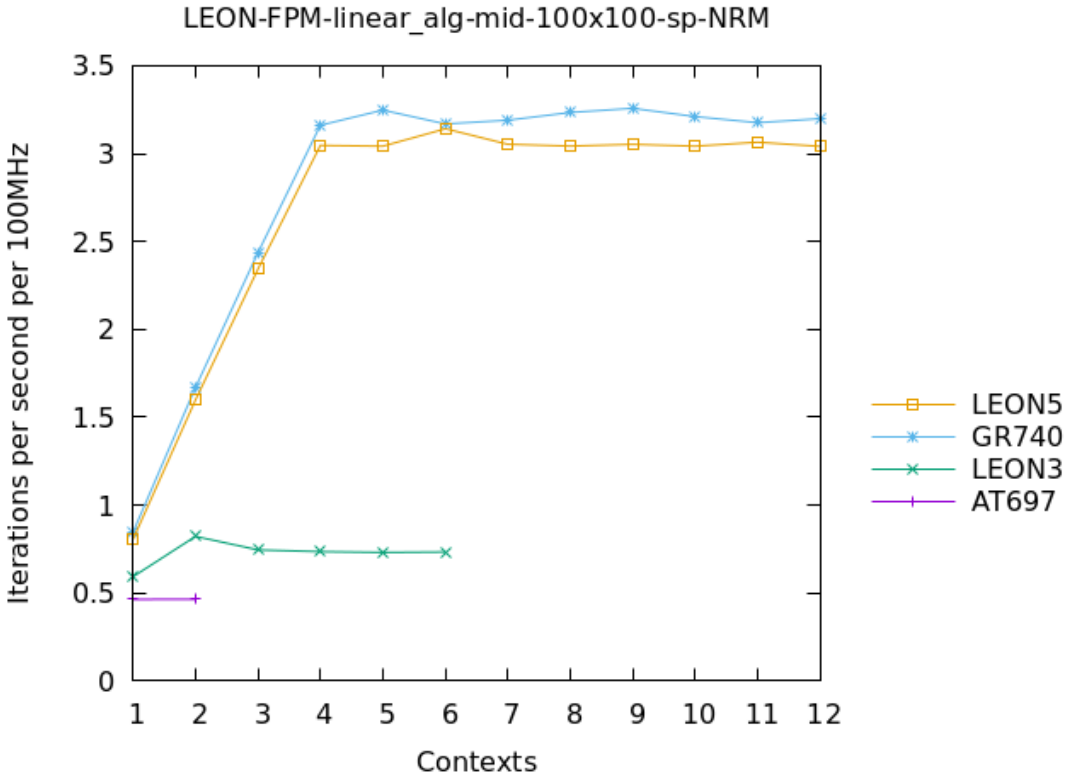

Figure 123: LEON - FPMark - linear_alg-mid-100x100-sp, iterations per second at 100MHz. Higher values denote better performance.

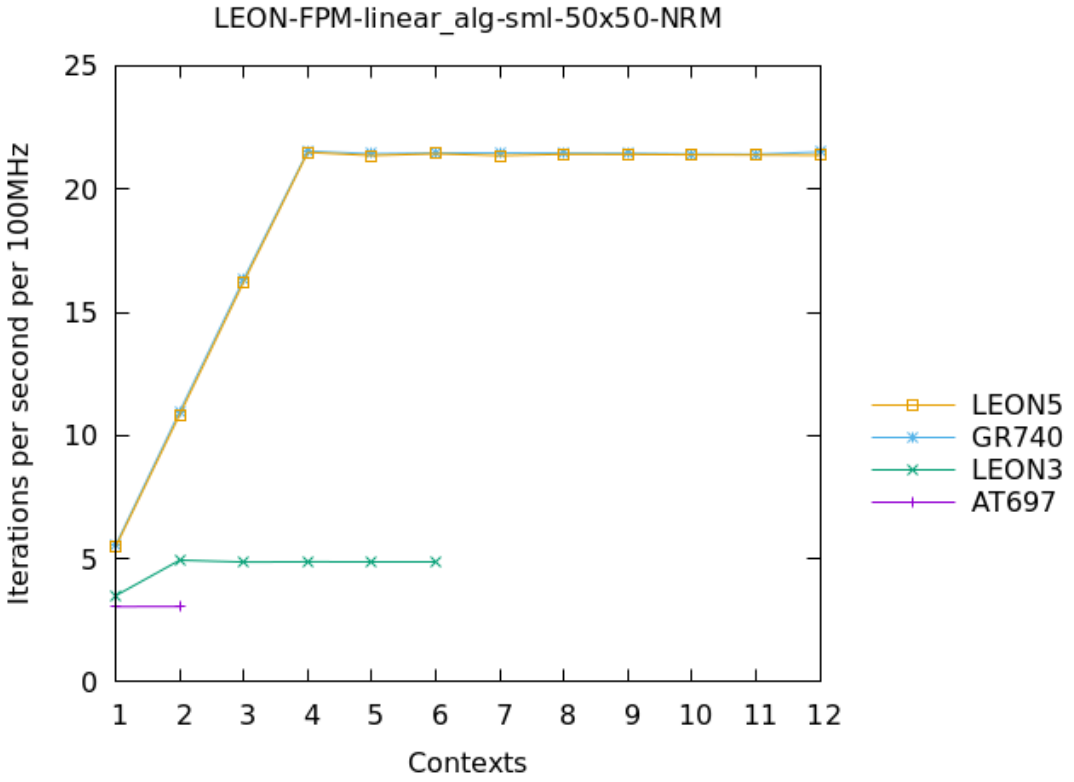

Figure 124: LEON - FPMark - linear_alg-sml-50x50, iterations per second at 100MHz. Higher values denote better performance.

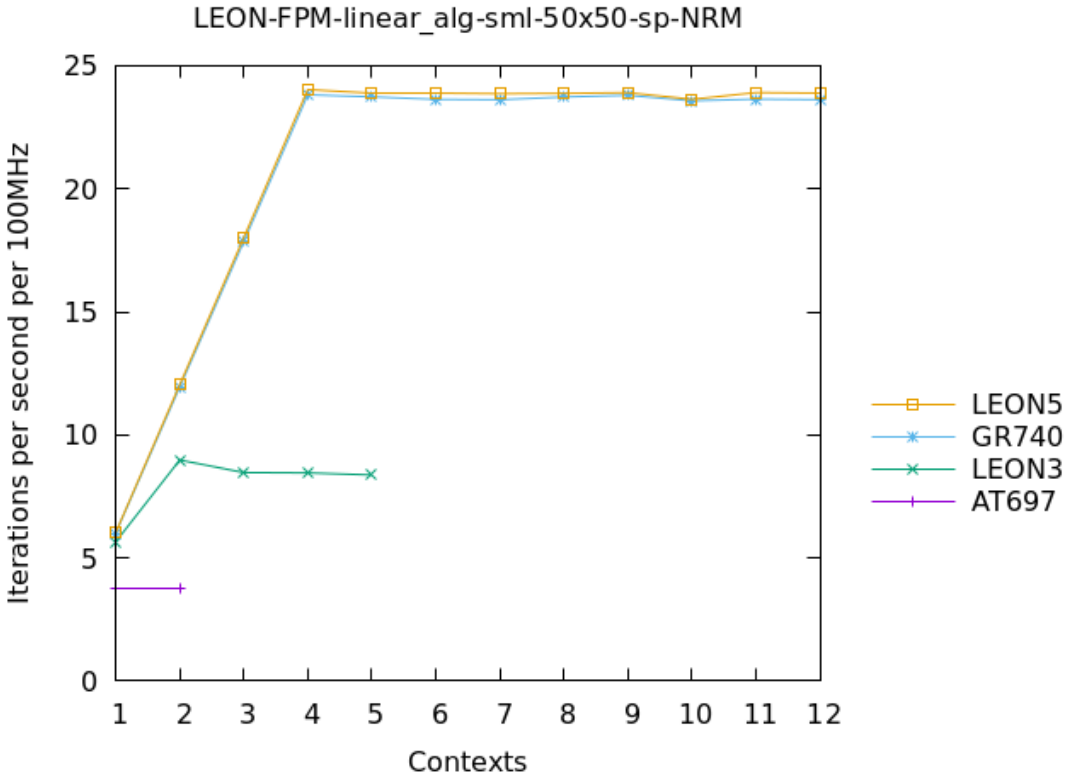

Figure 125: LEON - FPMark - linear_alg-sml-50x50-sp, iterations per second at 100MHz. Higher values denote better performance.

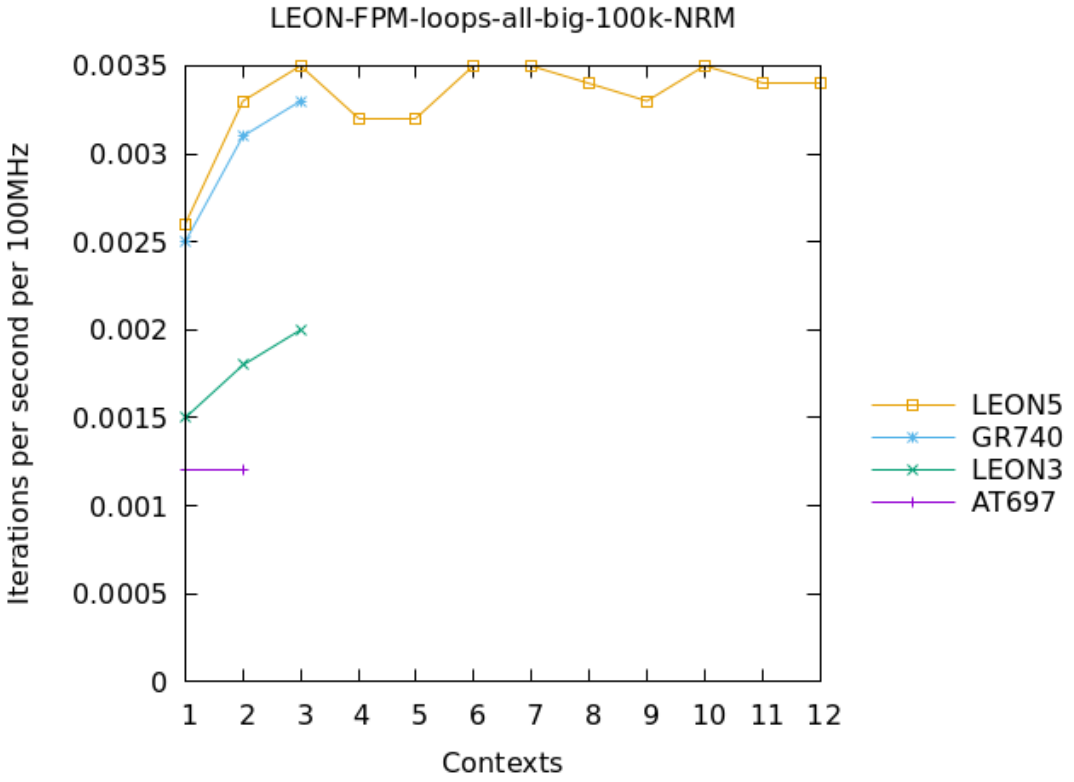

Figure 126: LEON - FPMark - loops-all-big-100k, iterations per second at 100MHz. Higher values denote better performance.

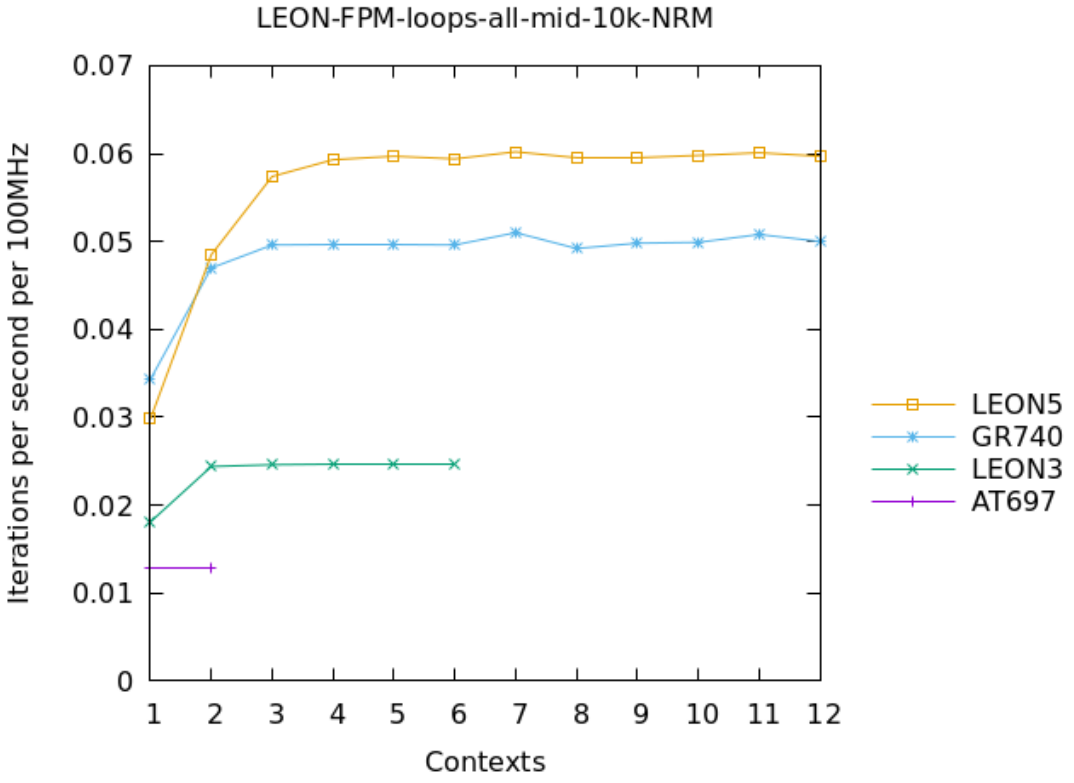

Figure 127: LEON - FPMark - loops-all-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

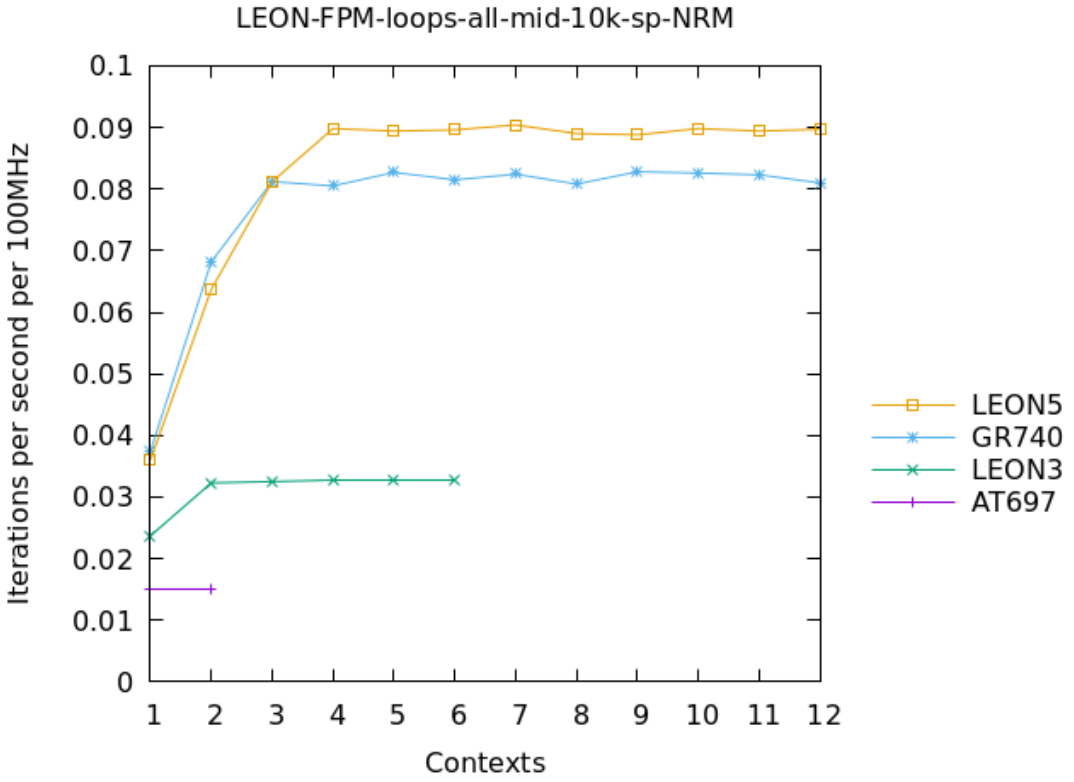

Figure 128: LEON - FPMark - loops-all-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

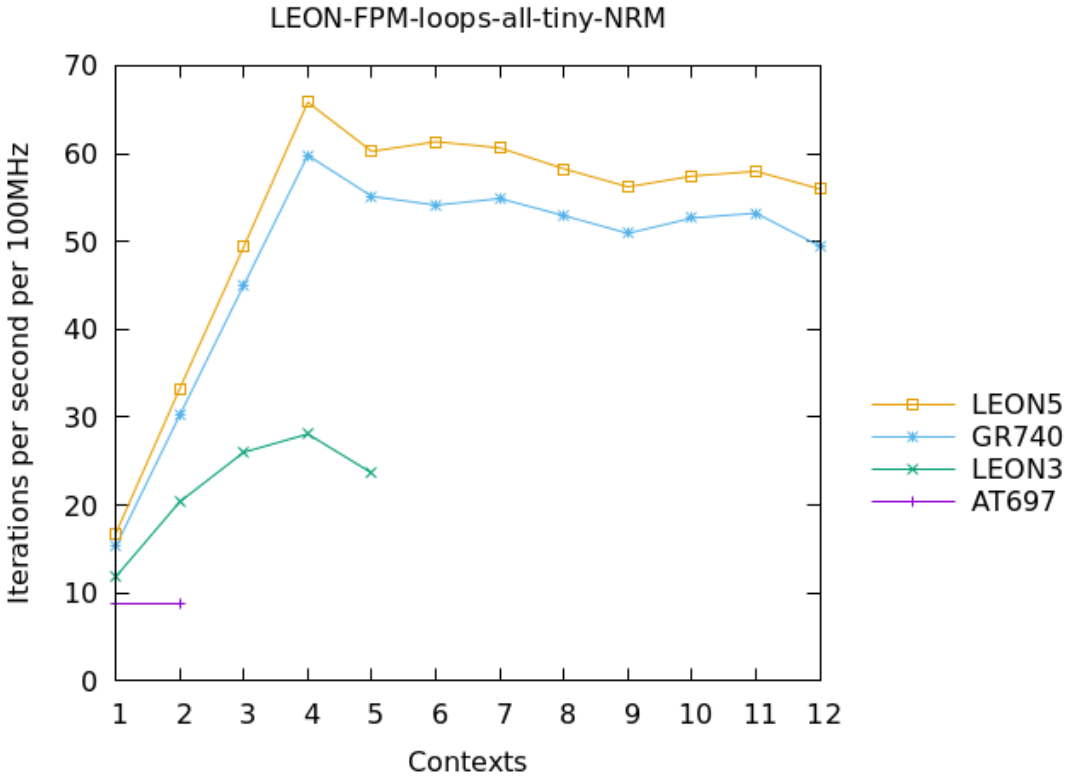

Figure 129: LEON - FPMark - loops-all-tiny, iterations per second at 100MHz. Higher values denote better performance.

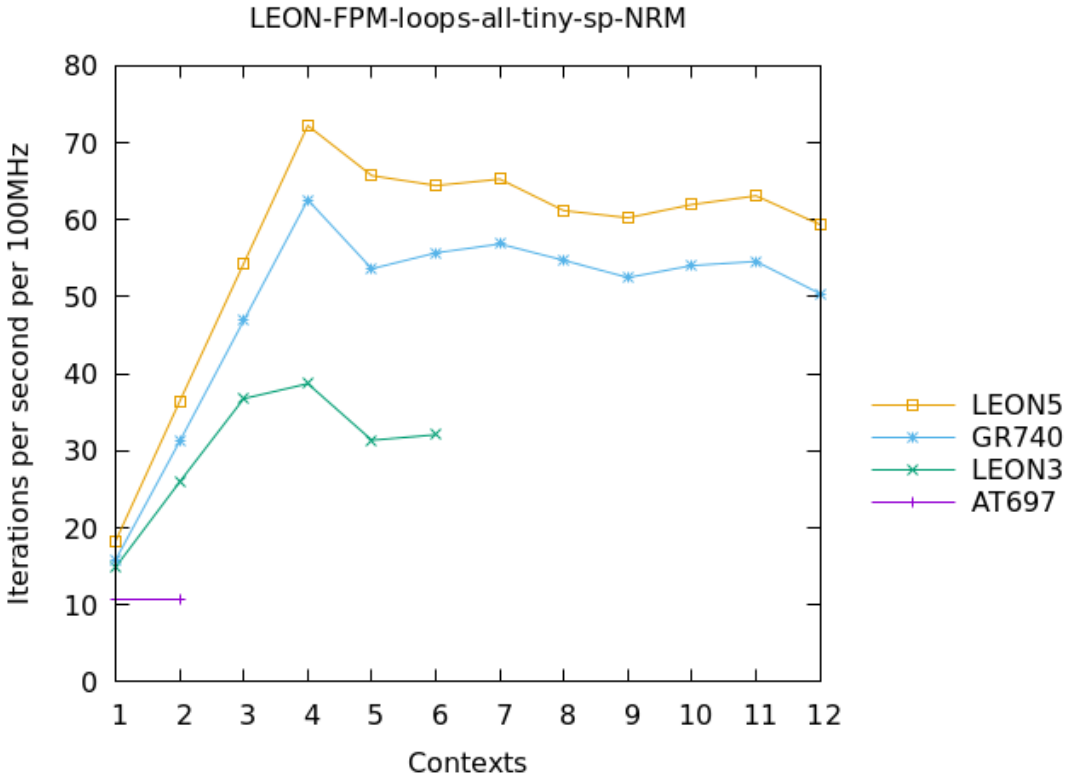

Figure 130: LEON - FPMark - loops-all-tiny-sp, iterations per second at 100MHz. Higher values denote better performance.

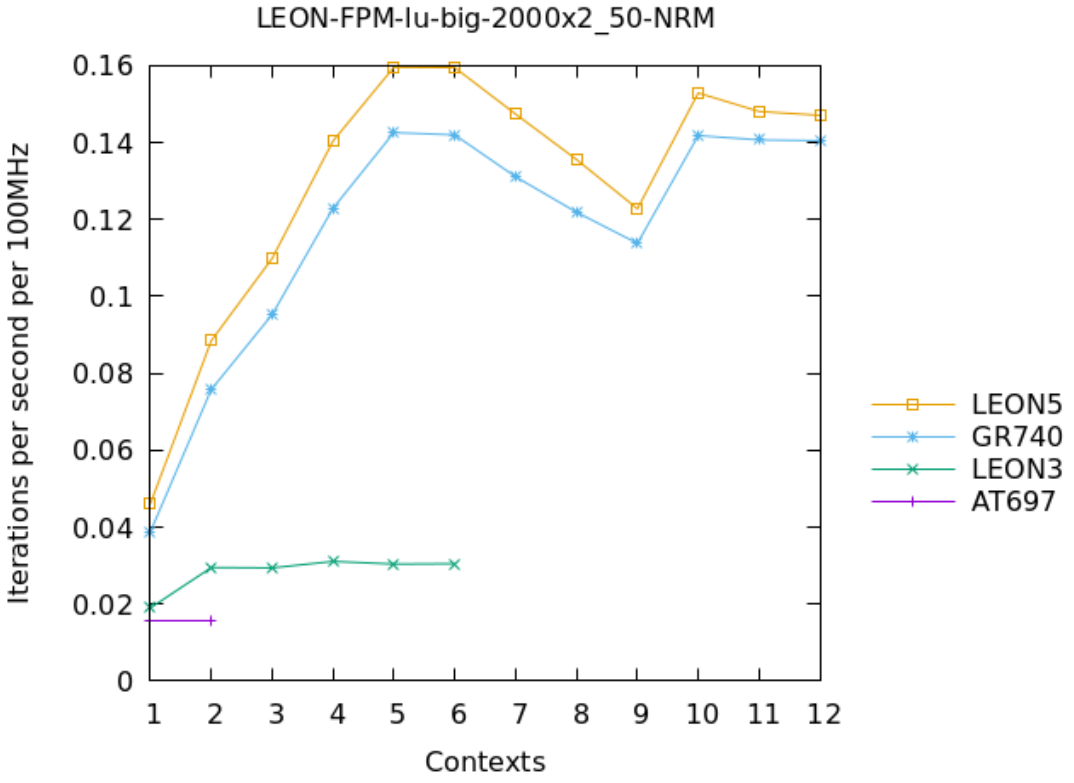

Figure 131: LEON - FPMark - lu-big-2000x2_50, iterations per second at 100MHz. Higher values denote better performance.

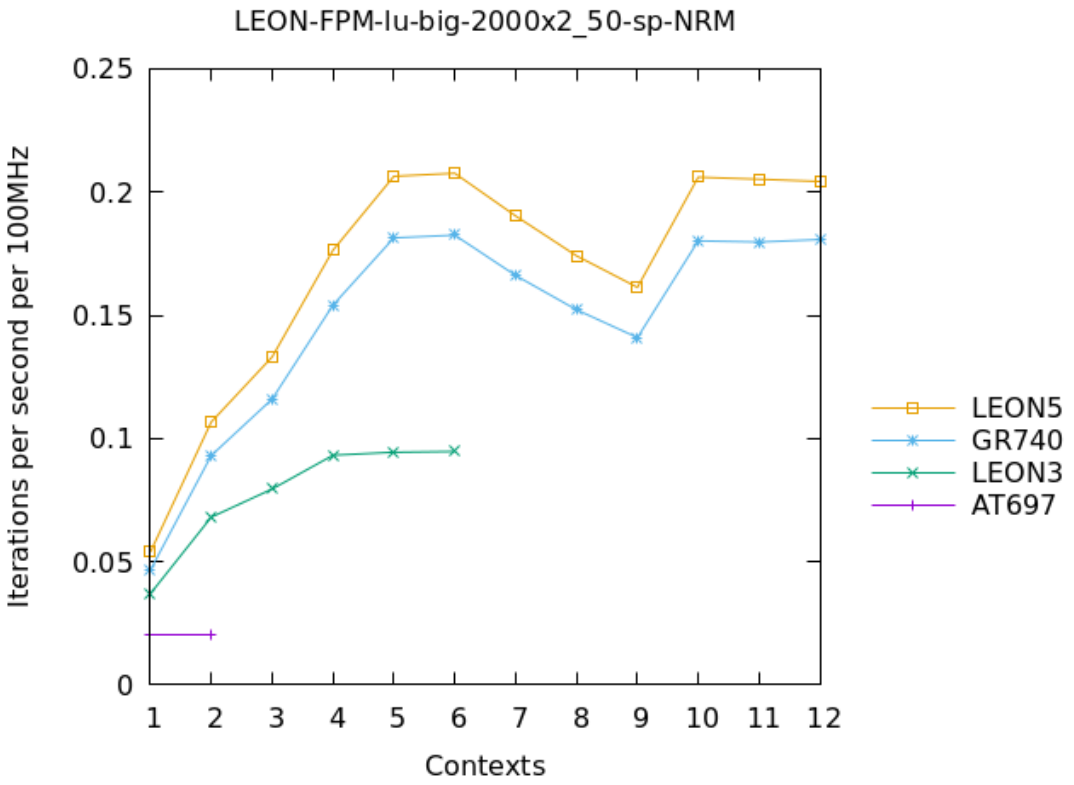

Figure 132: LEON - FPMark - lu-big-2000x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

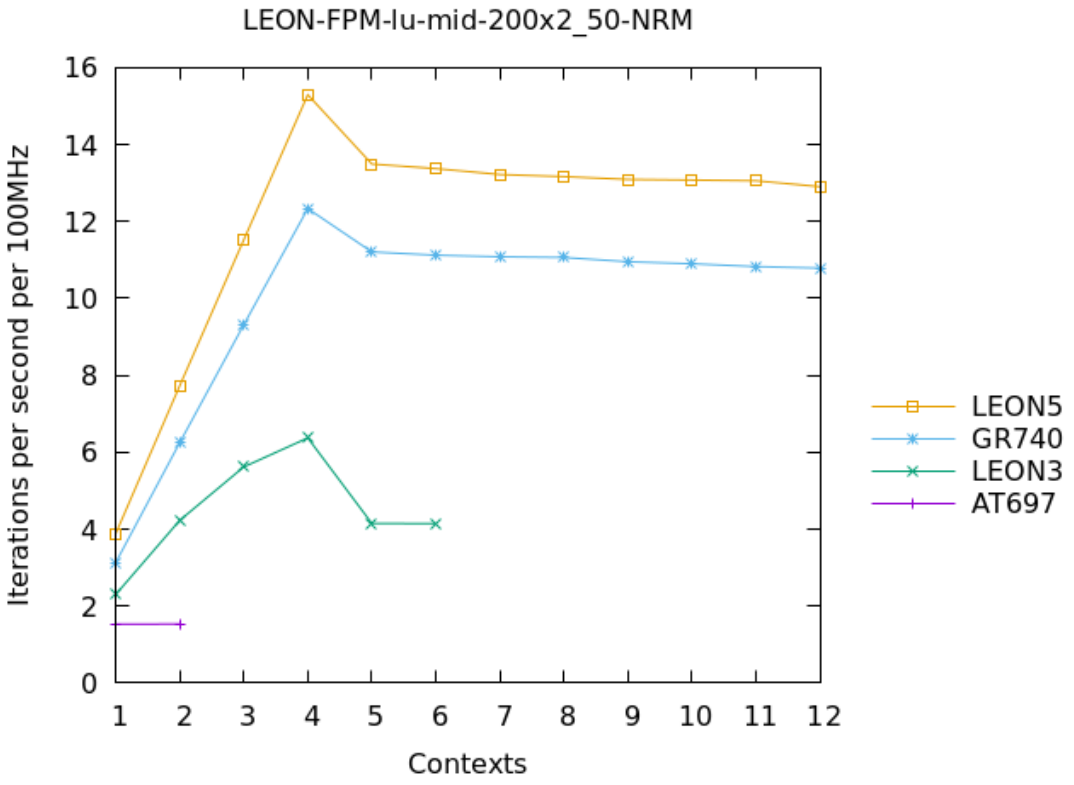

Figure 133: LEON - FPMark - lu-mid-200x2_50, iterations per second at 100MHz. Higher values denote better performance.

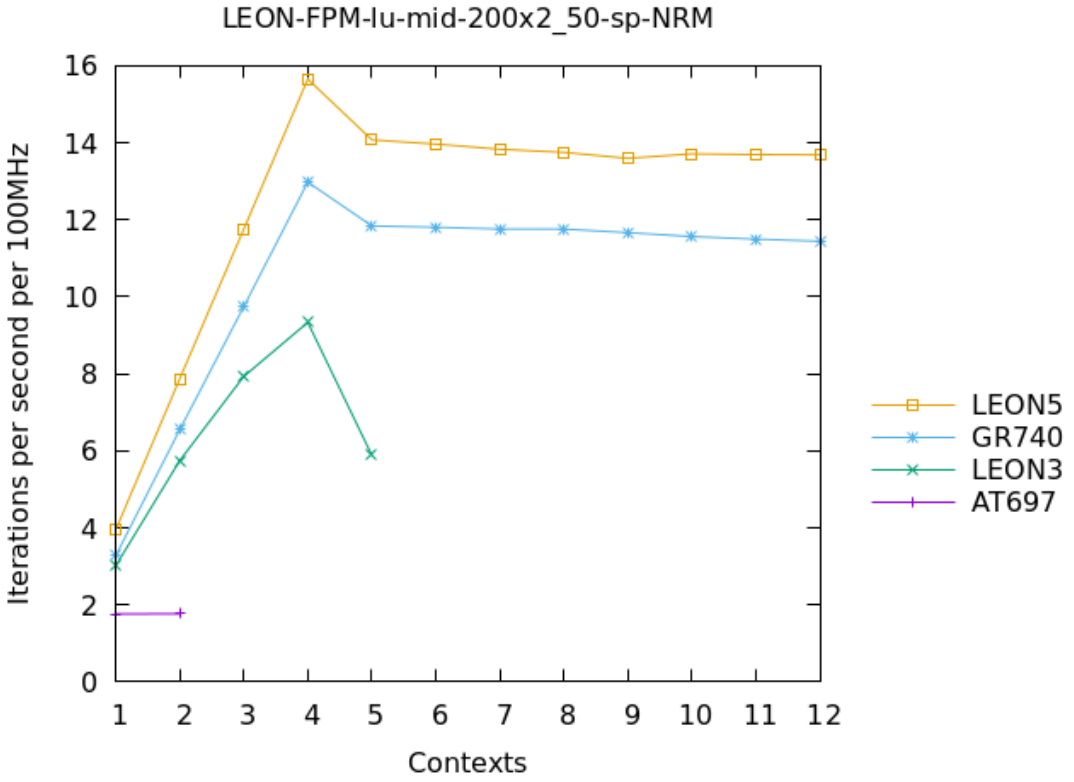

Figure 134: LEON - FPMark - lu-mid-200x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

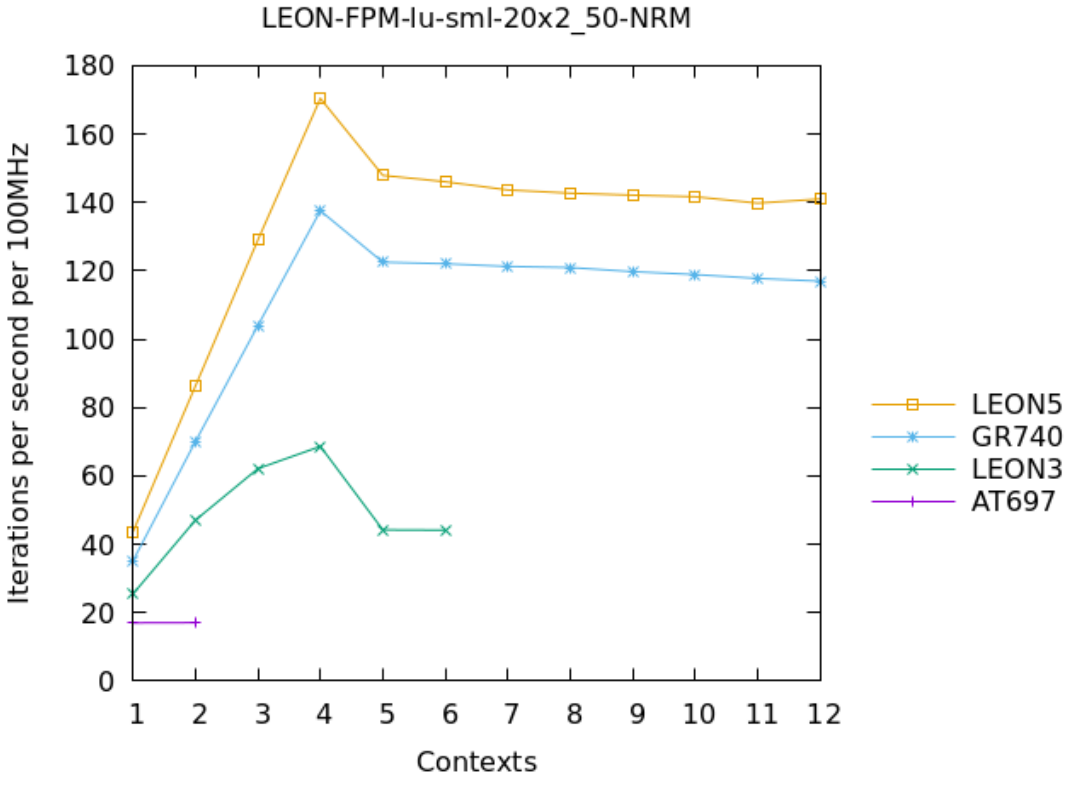

Figure 135: LEON - FPMark - lu-sml-20x2_50, iterations per second at 100MHz. Higher values denote better performance.

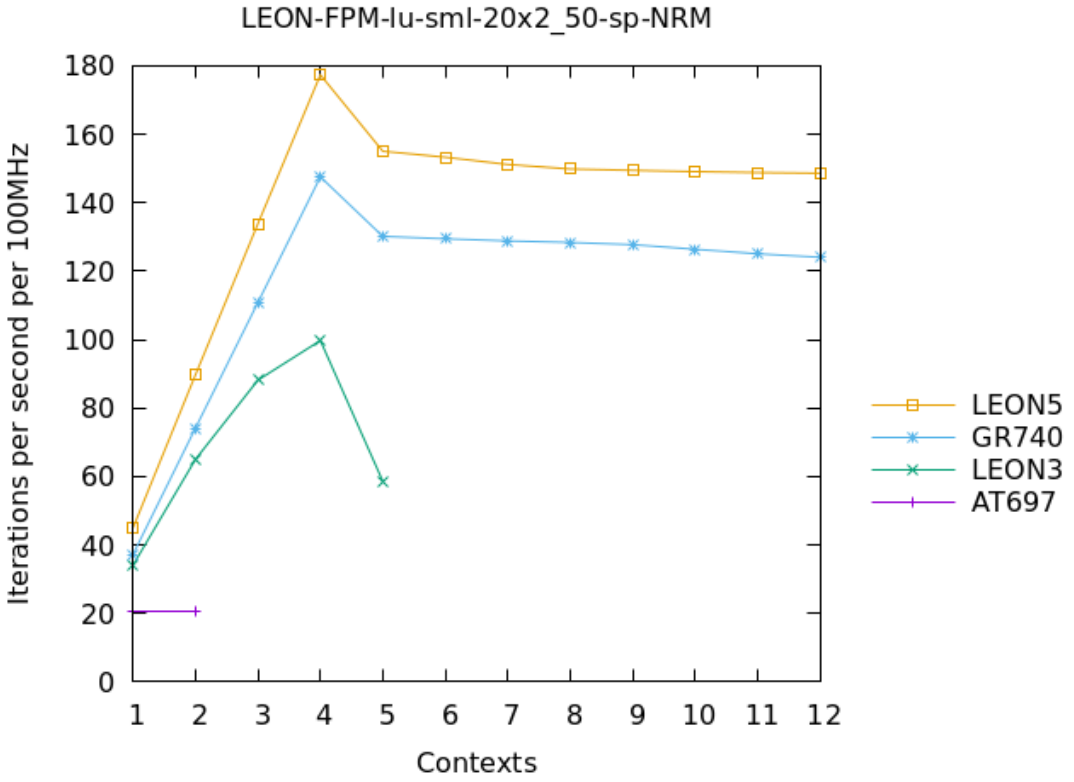

Figure 136: LEON - FPMark - lu-sml-20x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

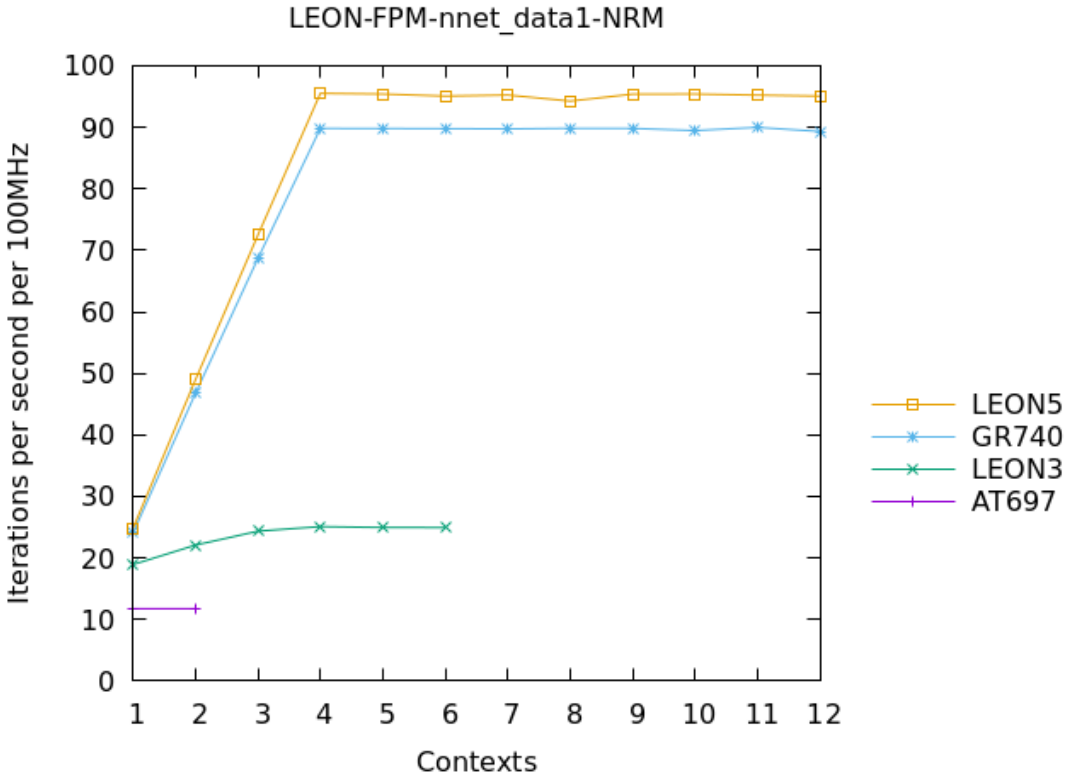

Figure 137: LEON - FPMark - nnet_data1, iterations per second at 100MHz. Higher values denote better performance.

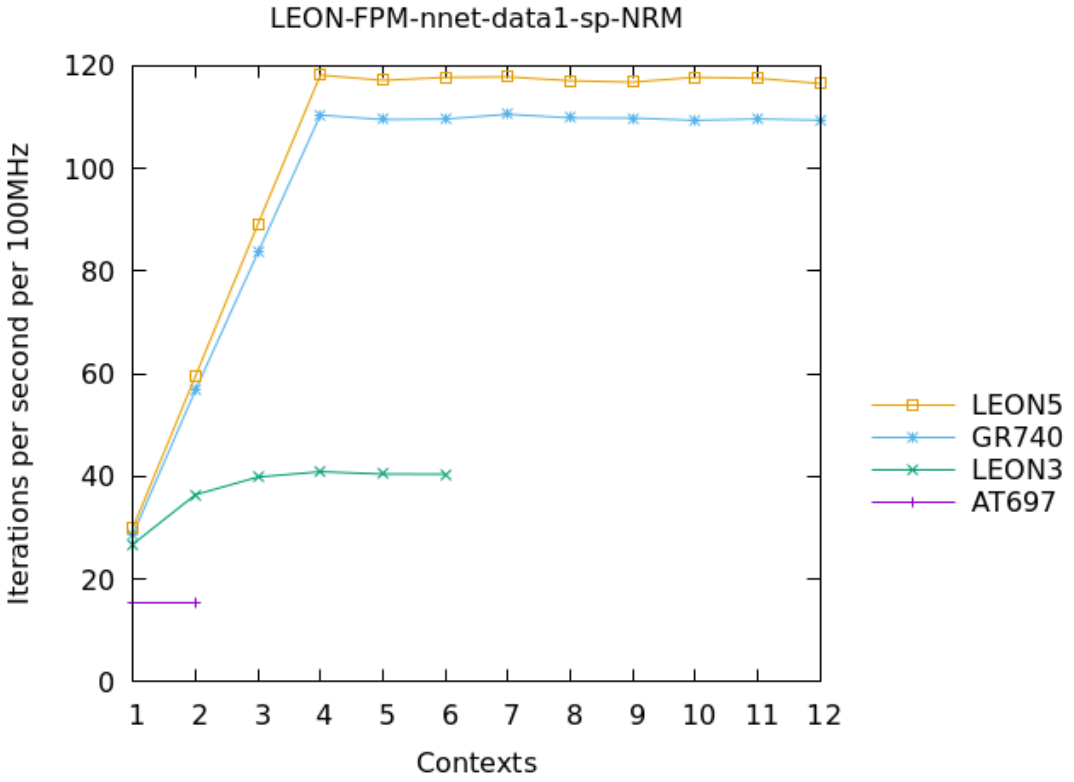

Figure 138: LEON - FPMark - nnet-data1-sp, iterations per second at 100MHz. Higher values denote better performance.

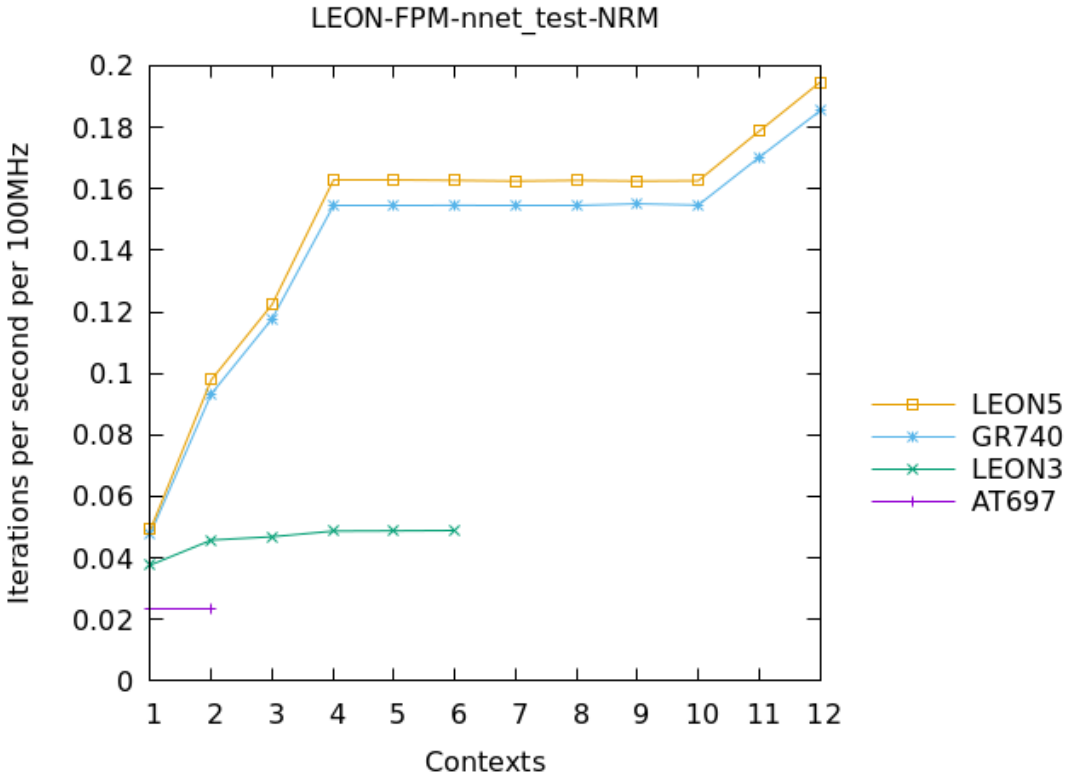

Figure 139: LEON - FPMark - nnet_test, iterations per second at 100MHz. Higher values denote better performance.

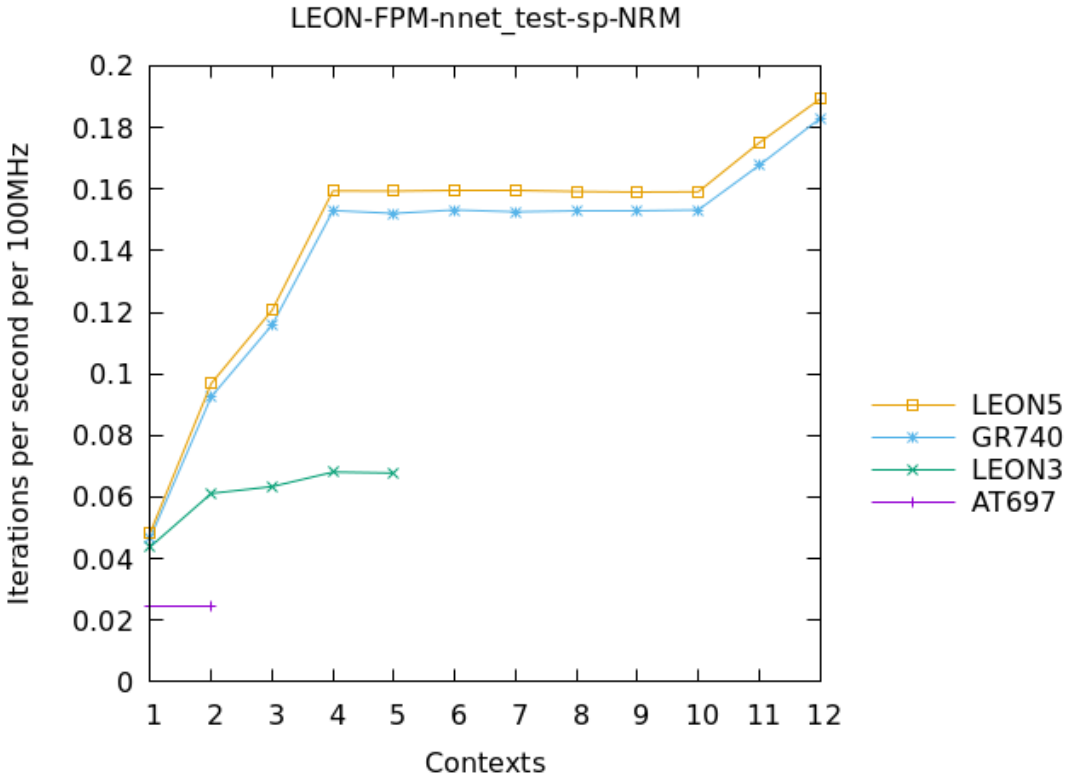

Figure 140: LEON - FPMark - nnet_test-sp, iterations per second at 100MHz. Higher values denote better performance.

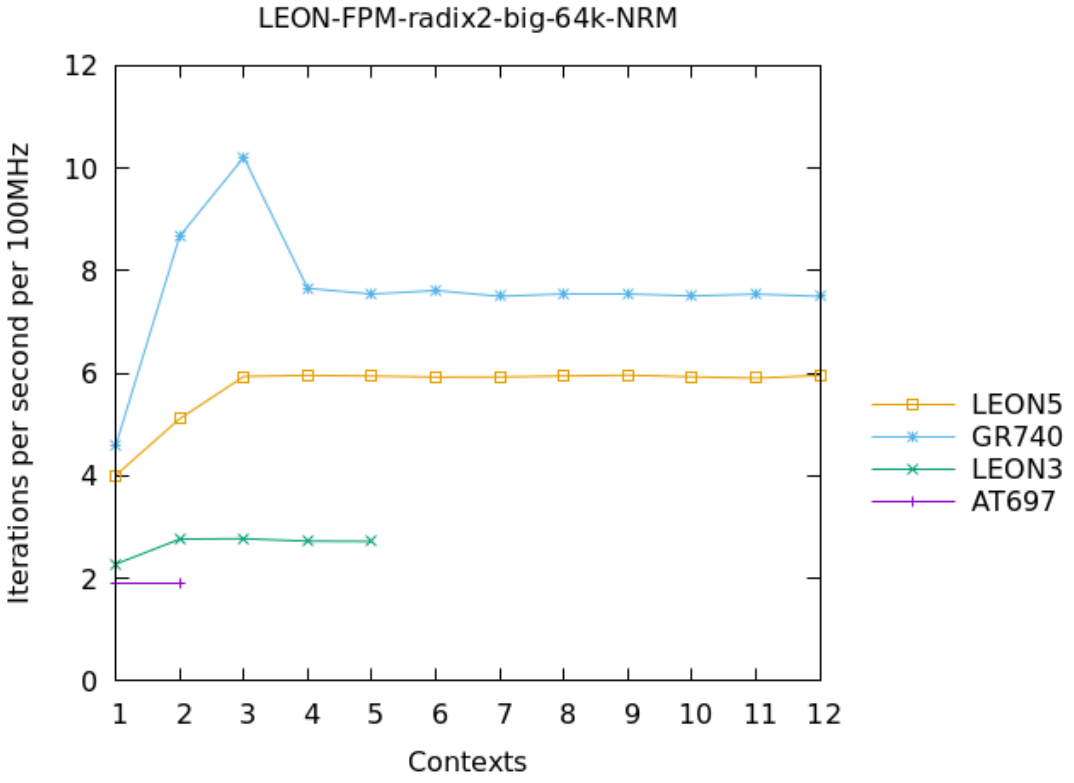

Figure 141: LEON - FPMark - radix2-big-64k, iterations per second at 100MHz. Higher values denote better performance.

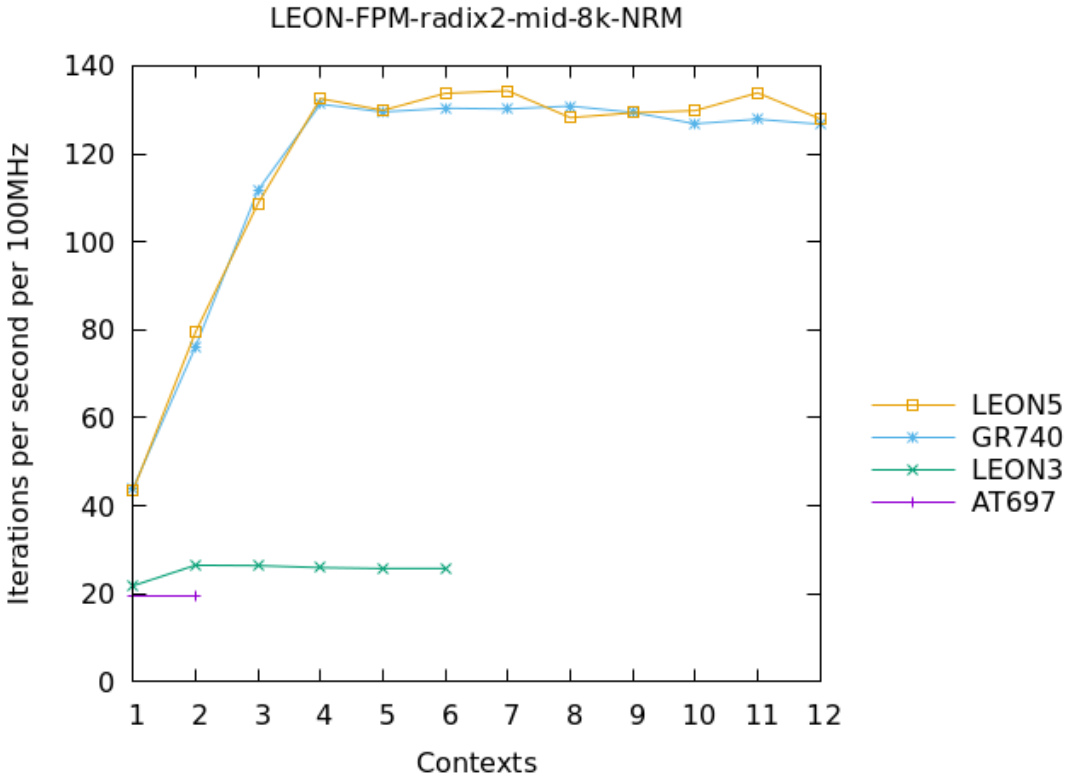

Figure 142: LEON - FPMark - radix2-mid-8k, iterations per second at 100MHz. Higher values denote better performance.

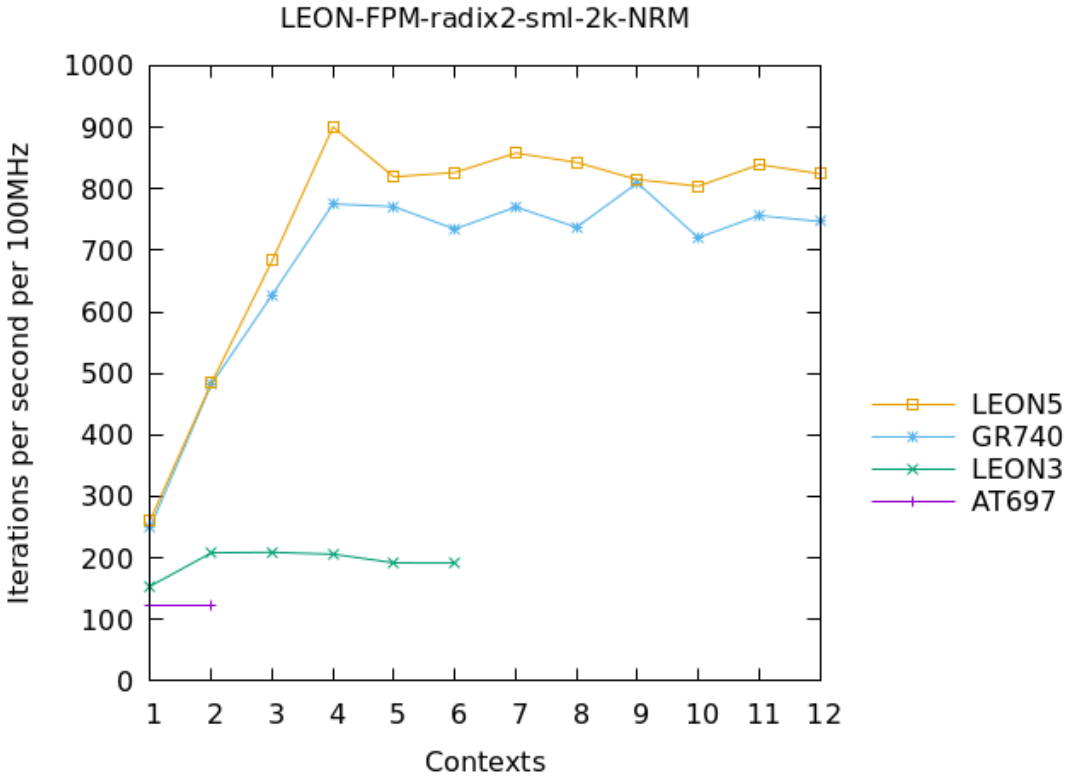

Figure 143: LEON - FPMark - radix2-sml-2k, iterations per second at 100MHz. Higher values denote better performance.

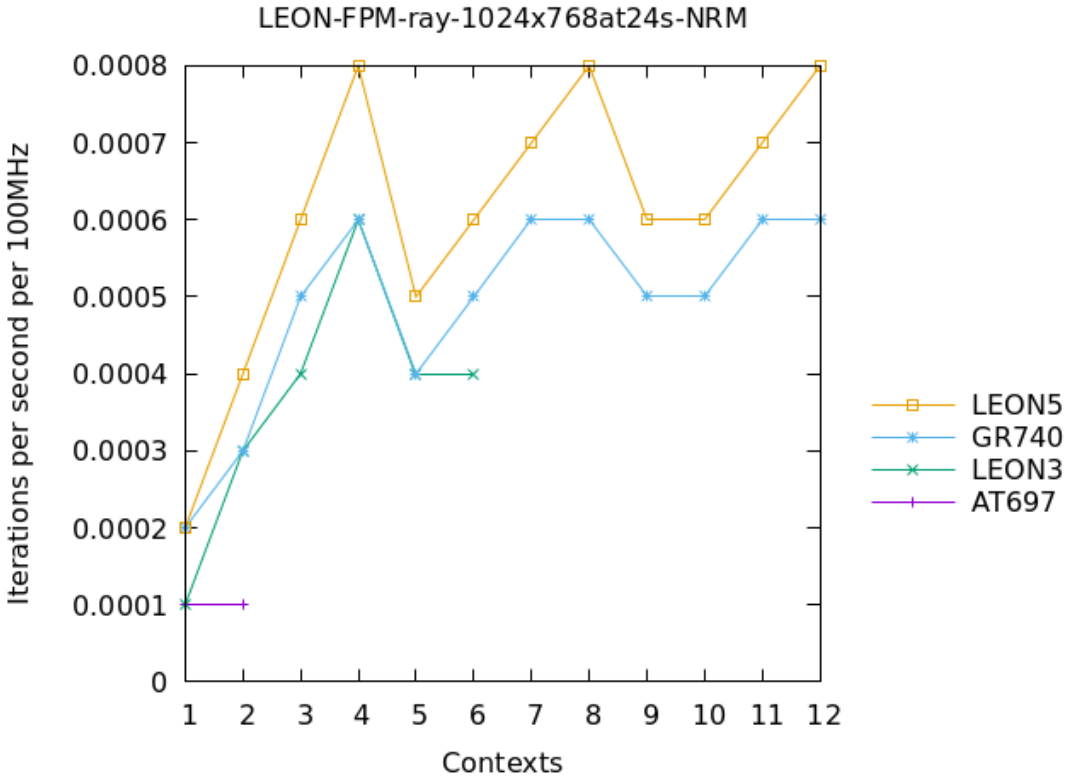

Figure 144: LEON - FPMark - ray-1024x768at24s, iterations per second at 100MHz. Higher values denote better performance.

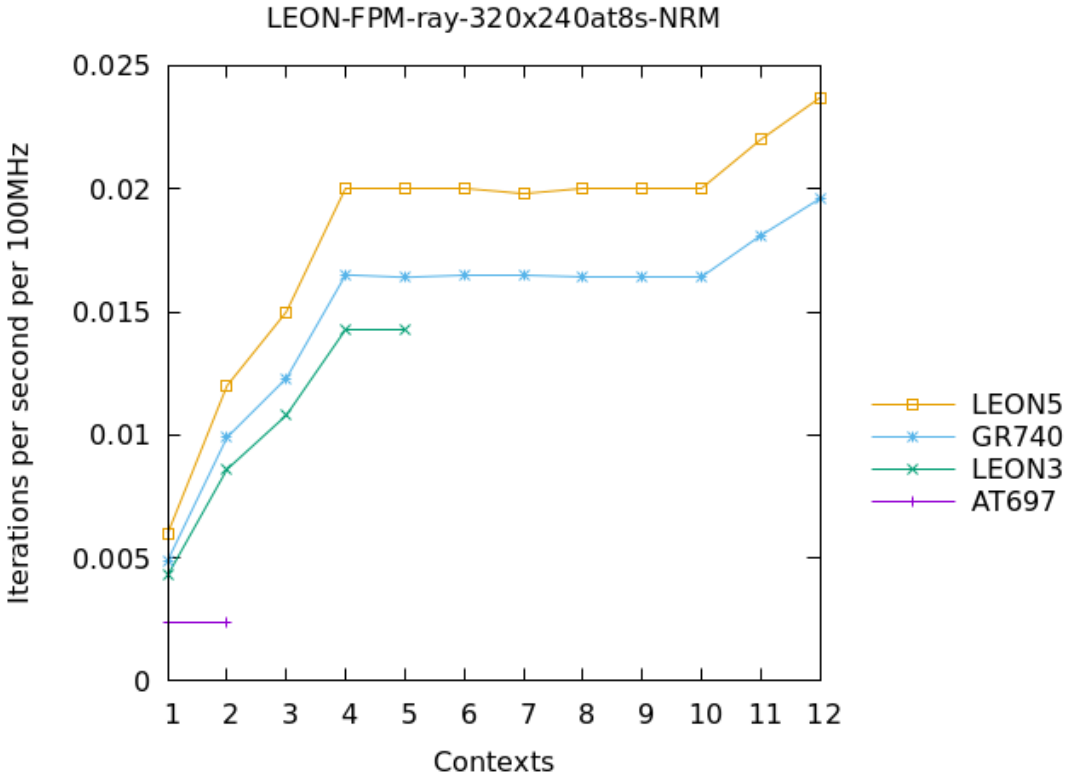

Figure 145: LEON - FPMark - ray-320x240at8s, iterations per second at 100MHz. Higher values denote better performance.

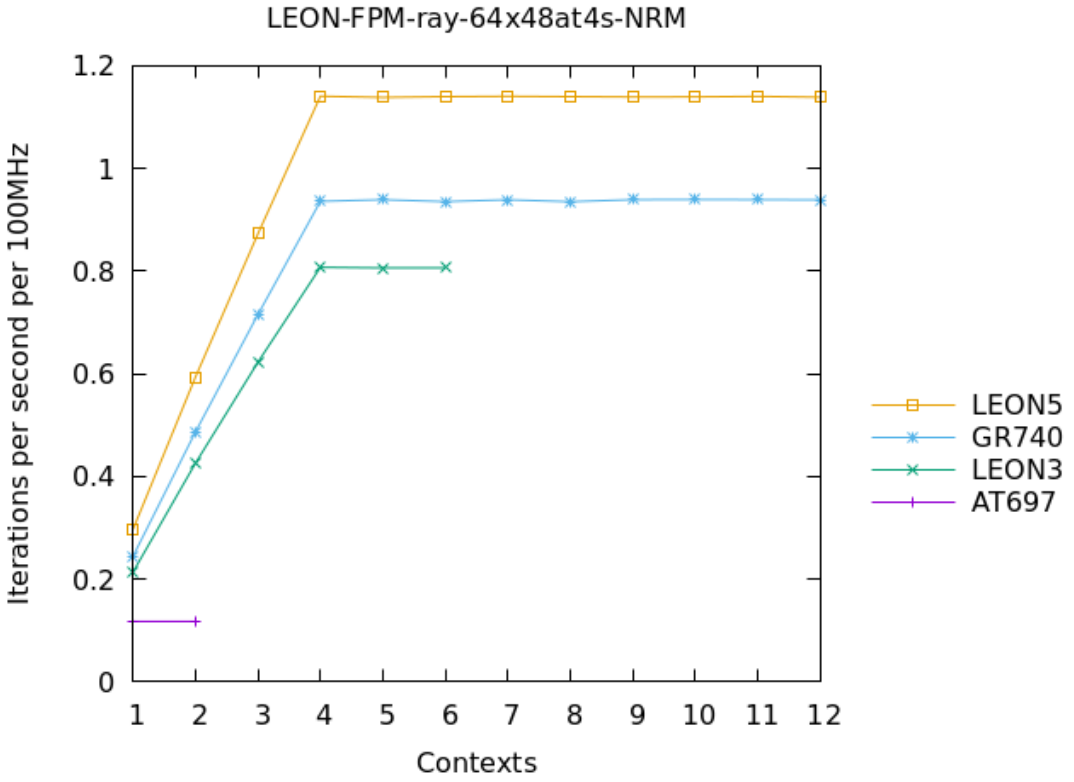

Figure 146: LEON - FPMark - ray-64x48at4s, iterations per second at 100MHz. Higher values denote better performance.

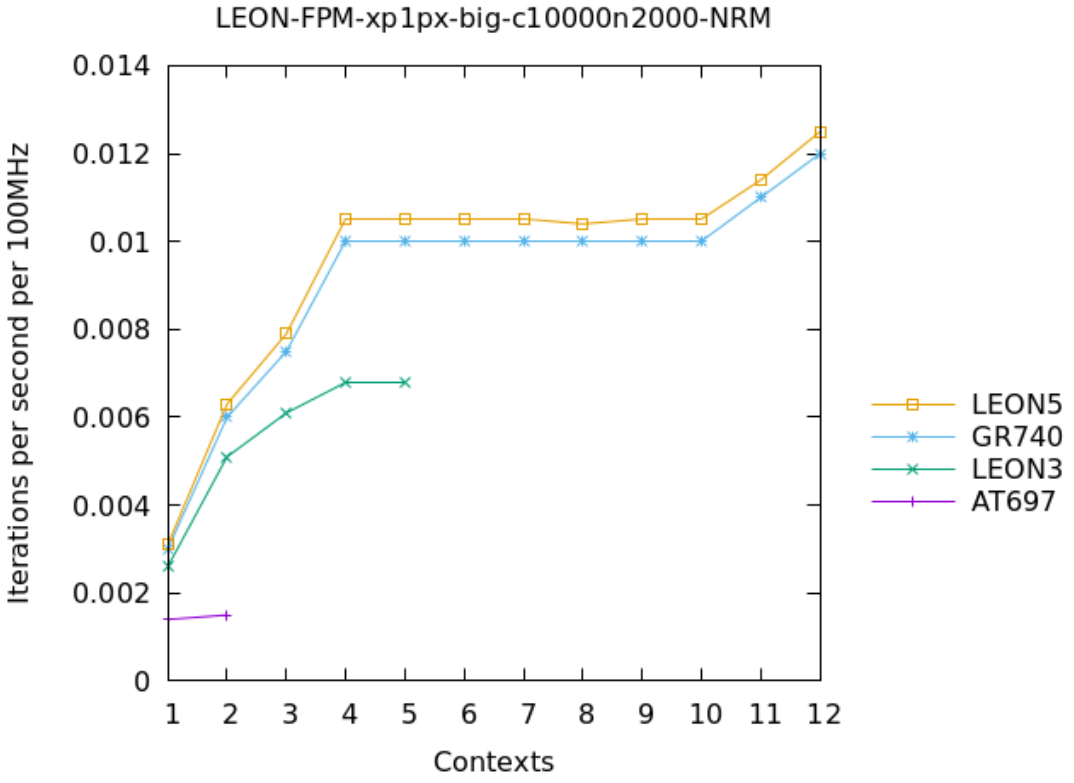

Figure 147: LEON - FPMark - xp1px-big-c10000n2000, iterations per second at 100MHz. Higher values denote better performance.

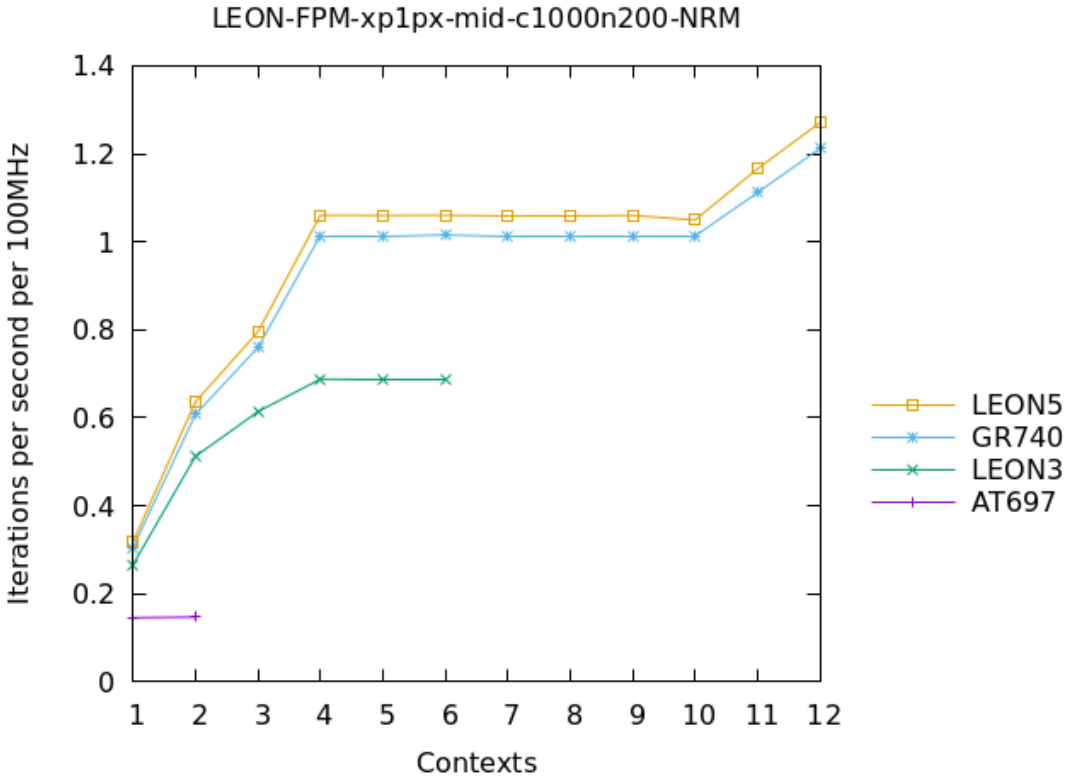

Figure 148: LEON - FPMark - xp1px-mid-c1000n200, iterations per second at 100MHz. Higher values denote better performance.

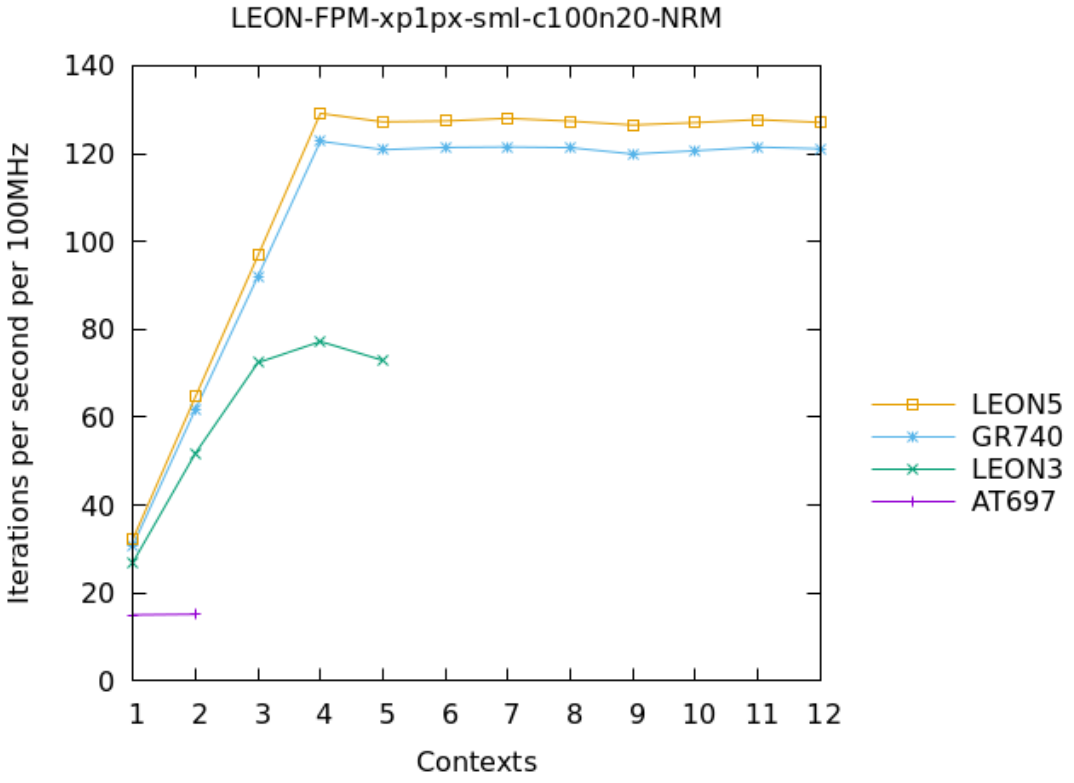

Figure 149: LEON - FPMark - xp1px-sml-c100n20, iterations per second at 100MHz. Higher values denote better performance.

10 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.

10.1 PWLS benchmarks

Figures 150 to 152 demonstrate the floating-point performance in the NOEL-V and LEON-based systems.

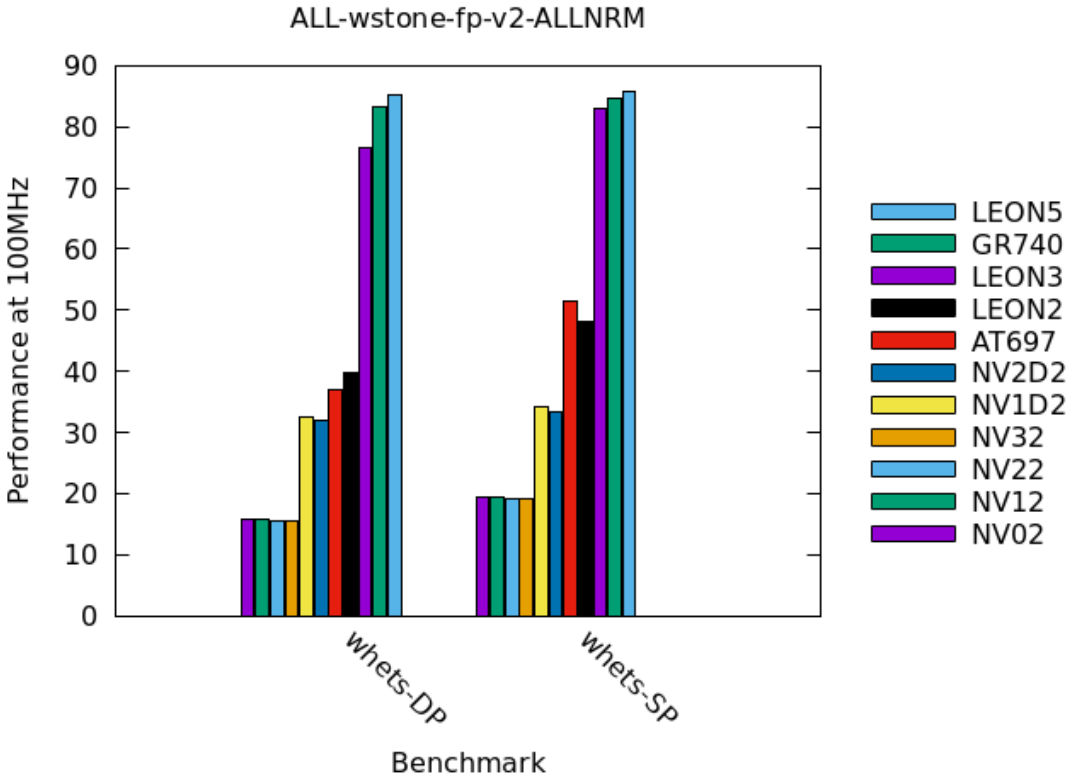

Figure 150: ALL - Whetstone performance for ALL targets, MWIPS at 100MHz. Higher values denote better performance.

Figures 150 and 151 clearly show an inferior floating-point performance of all NOEL-V configurations compared to any of the LEON-based systems, including NOEL-V configurations that use daiFPUrv, even when compared to LEON2 with daiFPU. The reason is a poor implementation of the floating-point interface in the current NOEL-V versions (GRLIB 2021.2).

Fig. 152 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 *NV02* and *NV12*

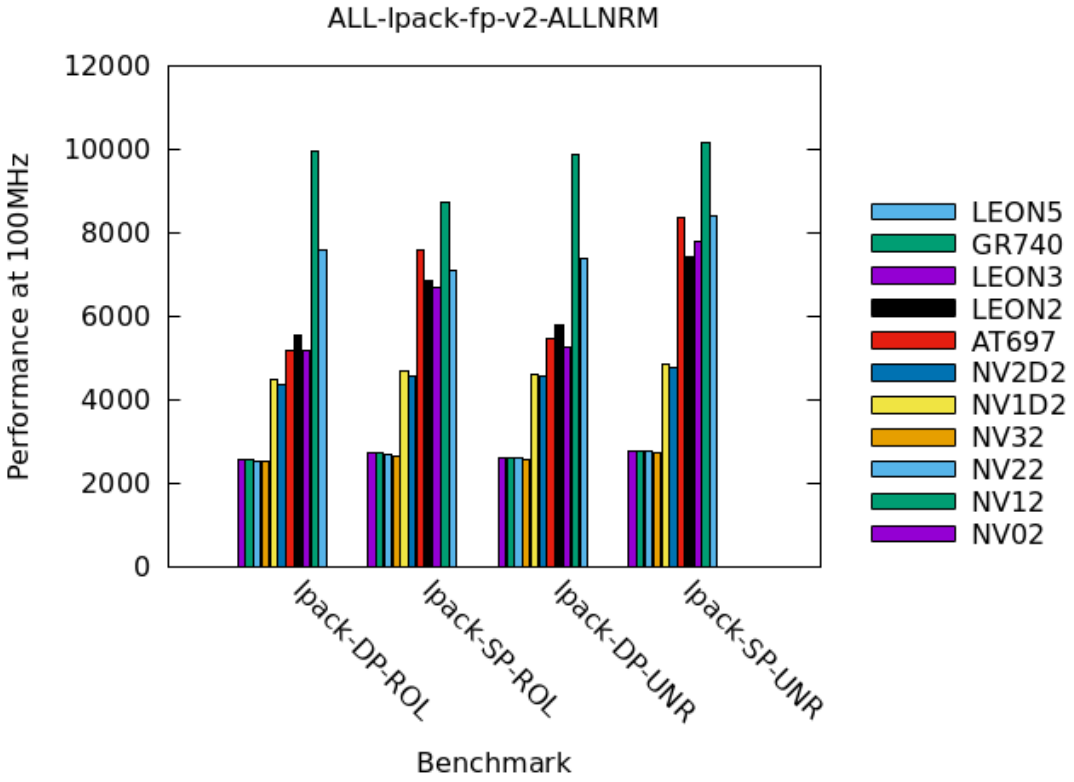

Figure 151: ALL - Linpack performance for ALL targets, Kflops at 100MHz. Higher values denote better performance.

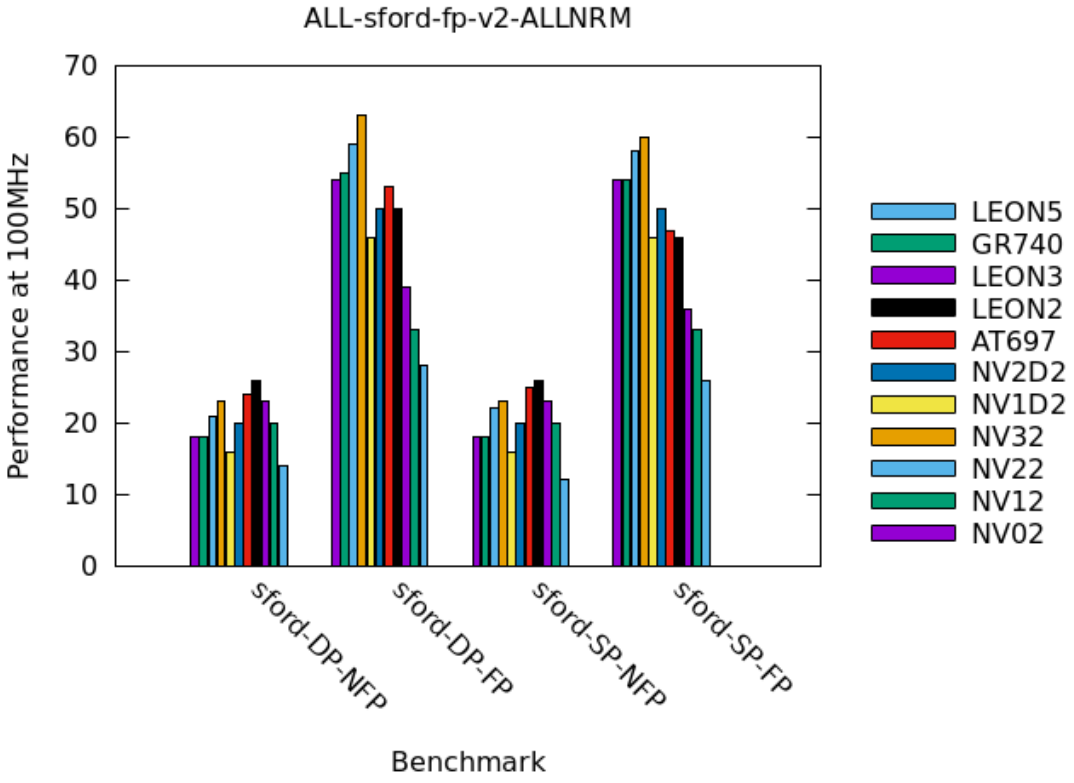

Figure 152: ALL - Stanford performance for ALL targets, milliseconds at 100MHz. Lower values denote better performance.

achieve a better performance than *LEON2*, *LEON3* and *GR740*, while *LEON5* scores better than *NV02* and *NV12*. This is attributed to wider instruction and data buses in NOEL-V (64bits), up to 4x the memory 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.

10.2 CoreMark

Figures 153 and 154 show absolute performance scaling in the NOEL-V and LEON-based systems for both the floating-point and integer version of the *CoreMark* benchmark.

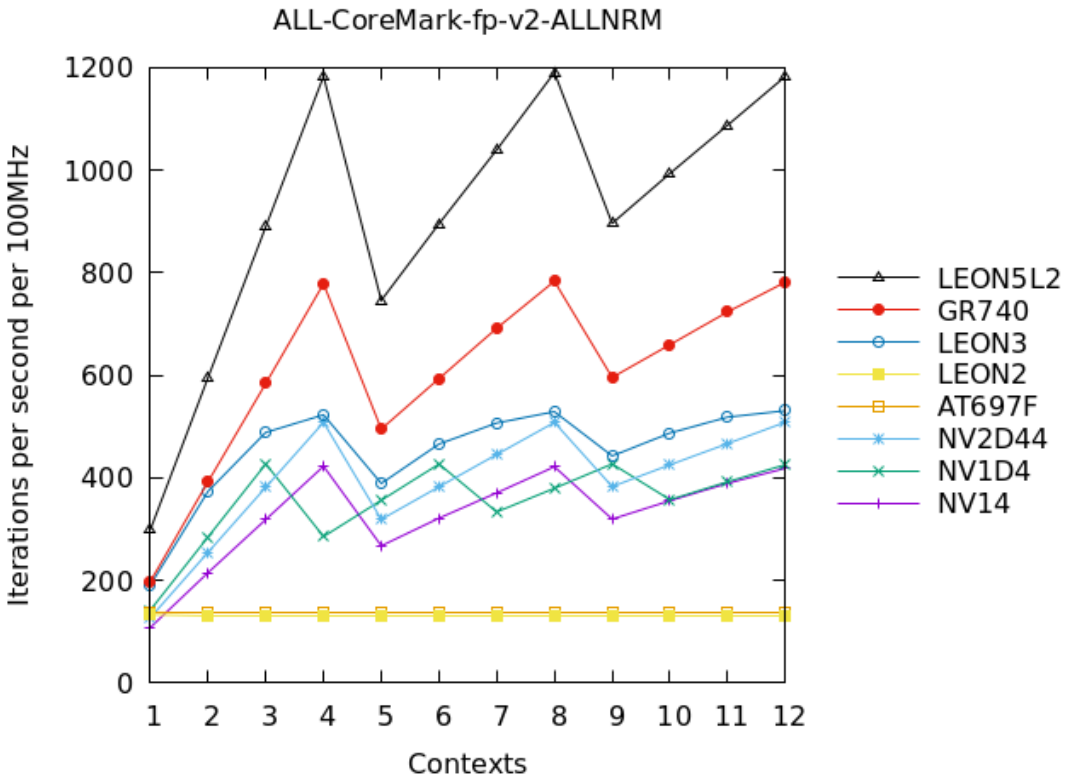

Figure 153: ALL - CoreMark - floating-point performance for all targets, CoreMark iterations at 100MHz. Higher values denote better performance.

Fig. 153 shows that

- LEON5 achieves 2.8x the floating-point performance of the NOEL-V *NV02*, and 2.33x the performance of *NV2D44*.
- GR740 achieves 2x the floating-point performance of the NOEL-V *NV02*, and 1.54x the performance of *NV2D44*.
- LEON3 with *GRFPU* about 1.25x the floating-point performance of NOEL-V *NV02*, and 1.08x the performance of *NV2D44*.

Given that in the sampled configurations NOEL-V uses a serial daiFPUrv, while LEON3 uses a pipelined GRFPU, the reason for fact that NOEL-V configuration *NV2D44* nearly matches the performance of LEON3 is seen in the L2Cache that is present in *NV2D44* and missing in LEON3.

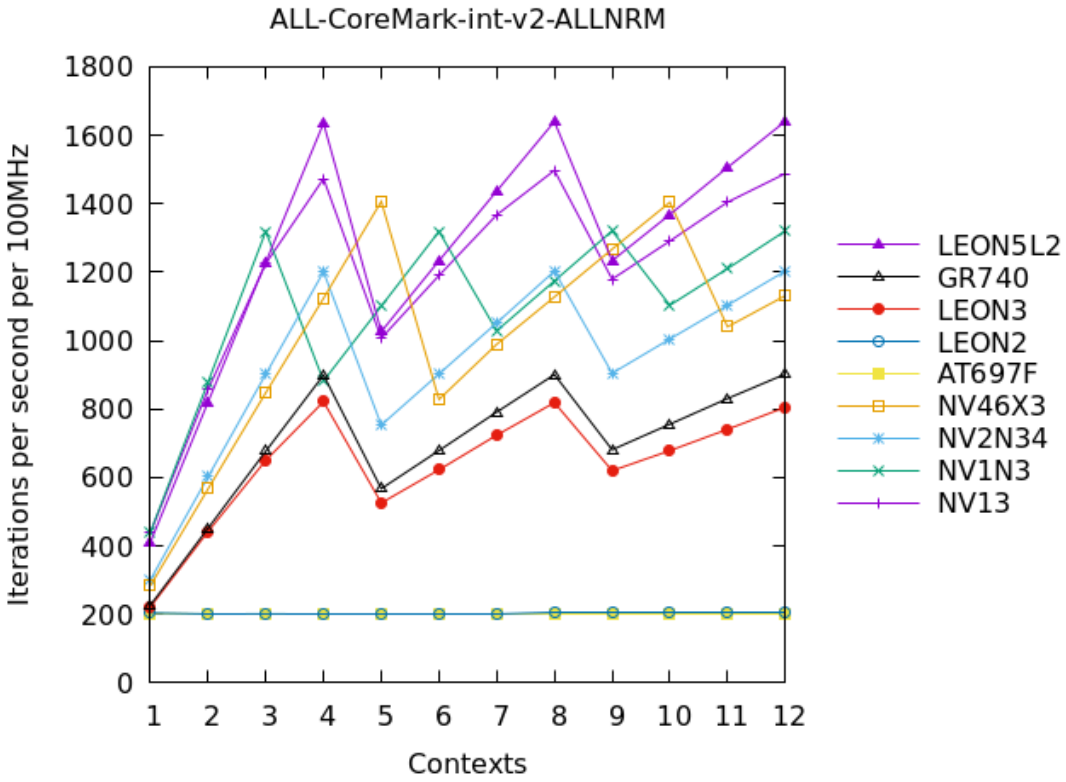

Figure 154: ALL - CoreMark - integer performance for all targets, CoreMark iterations at 100MHz. Higher values denote better performance.

Fig. 154 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 *NV01* and *NV11* are slightly worse than *LEON5L2* and slightly better than *LEON5*. The performance of NOEL-V *NV01* 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 *NV61* (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 8.2 and 9.2 respectively.

10.3 CoreMark-Pro

10.3.1 Performance plots

Figures 155 to 163 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 floating-point performance superior to any of the NOEL-V configurations. For certain integer workloads the configuration *NV2D44* shows a performance comparable to *GR740* - *cjpeg*, *core*, *zip-test*. For *radix2 GR740* exhibits suboptimal performance, probably due to saturation of the memory system.

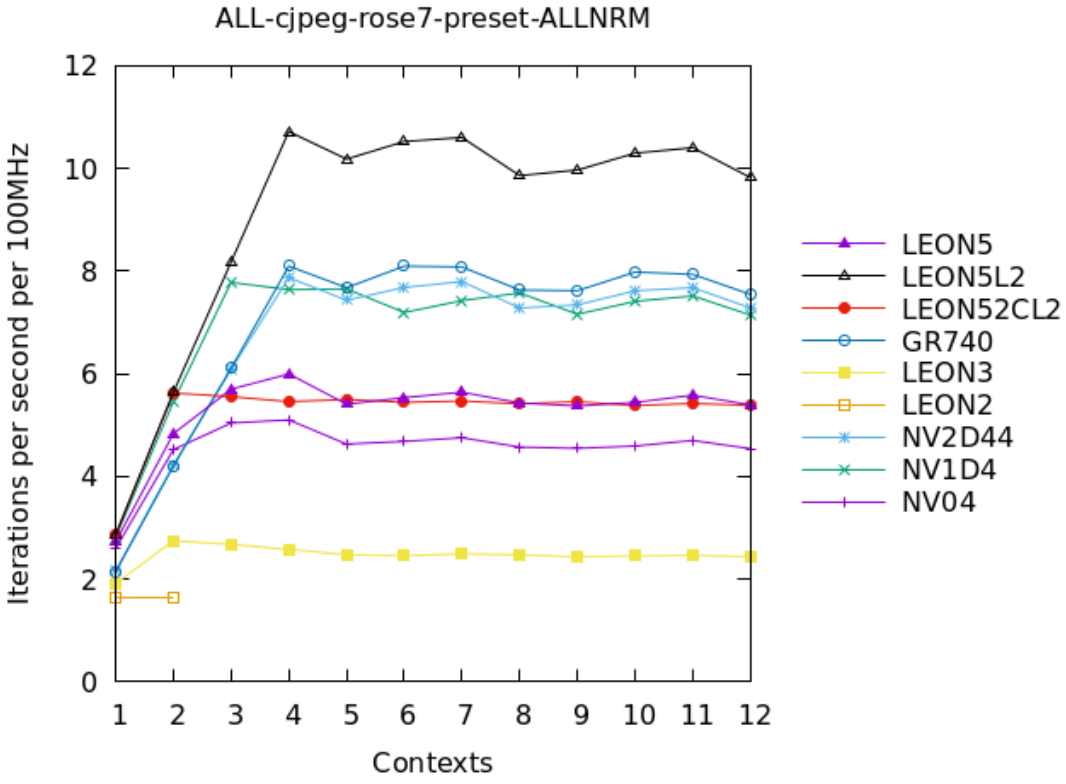

Figure 155: ALL - CoreMark-Pro - cjpeg, iterations per second at 100MHz. Higher values denote better performance.

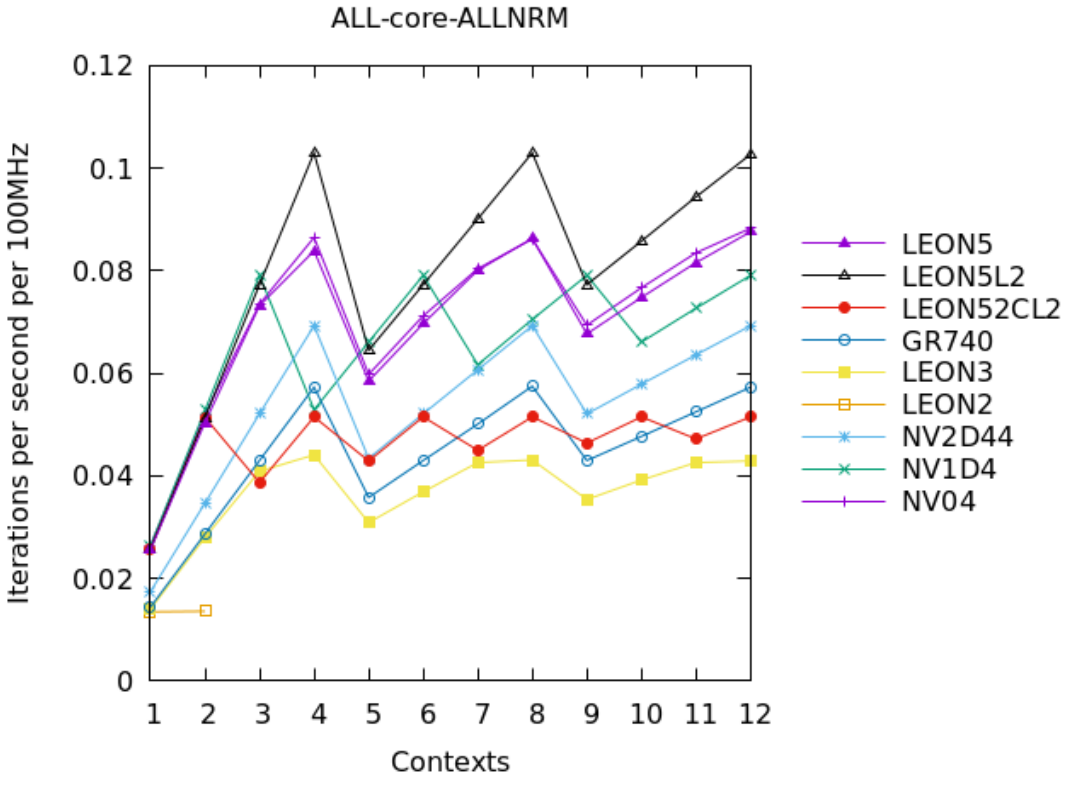

Figure 156: ALL - CoreMark-Pro - core, iterations per second at 100MHz. Higher values denote better performance.

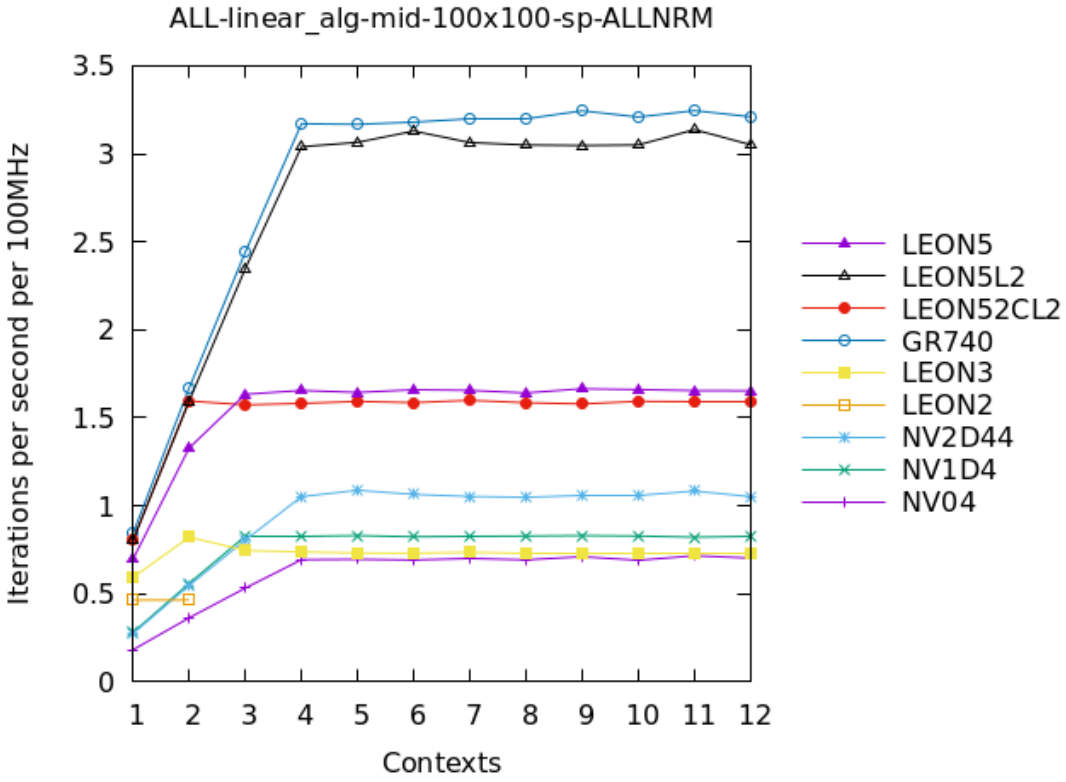

Figure 157: ALL - CoreMark-Pro - linear, iterations per second at 100MHz. Higher values denote better performance.

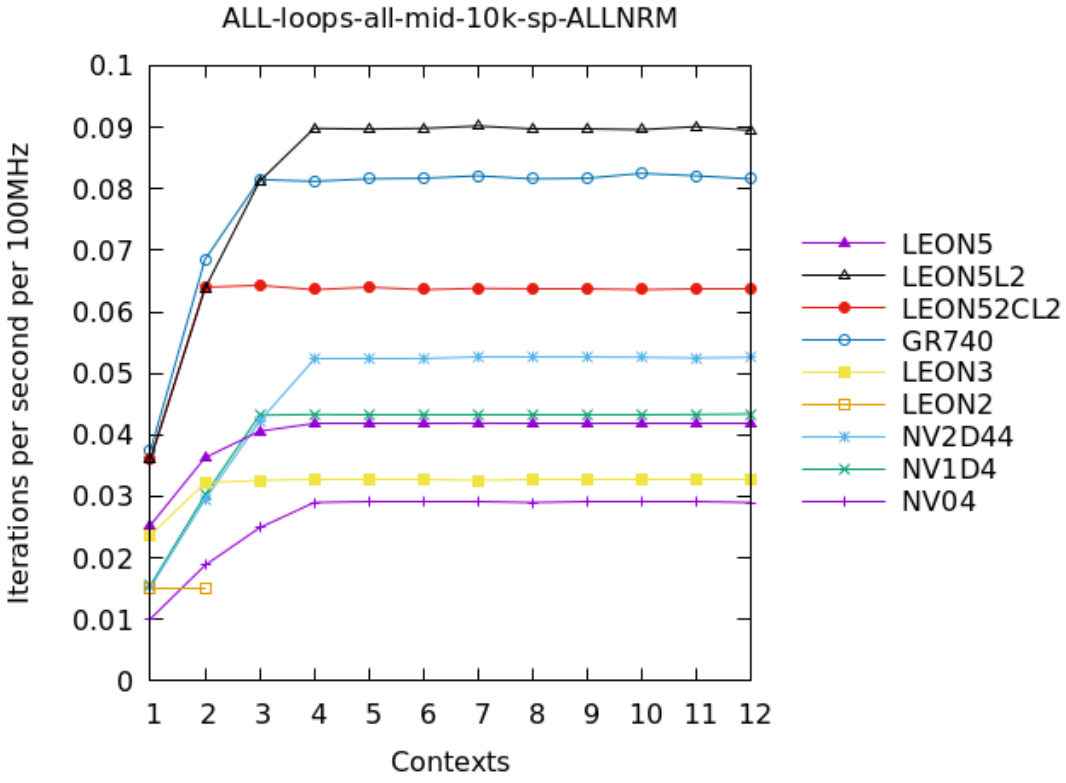

Figure 158: ALL - CoreMark-Pro - loops, iterations per second at 100MHz. Higher values denote better performance.

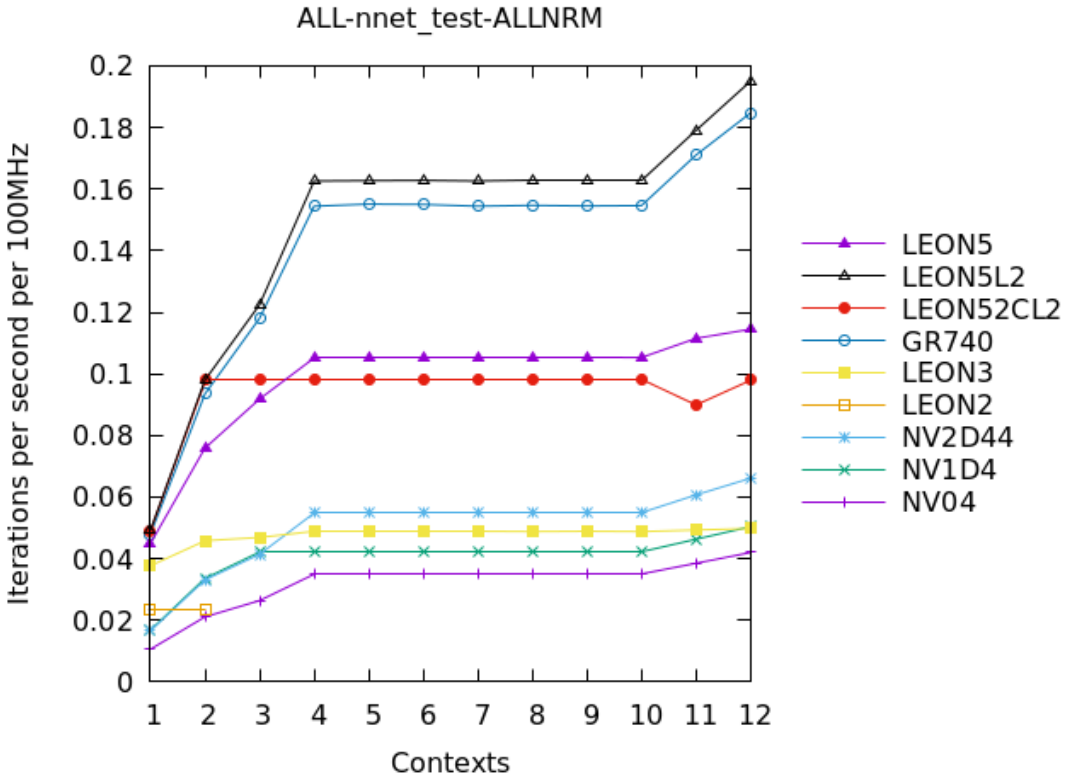

Figure 159: ALL - CoreMark-Pro - nnet, iterations per second at 100MHz. Higher values denote better performance.

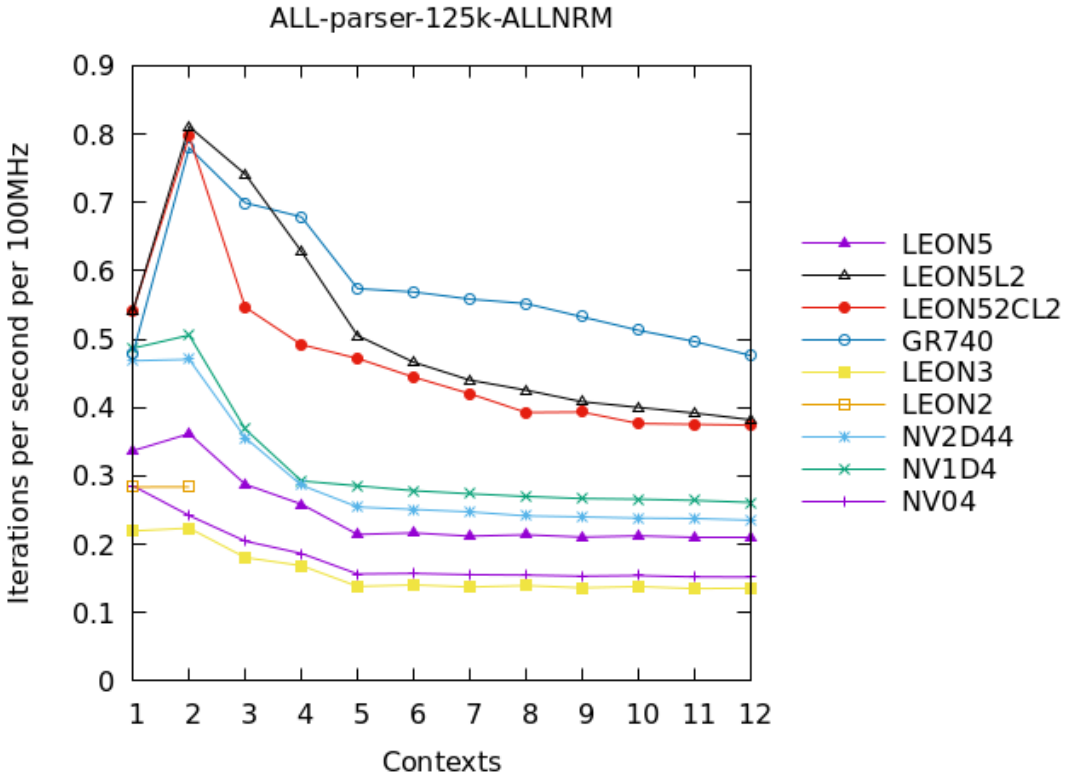

Figure 160: ALL - CoreMark-Pro - parser, iterations per second at 100MHz. Higher values denote better performance.

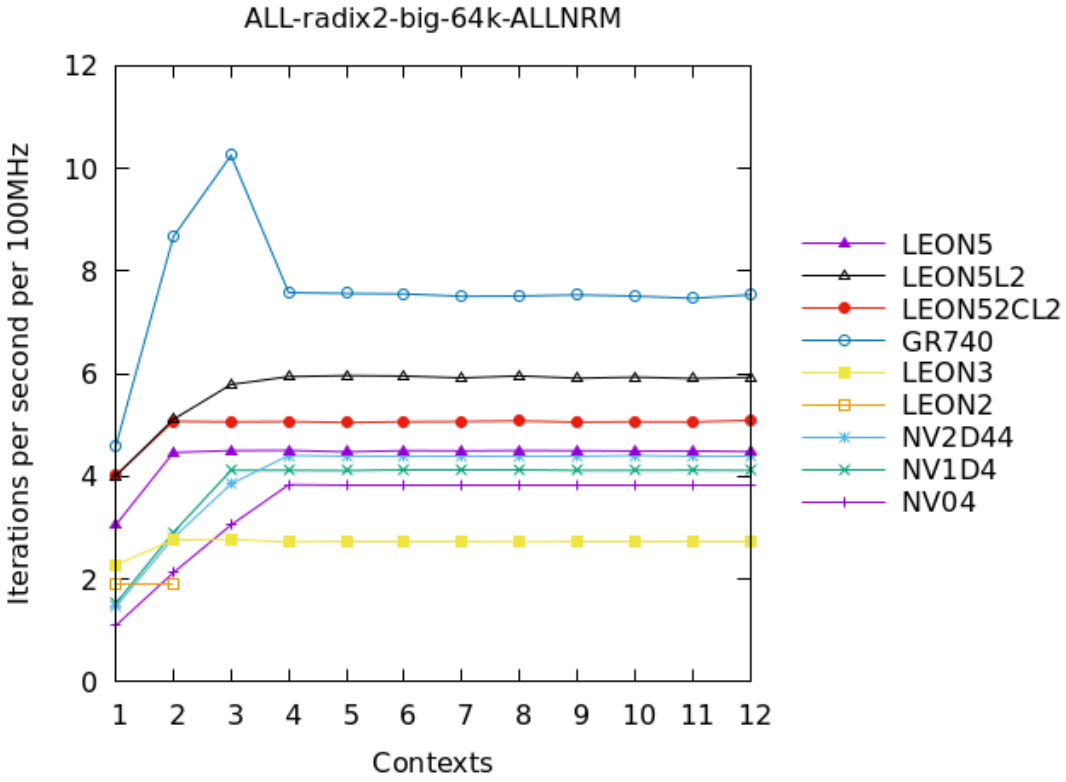

Figure 161: ALL - CoreMark-Pro - radix2, iterations per second at 100MHz. Higher values denote better performance.

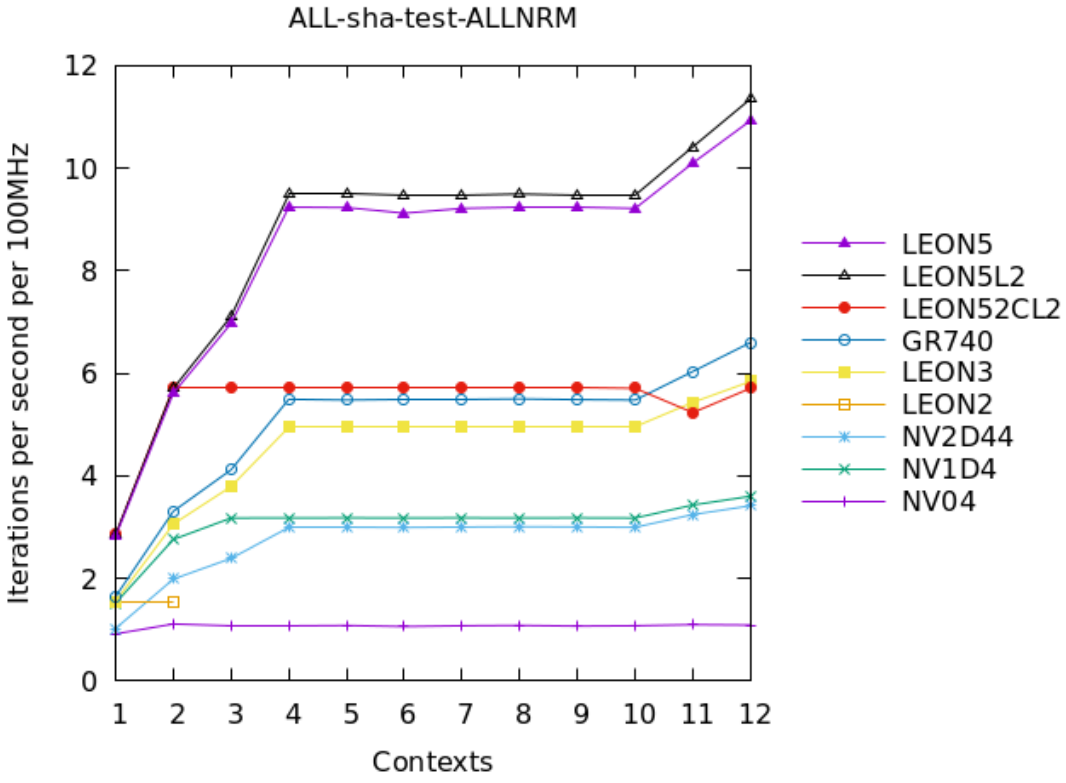

Figure 162: ALL - CoreMark-Pro - sha, iterations per second at 100MHz. Higher values denote better performance.

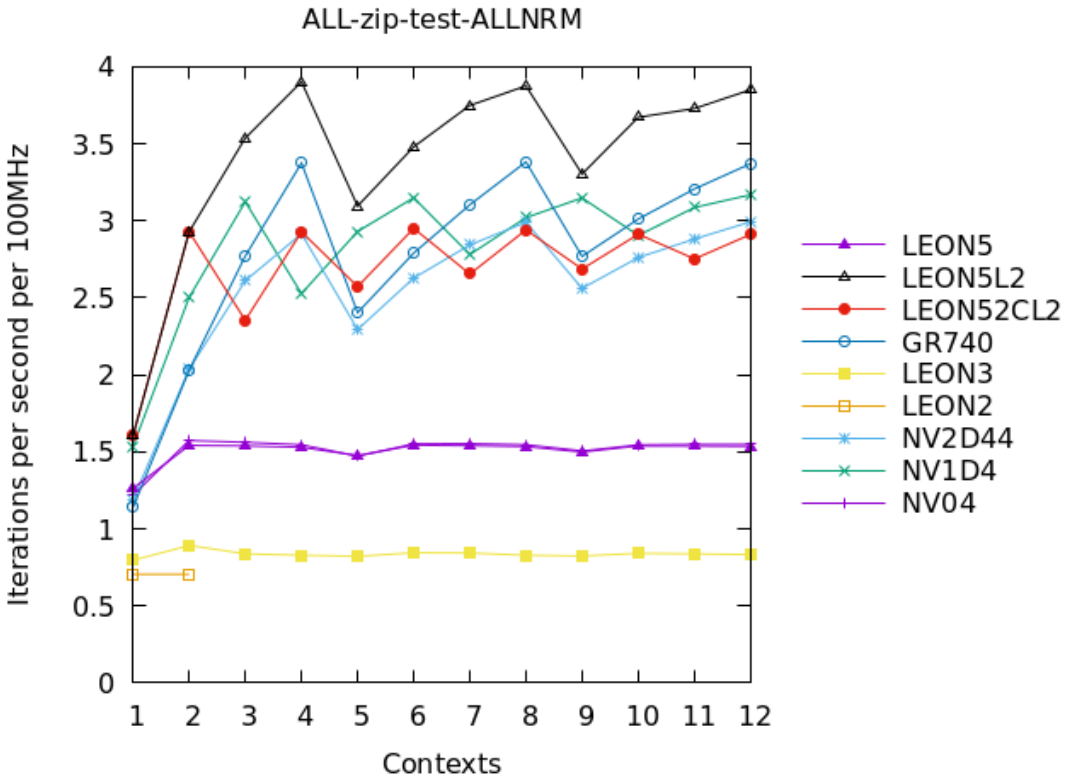

Figure 163: ALL - CoreMark-Pro - zip, iterations per second at 100MHz. Higher values denote better performance.

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

NOEL-V with *daiFPUrv* and L2Cache achieves in general better performance than *LEON3*.

10.3.2 Scaling plots

Figures 164 to 172 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 either FPU exhibit a similar performance pattern in *linear* and *nnet* like GR740. In *core* the NOEL-V configurations *CFG0* and *CFG1* as well as *LEON3* get slightly memory-bound, indicated by the curved rising slope.

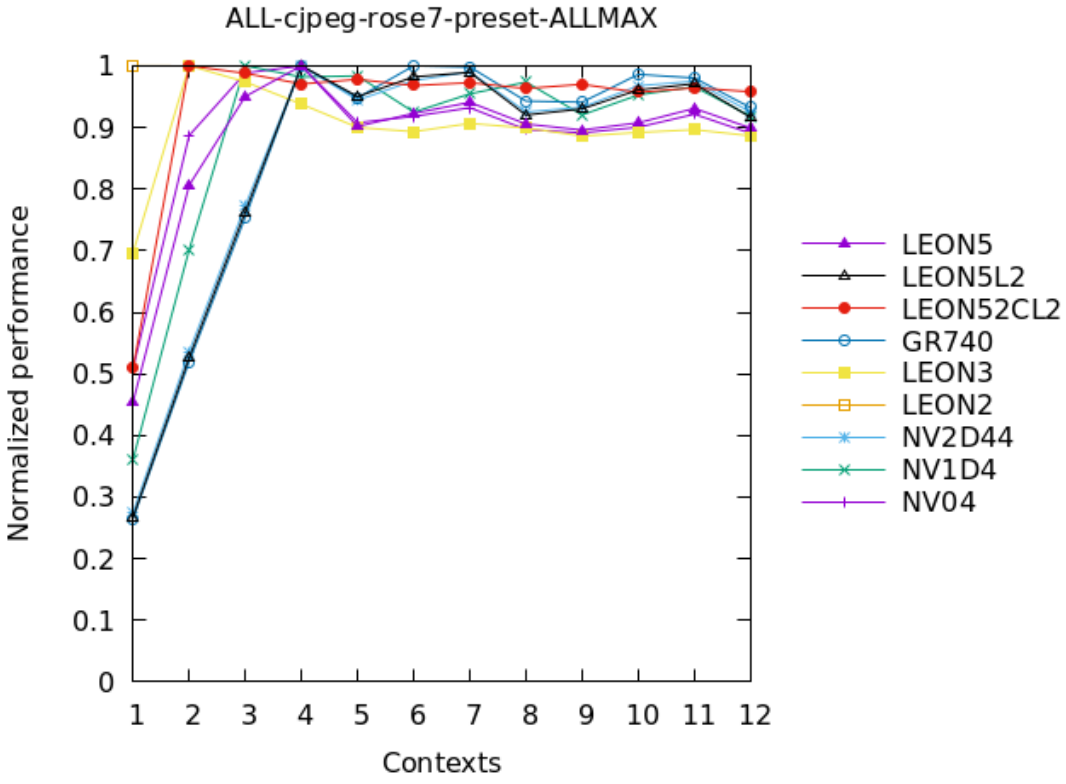

Figure 164: ALL - CoreMark-Pro - cjpeg, relative performance.

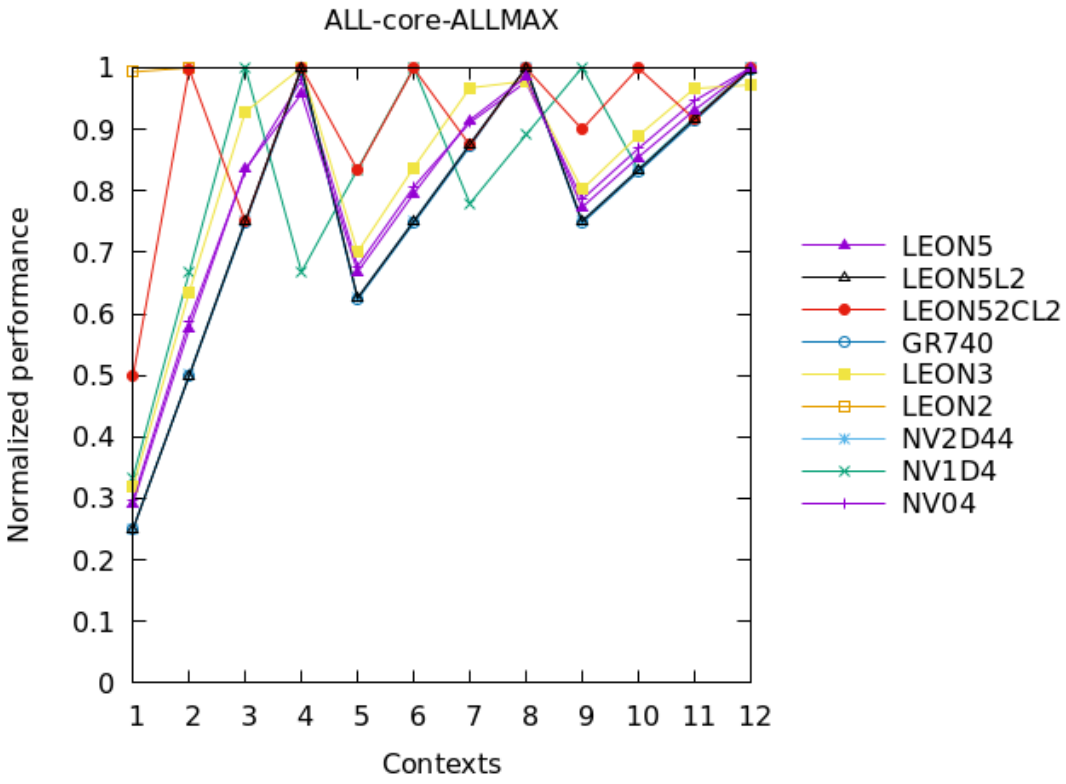

Figure 165: ALL - CoreMark-Pro - core, relative performance.

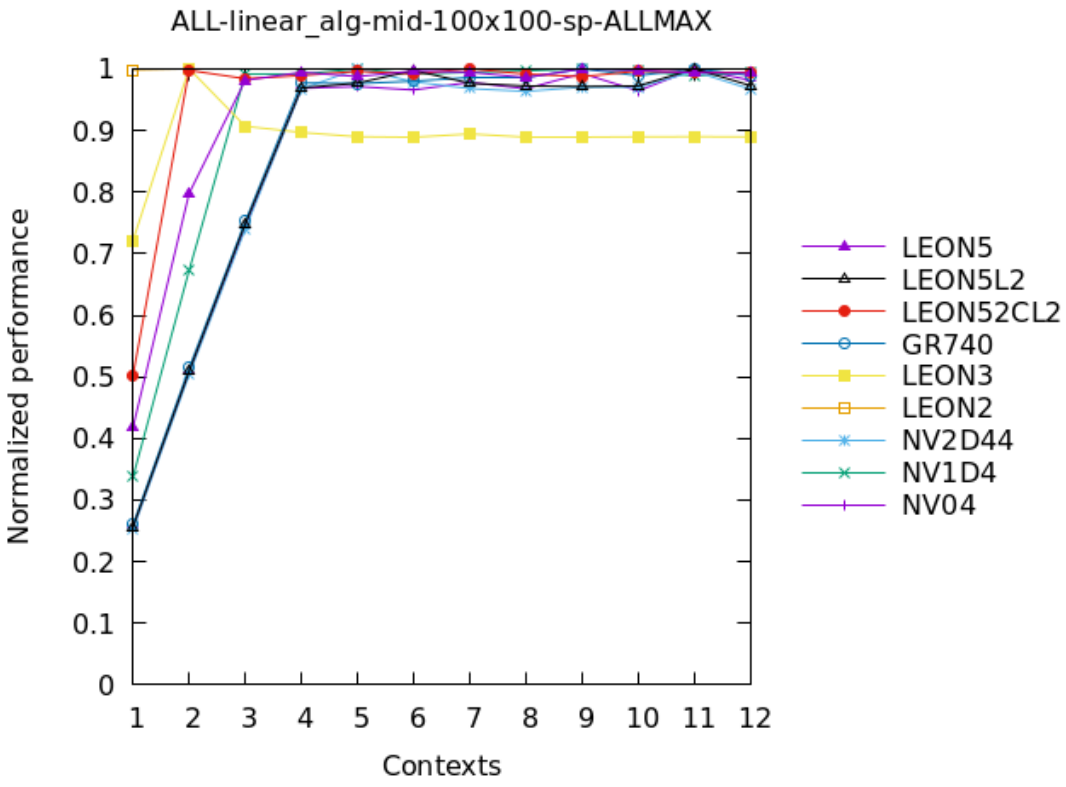

Figure 166: ALL - CoreMark-Pro - linear, relative performance.

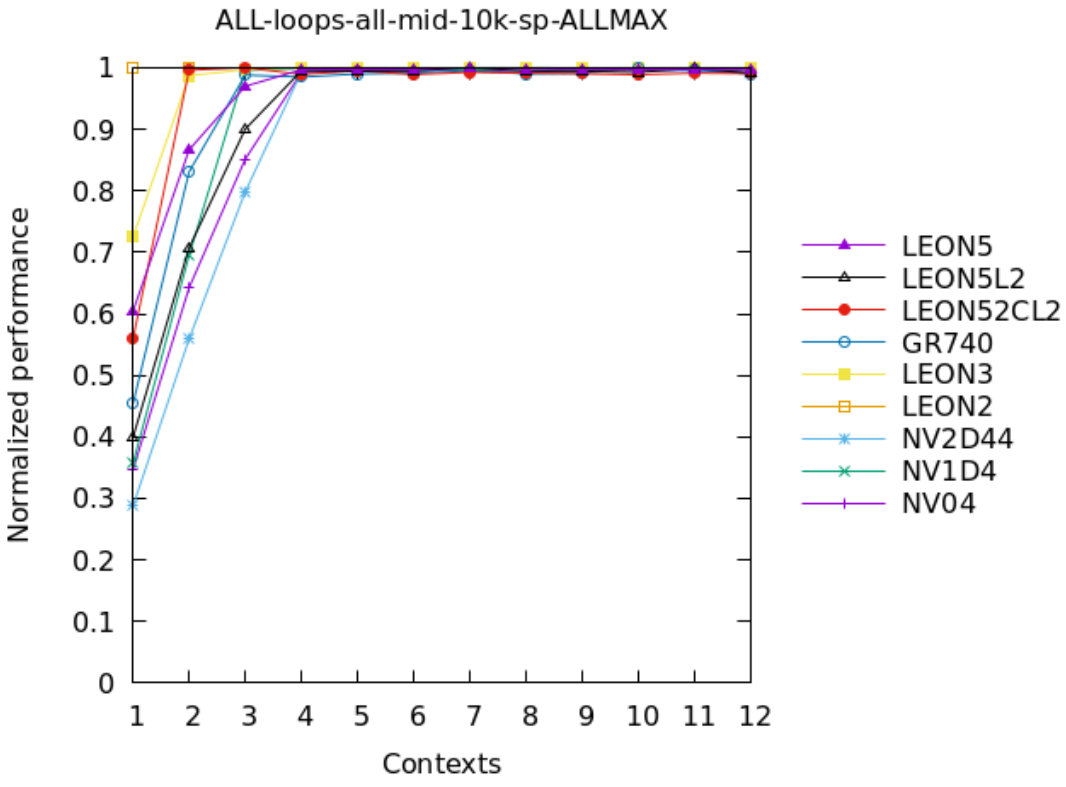

Figure 167: ALL - CoreMark-Pro - loops, relative performance.

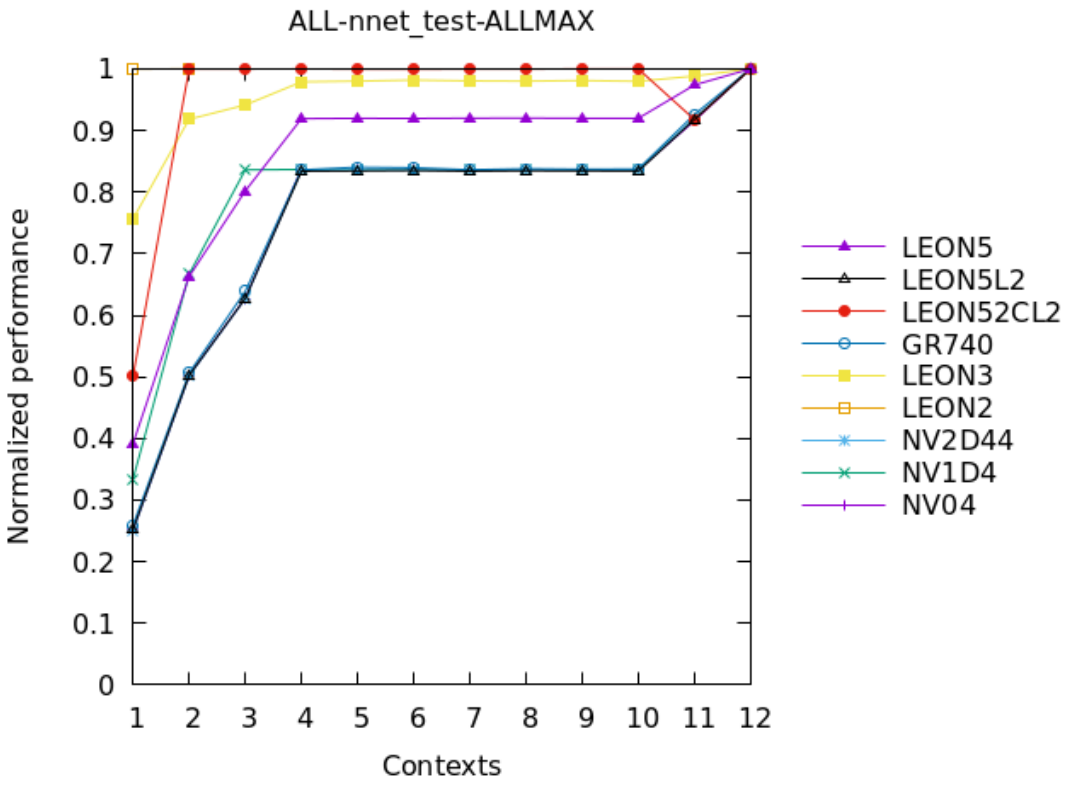

Figure 168: ALL - CoreMark-Pro - nnet, relative performance.

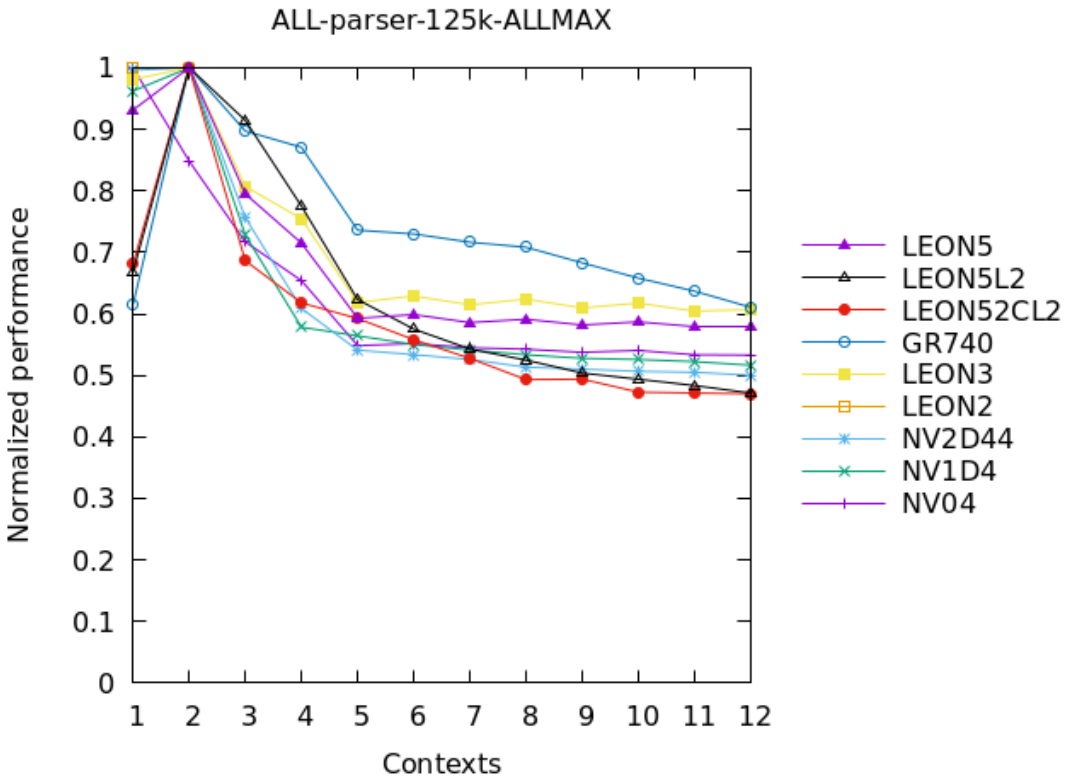

Figure 169: ALL - CoreMark-Pro - parser, relative performance.

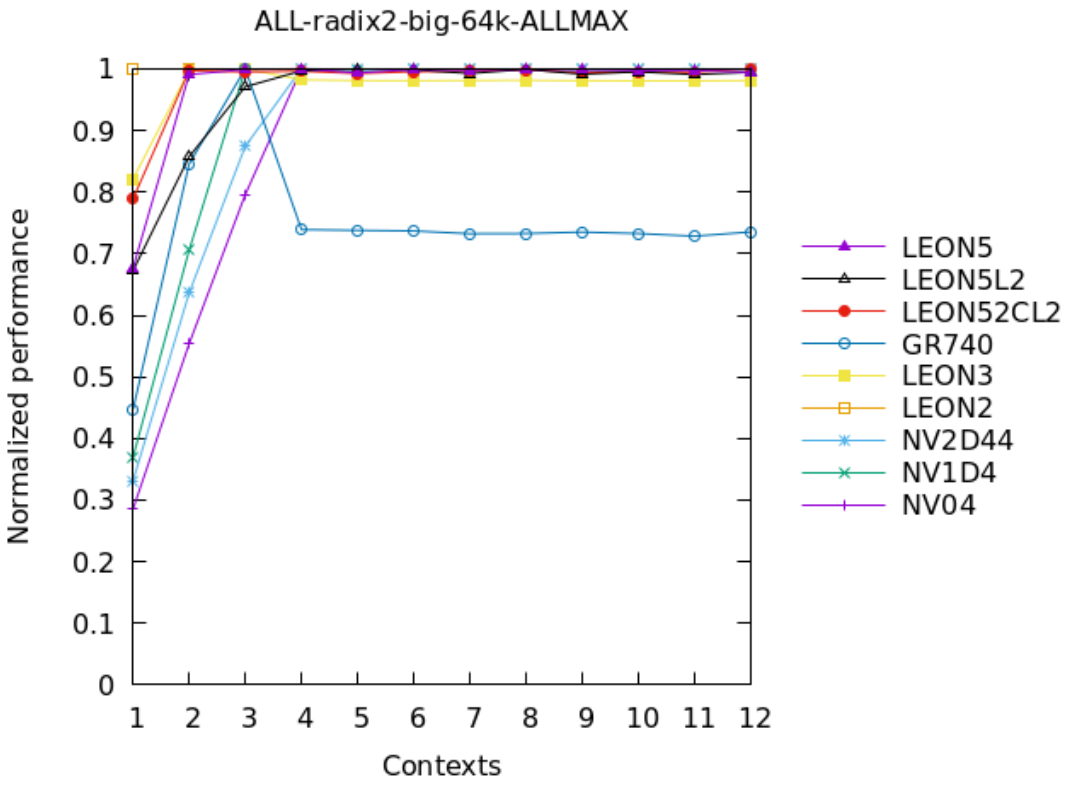

Figure 170: ALL - CoreMark-Pro - radix2, relative performance.

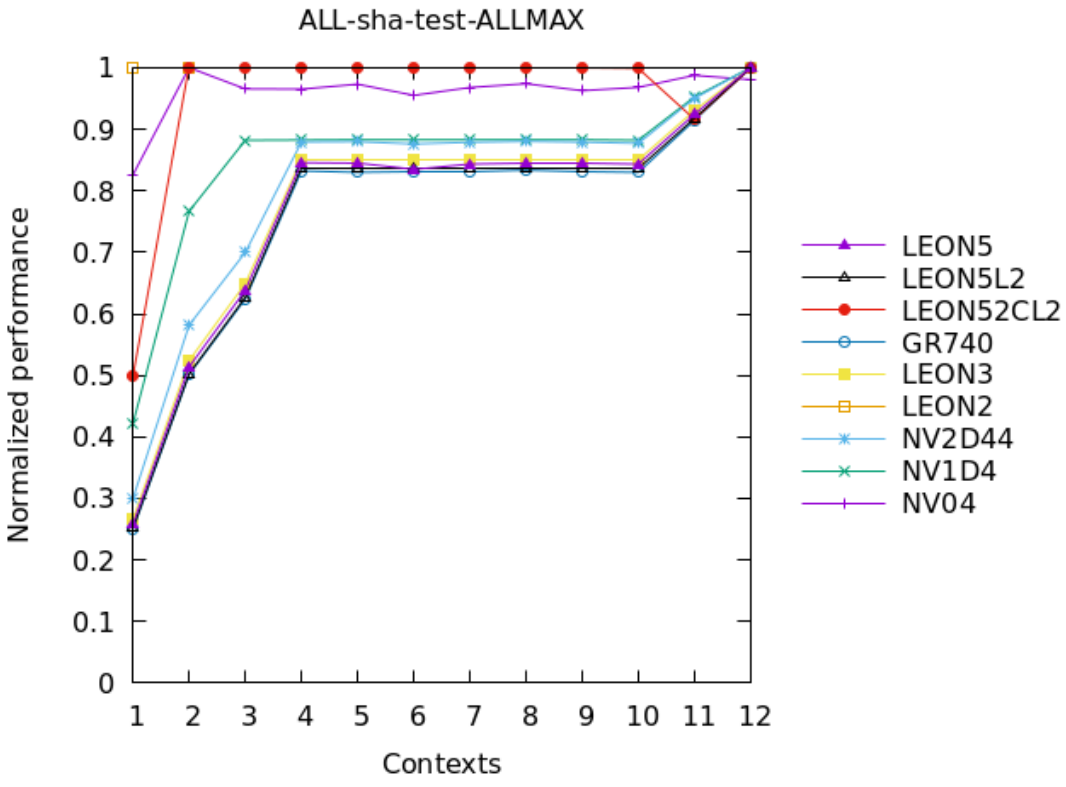

Figure 171: ALL - CoreMark-Pro - sha, relative performance.

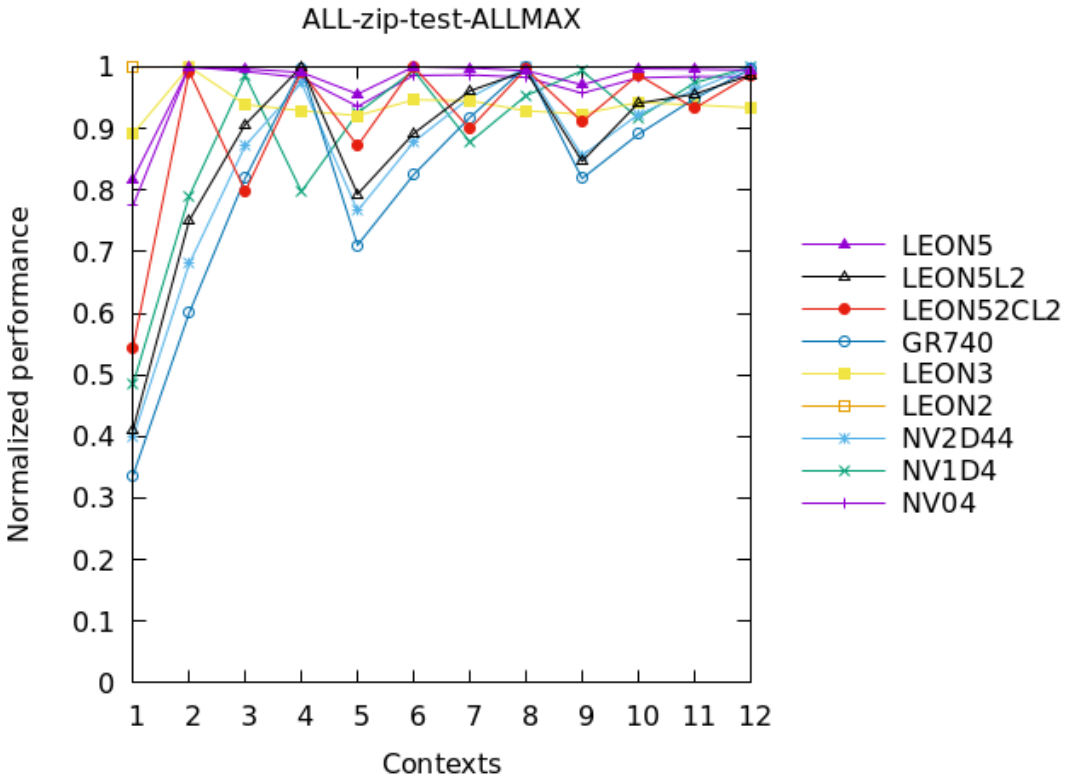

Figure 172: ALL - CoreMark-Pro - zip, relative performance.

10.4 FPMark

10.4.1 Performance plots

Figures 173 to 225 show absolute performance scaling in the NOEL-V and LEON-based systems.

In general, all the plots show a superior floating-point performance of LEON-based systems compared to all the RISC-V systems (including PolarFire SoC).

An interesting exceptionis the *inner-product* workload that exhibits an interesting behaviour shown in Figures 190 to 196; the behaviour differs with workload configurations and also processors. An interesting feature is the drop in performance for *GR740* beyond three contexts, also shown for *radix2* in Fig. 217.

A similar drop in performance is shown in Fig. 215 for *PFS*.

The plotted performance of the *ray-1024x768at24s* workload is affected by rounding errors due to very low number of iterations per second executed in all LEON and NOEL-V targets.

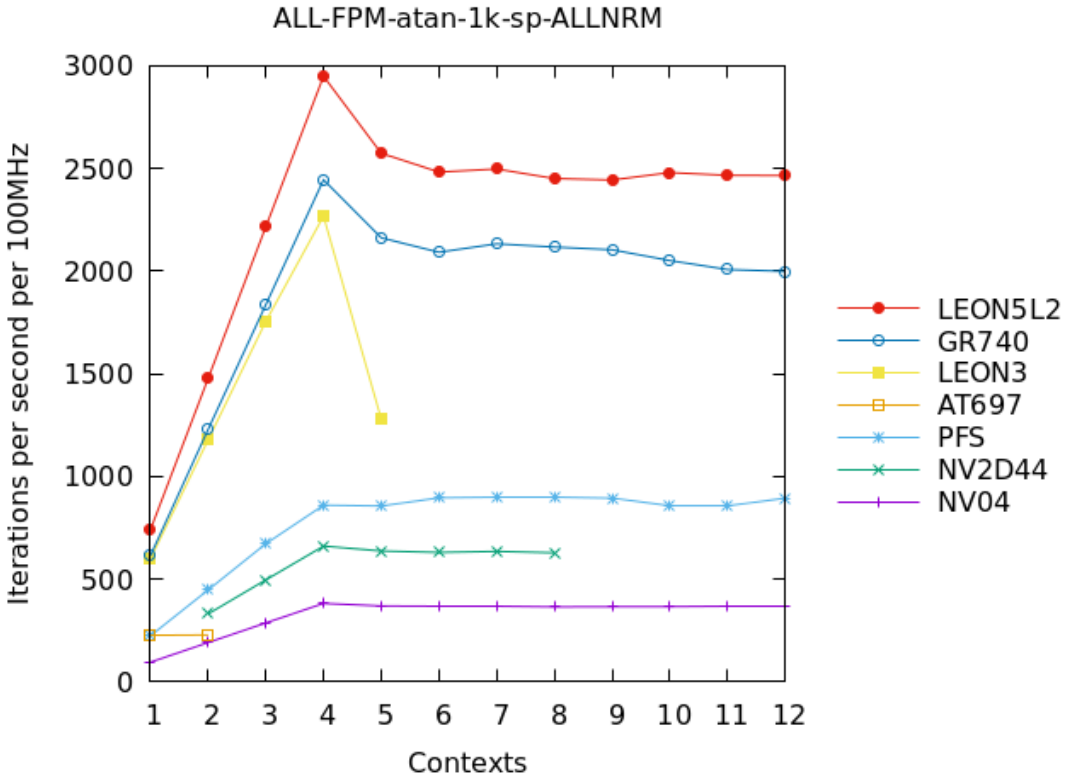

Figure 173: ALL - FPMark - atan-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

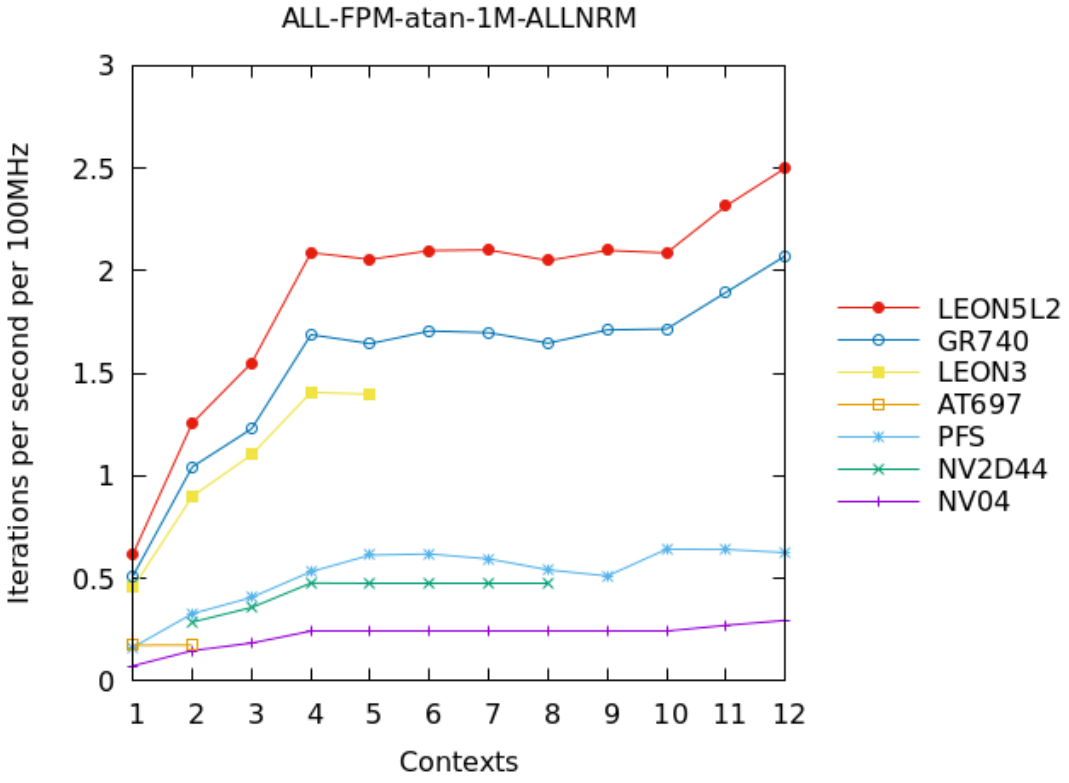

Figure 174: ALL - FPMark - atan-1M, iterations per second at 100MHz. Higher values denote better performance.

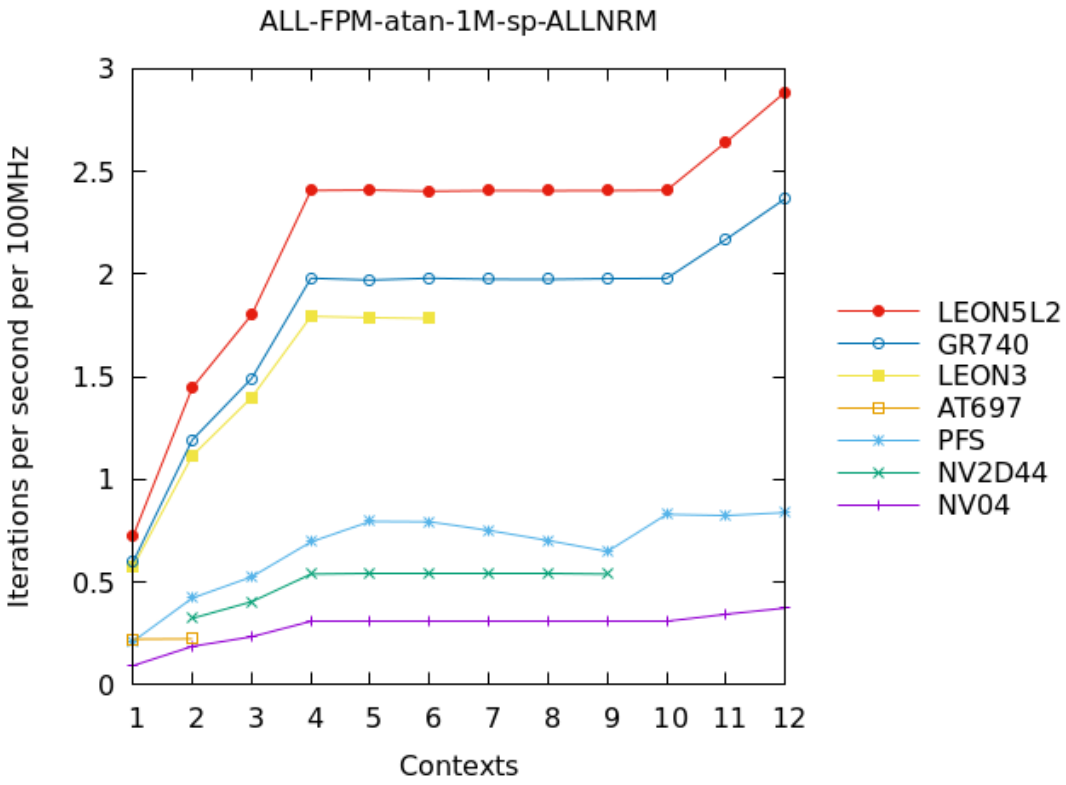

Figure 175: ALL - FPMark - atan-1M-sp, iterations per second at 100MHz. Higher values denote better performance.

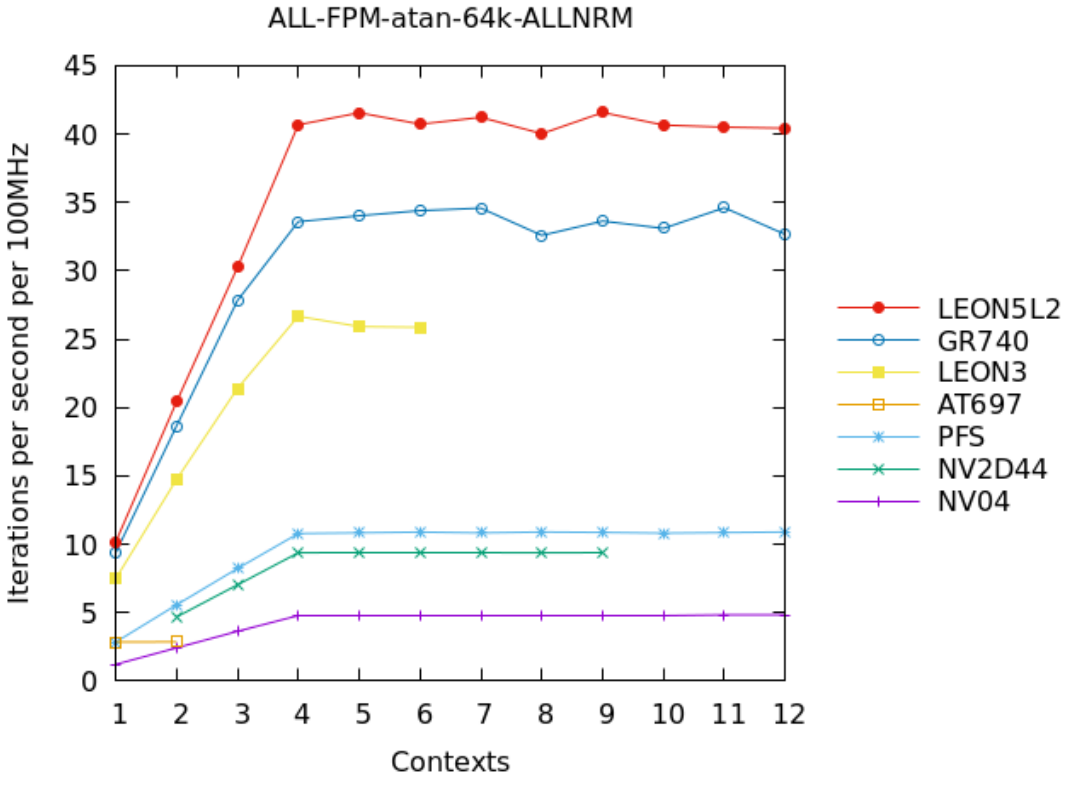

Figure 176: ALL - FPMark - atan-64k, iterations per second at 100MHz. Higher values denote better performance.

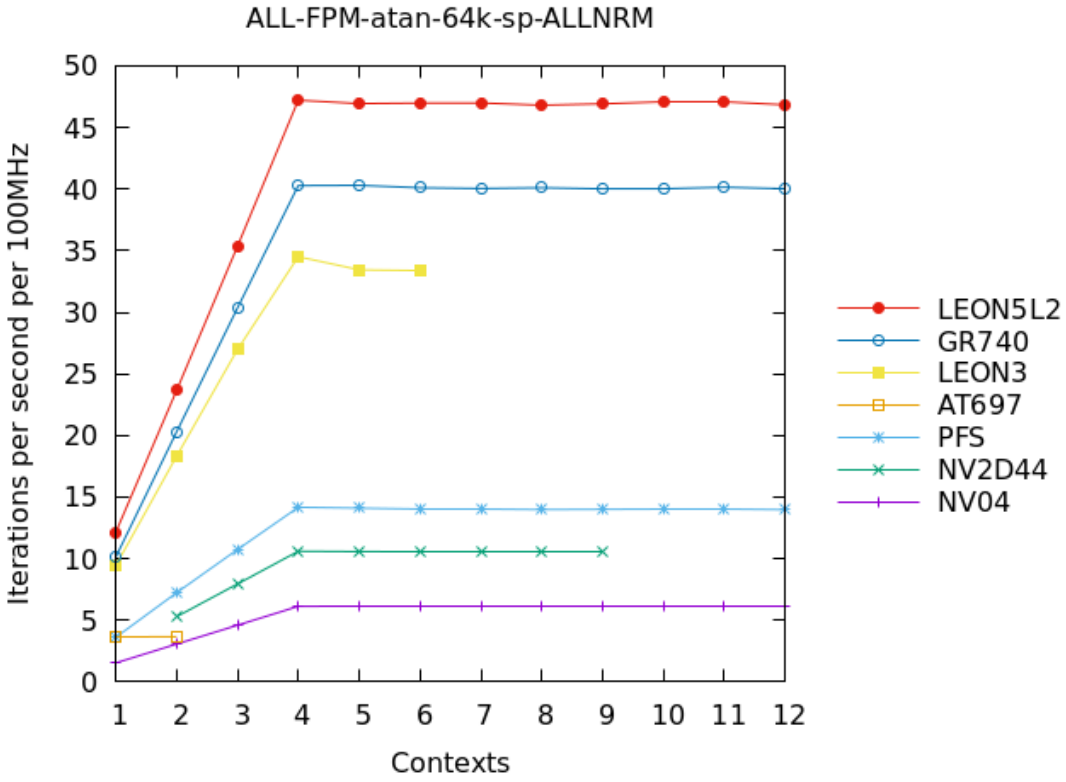

Figure 177: ALL - FPMark - atan-64k-sp, iterations per second at 100MHz. Higher values denote better performance.

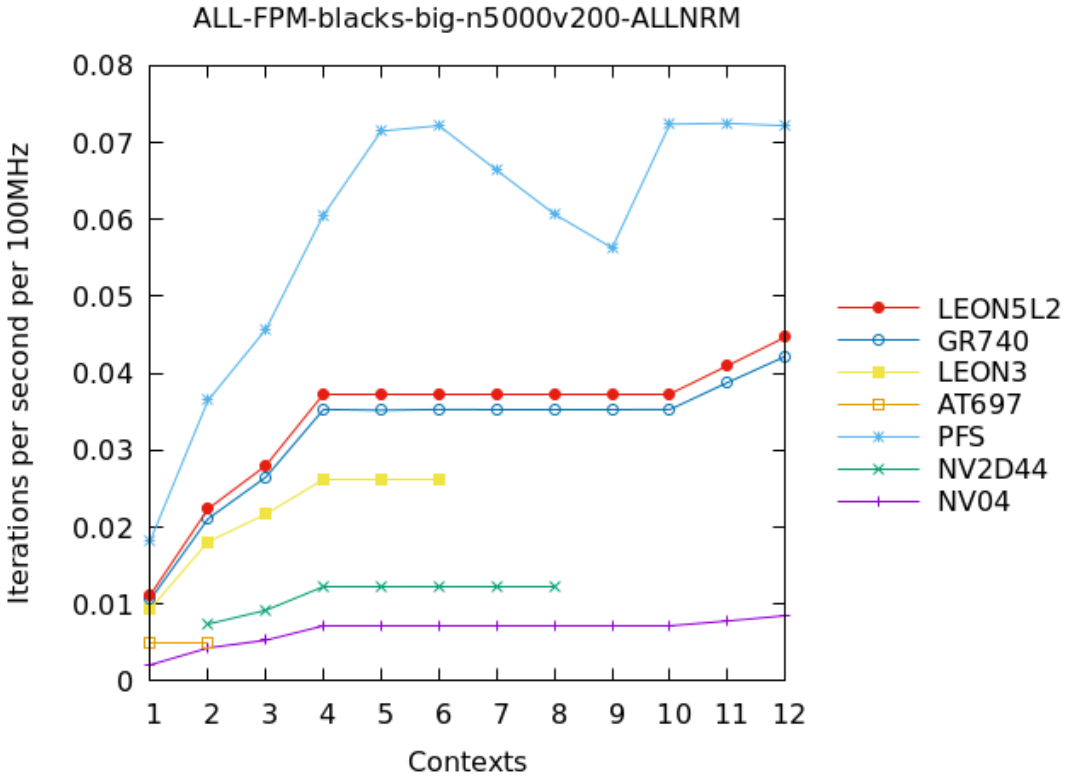

Figure 178: ALL - FPMark - blacks-big-n5000v200, iterations per second at 100MHz. Higher values denote better performance.

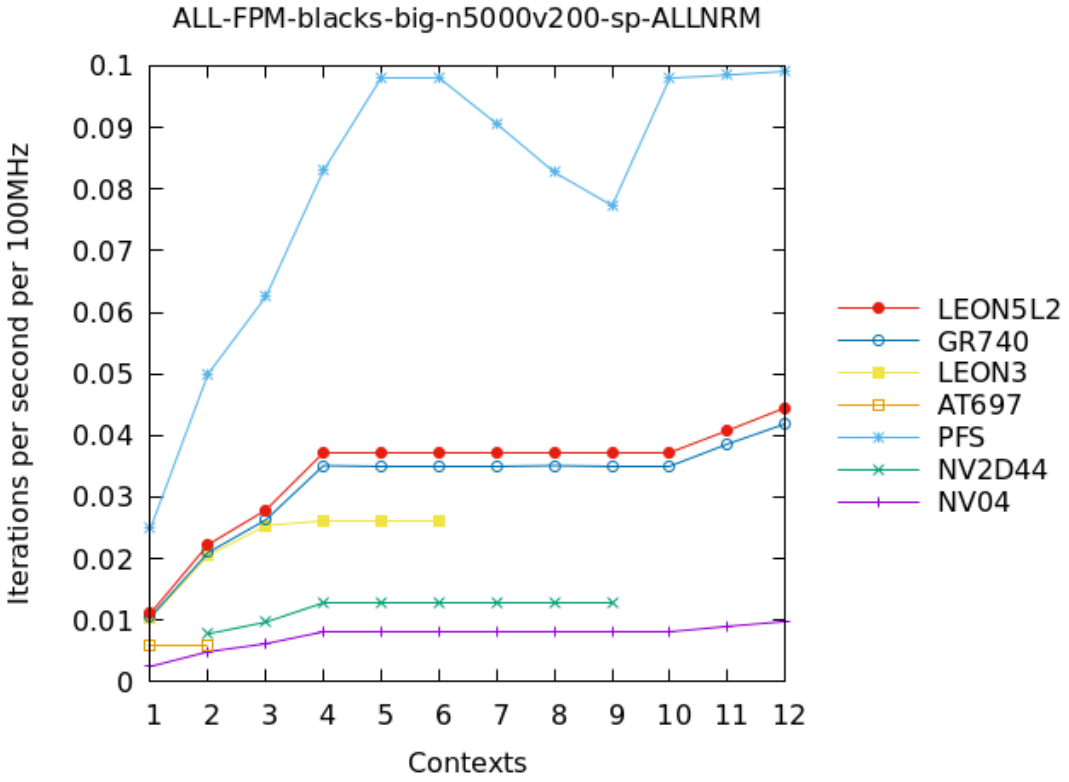

Figure 179: ALL - FPMark - blacks-big-n5000v200-sp, iterations per second at 100MHz. Higher values denote better performance.

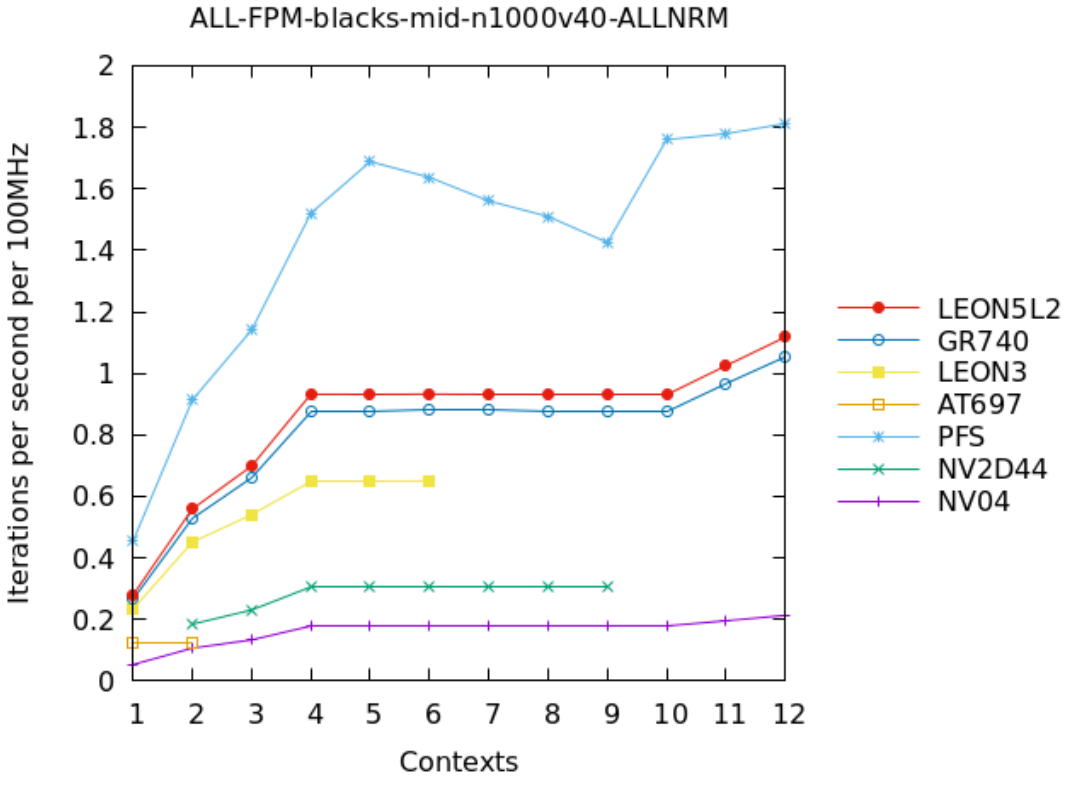

Figure 180: ALL - FPMark - blacks-mid-n1000v40, iterations per second at 100MHz. Higher values denote better performance.

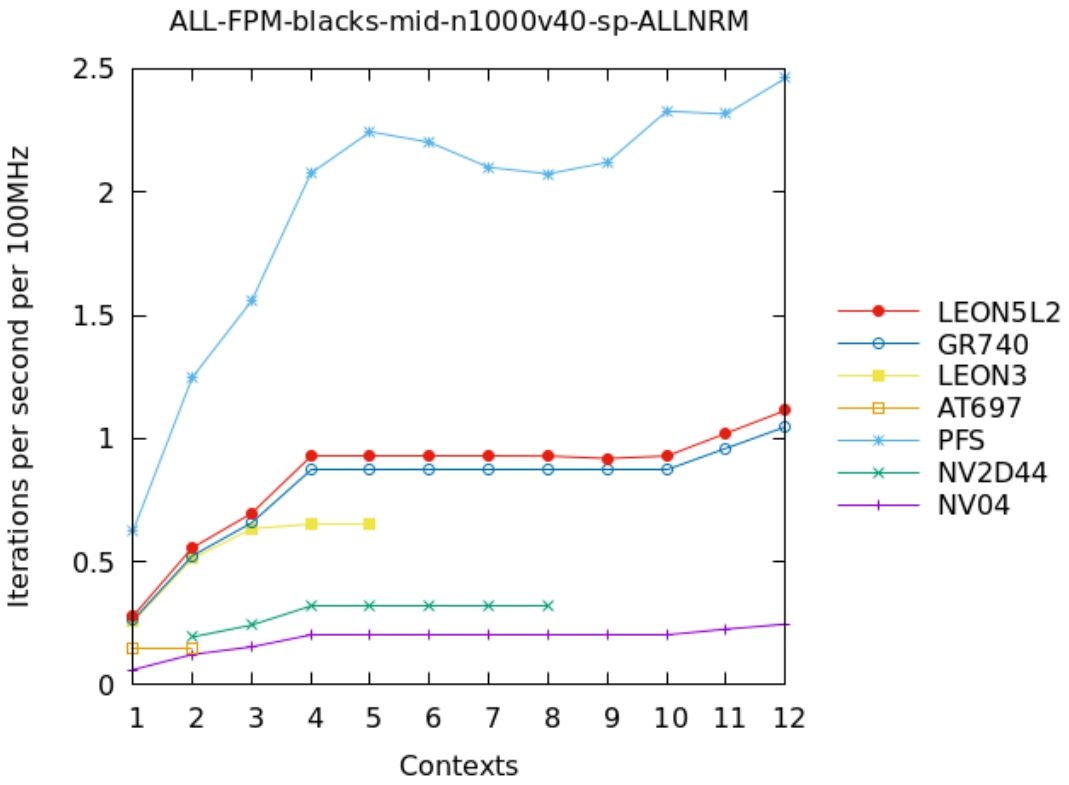

Figure 181: ALL - FPMark - blacks-mid-n1000v40-sp, iterations per second at 100MHz. Higher values denote better performance.

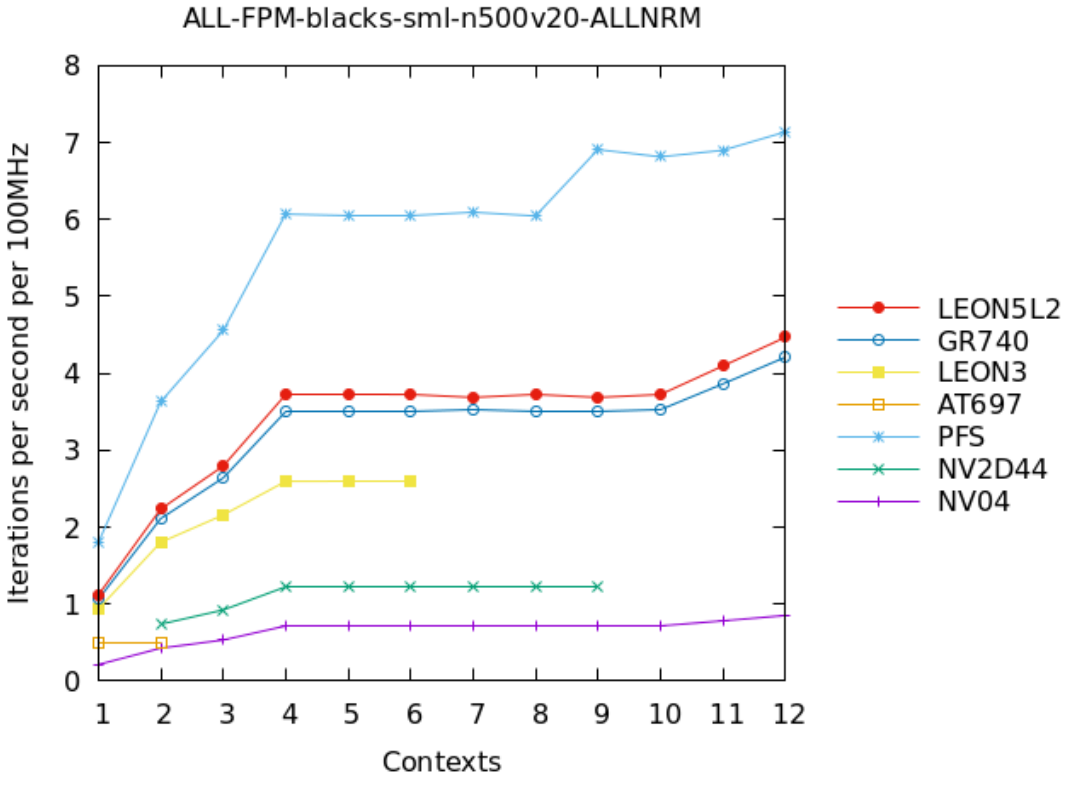

Figure 182: ALL - FPMark - blacks-sml-n500v20, iterations per second at 100MHz. Higher values denote better performance.

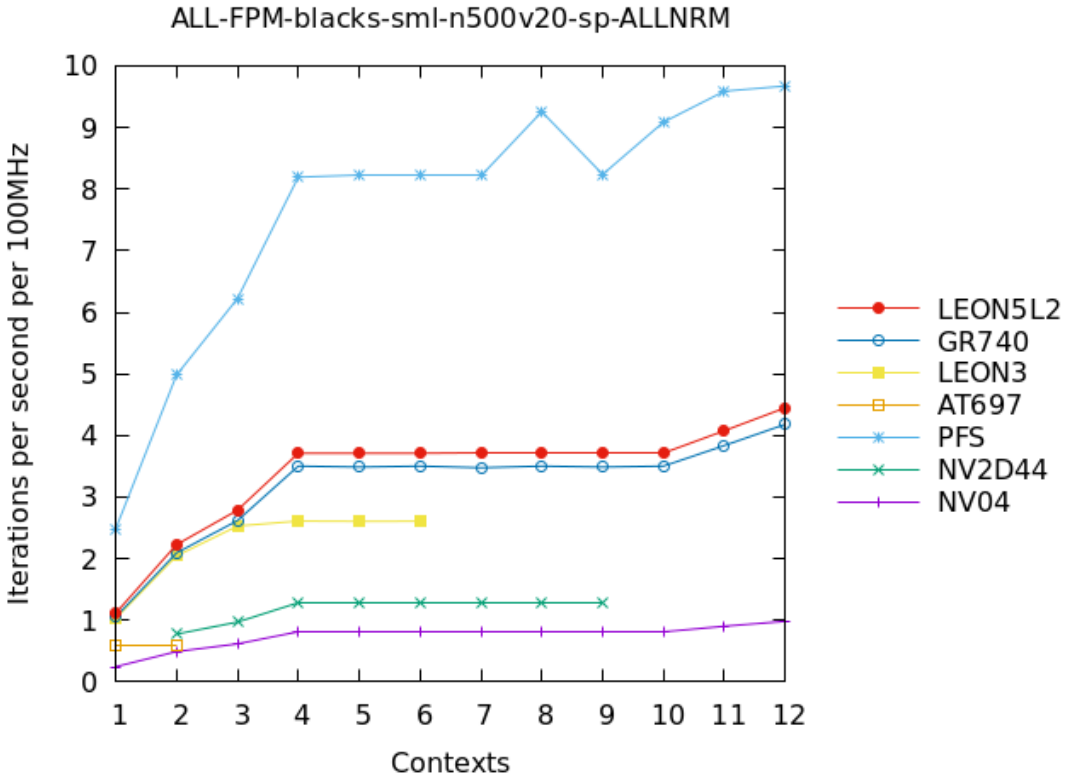

Figure 183: ALL - FPMark - blacks-sml-n500v20-sp, iterations per second at 100MHz. Higher values denote better performance.

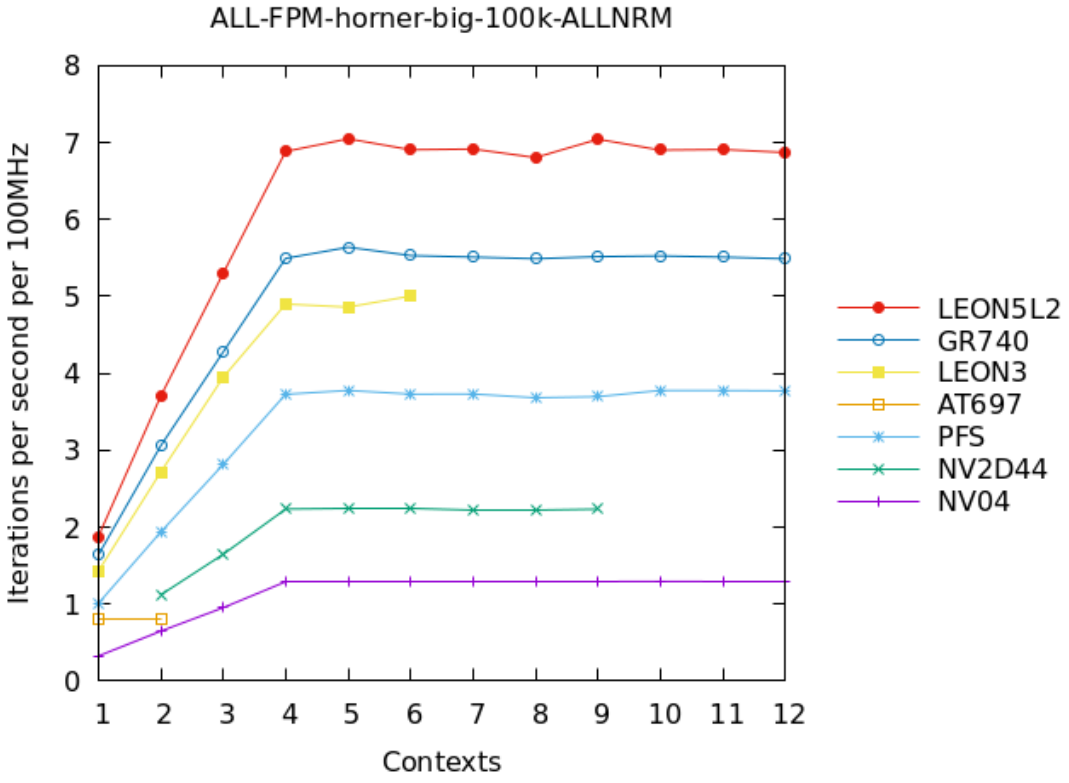

Figure 184: ALL - FPMark - horner-big-100k, iterations per second at 100MHz. Higher values denote better performance.

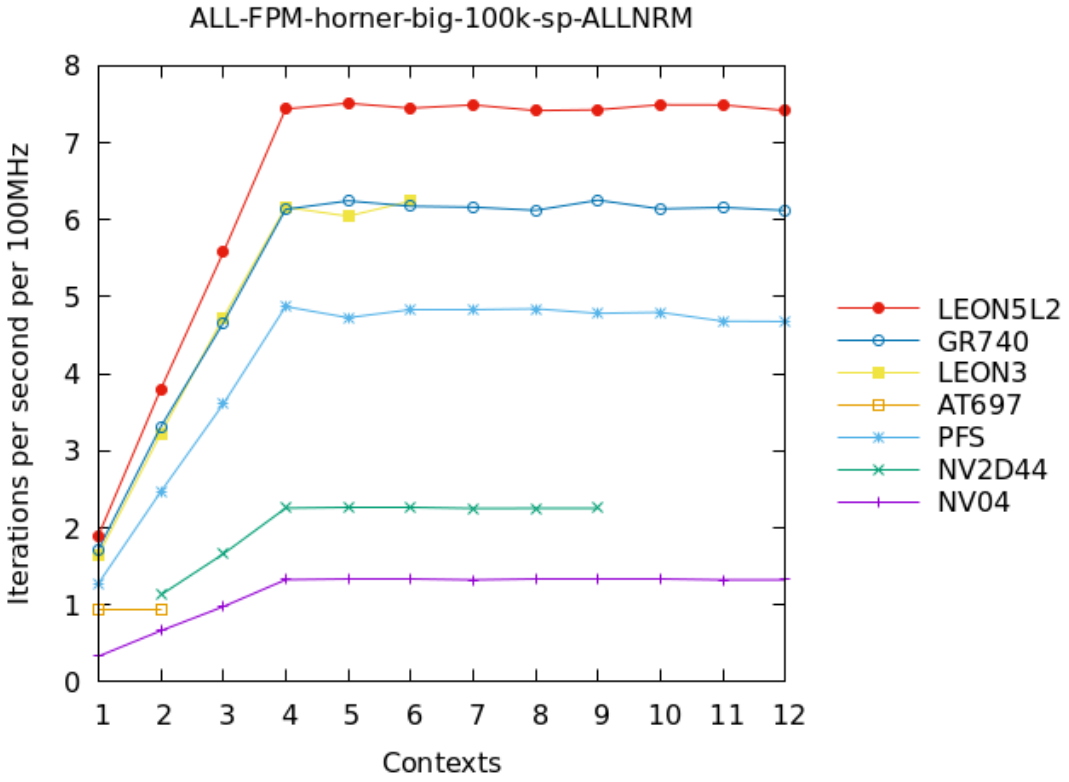

Figure 185: ALL - FPMark - horner-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

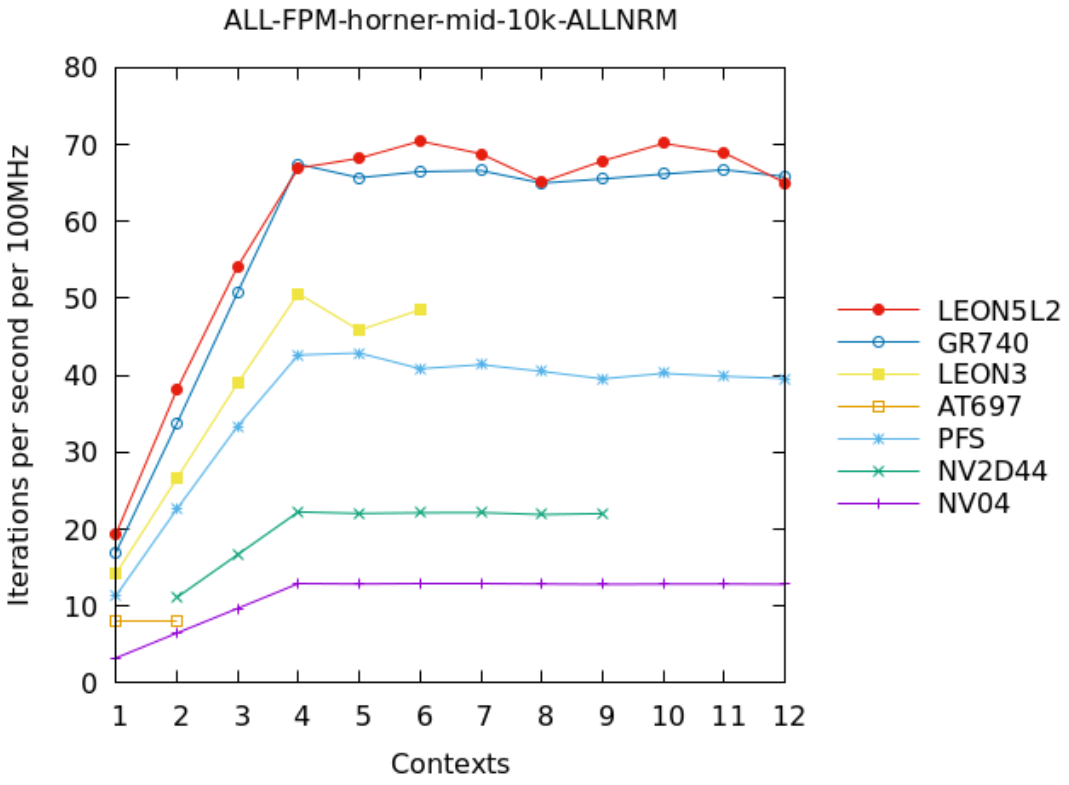

Figure 186: ALL - FPMark - horner-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

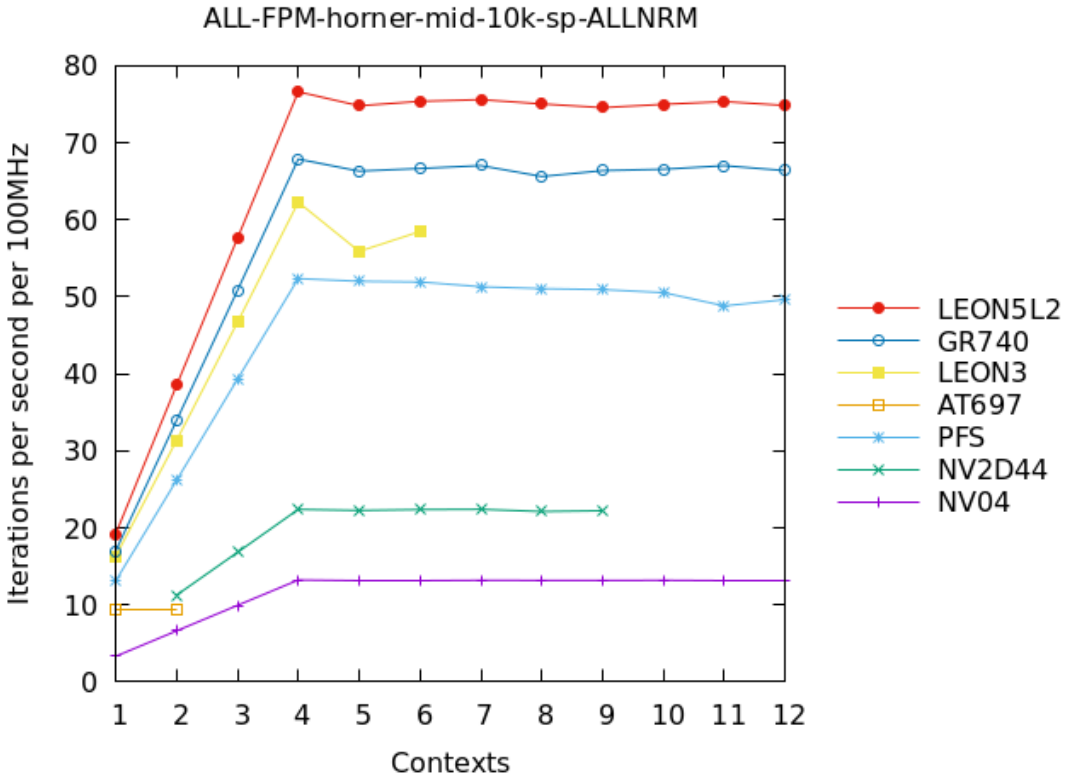

Figure 187: ALL - FPMark - horner-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

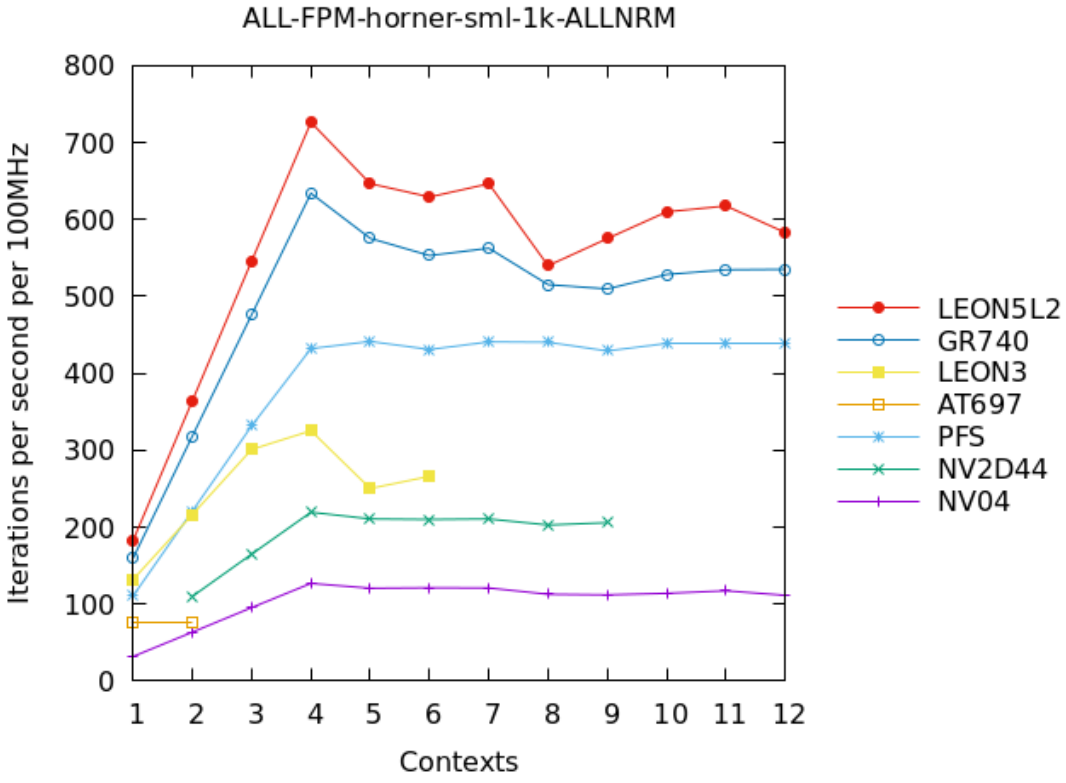

Figure 188: ALL - FPMark - horner-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

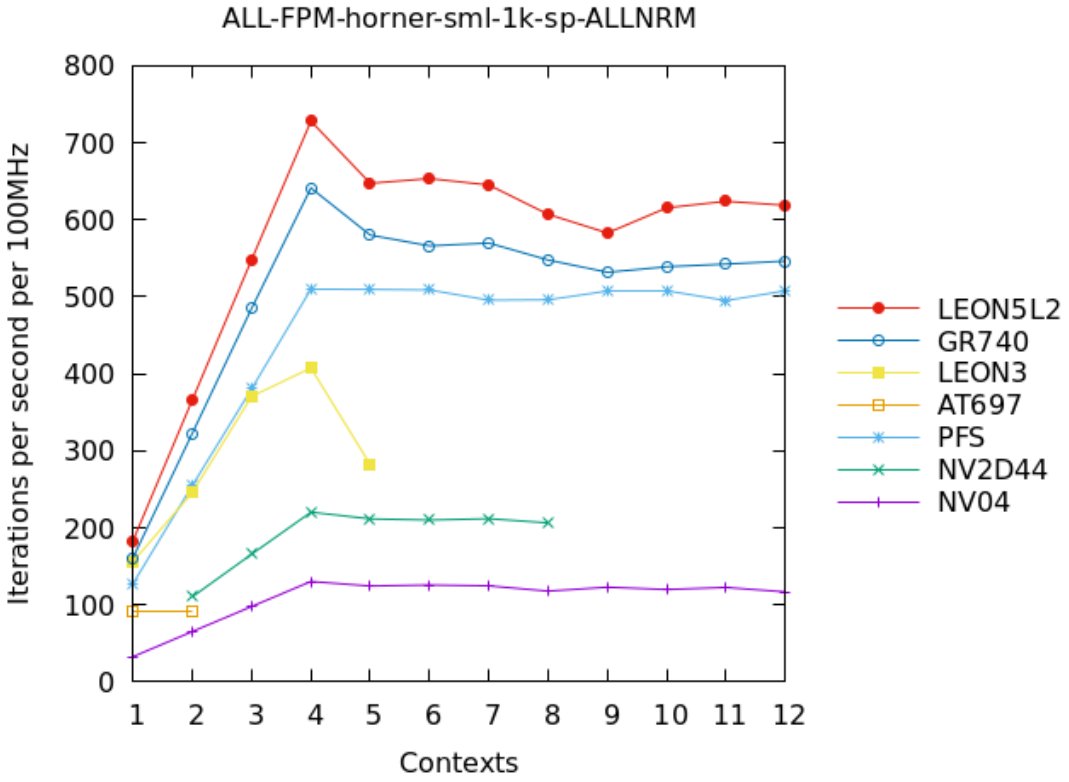

Figure 189: ALL - FPMark - horner-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

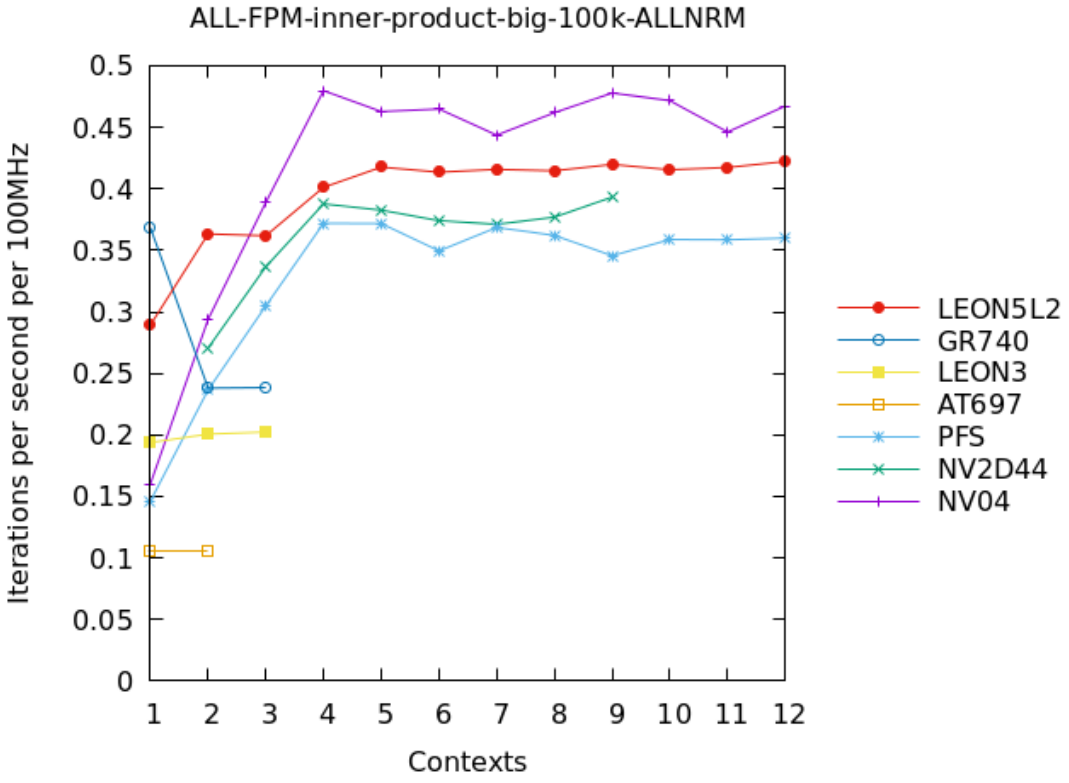

Figure 190: ALL - FPMark - inner-product-big-100k, iterations per second at 100MHz. Higher values denote better performance.

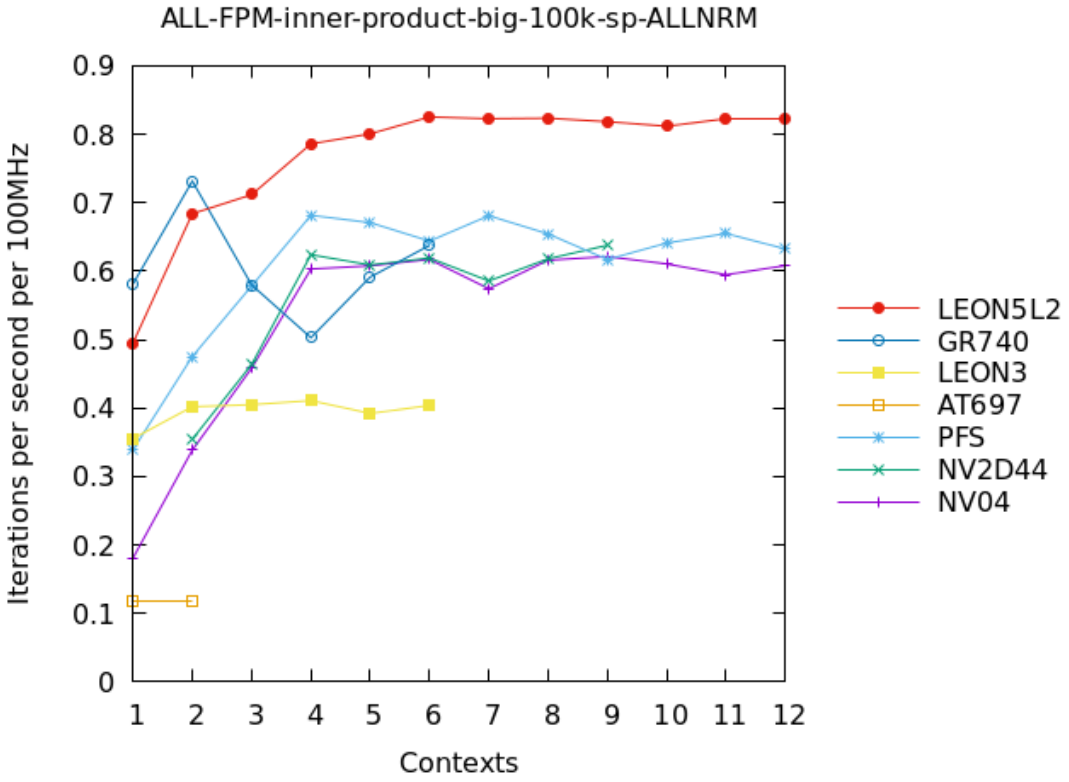

Figure 191: ALL - FPMark - inner-product-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

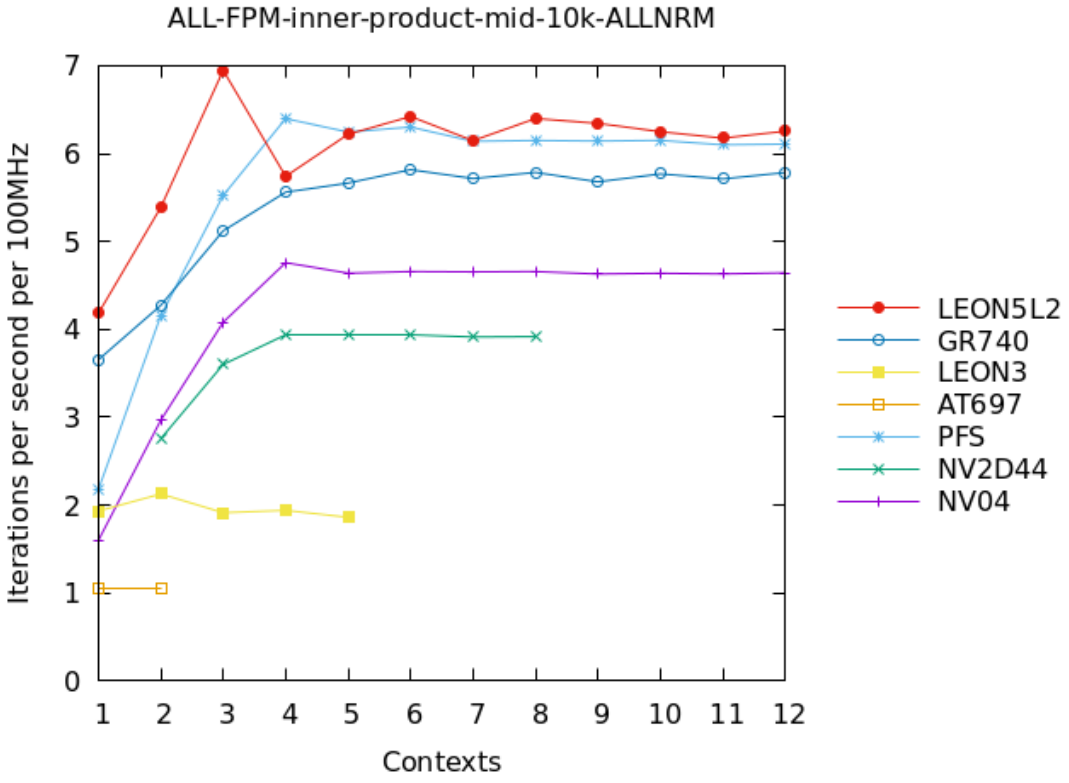

Figure 192: ALL - FPMark - inner-product-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

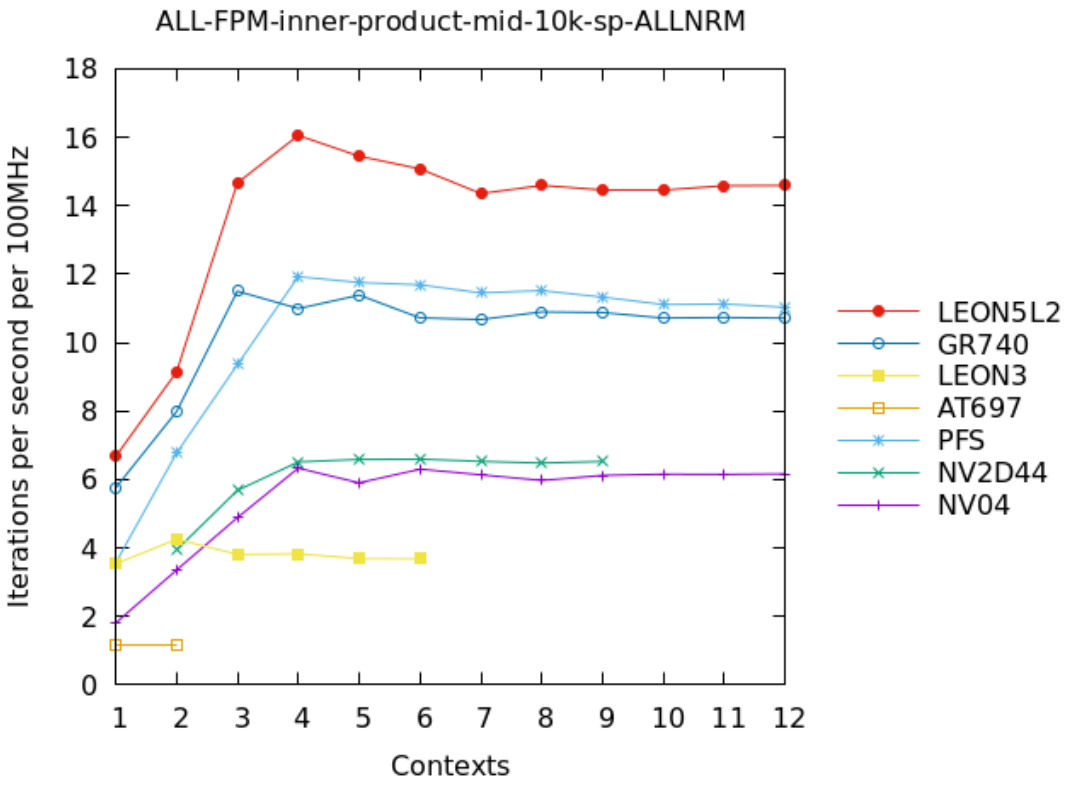

Figure 193: ALL - FPMark - inner-product-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

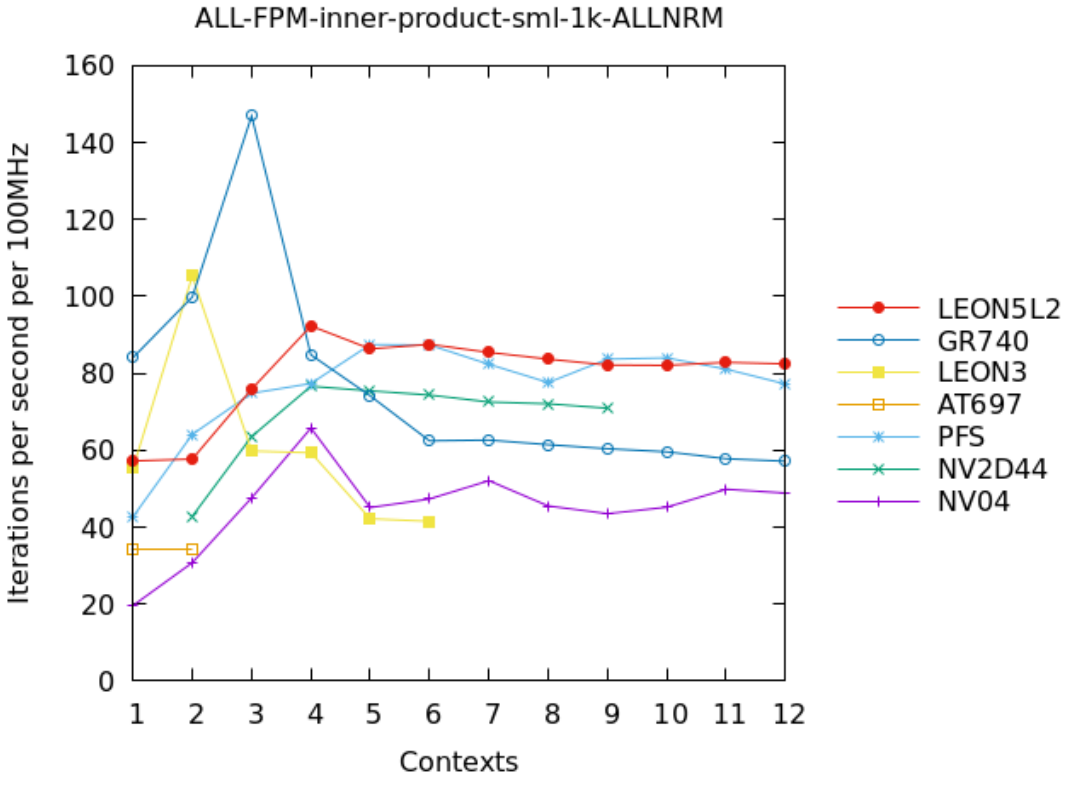

Figure 194: ALL - FPMark - inner-product-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

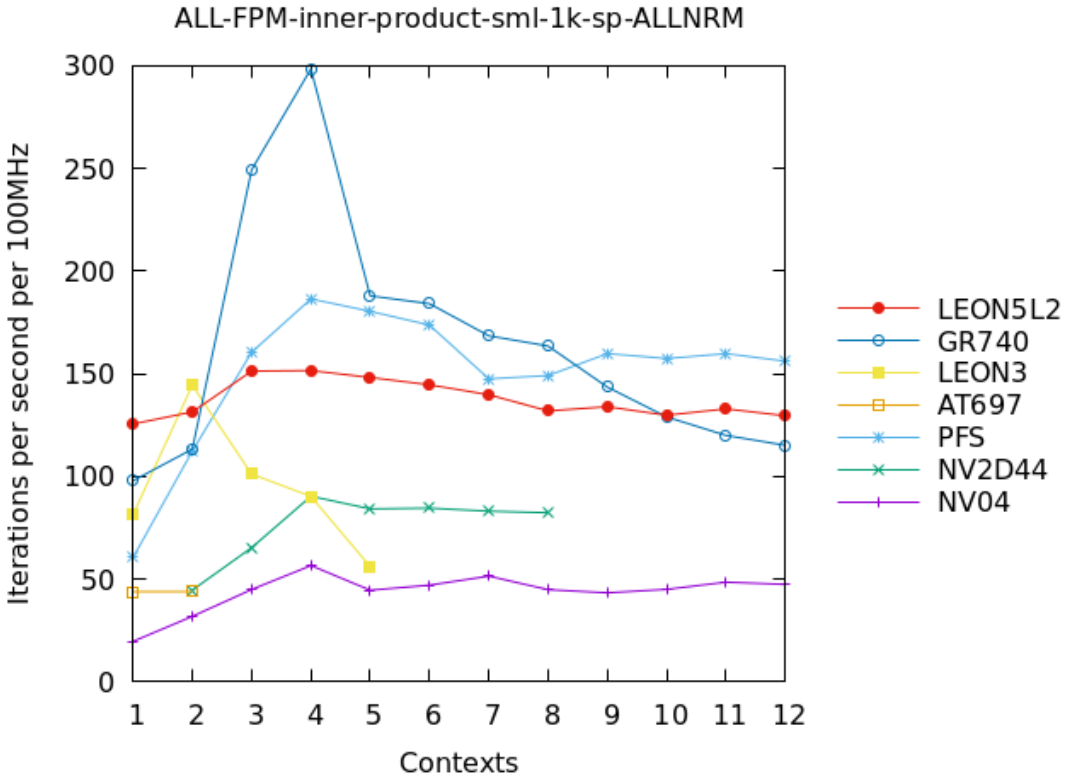

Figure 195: ALL - FPMark - inner-product-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

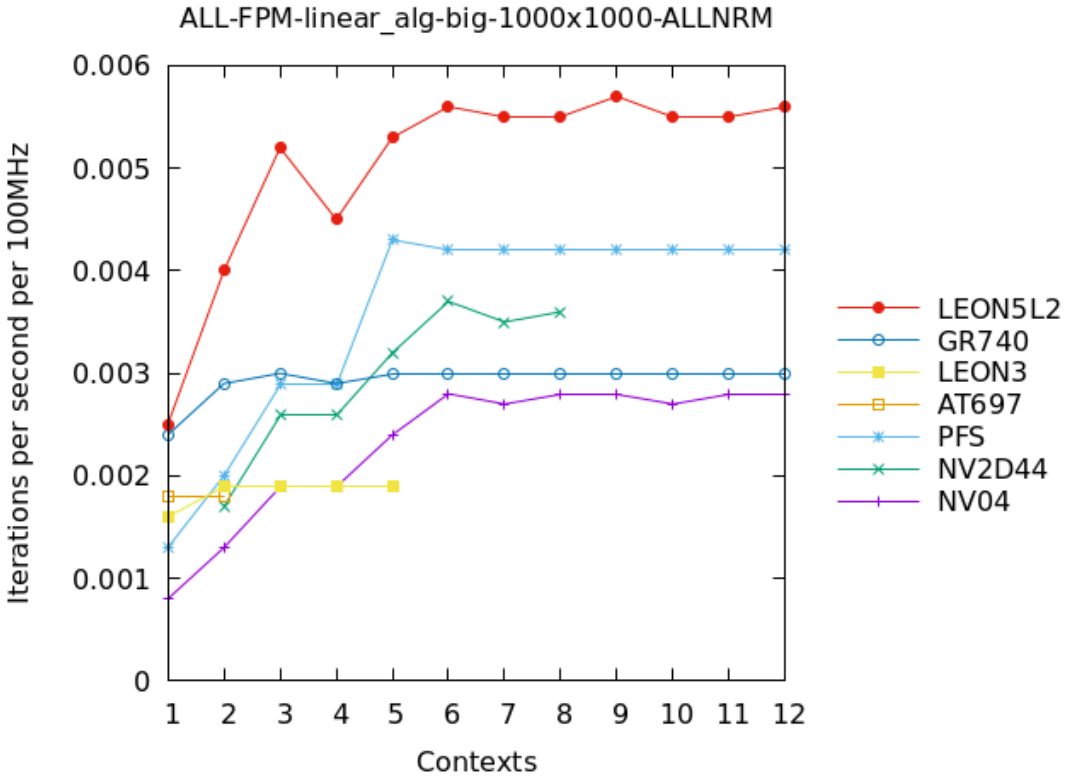

Figure 196: ALL - FPMark - linear_alg-big-1000x1000, iterations per second at 100MHz. Higher values denote better performance.

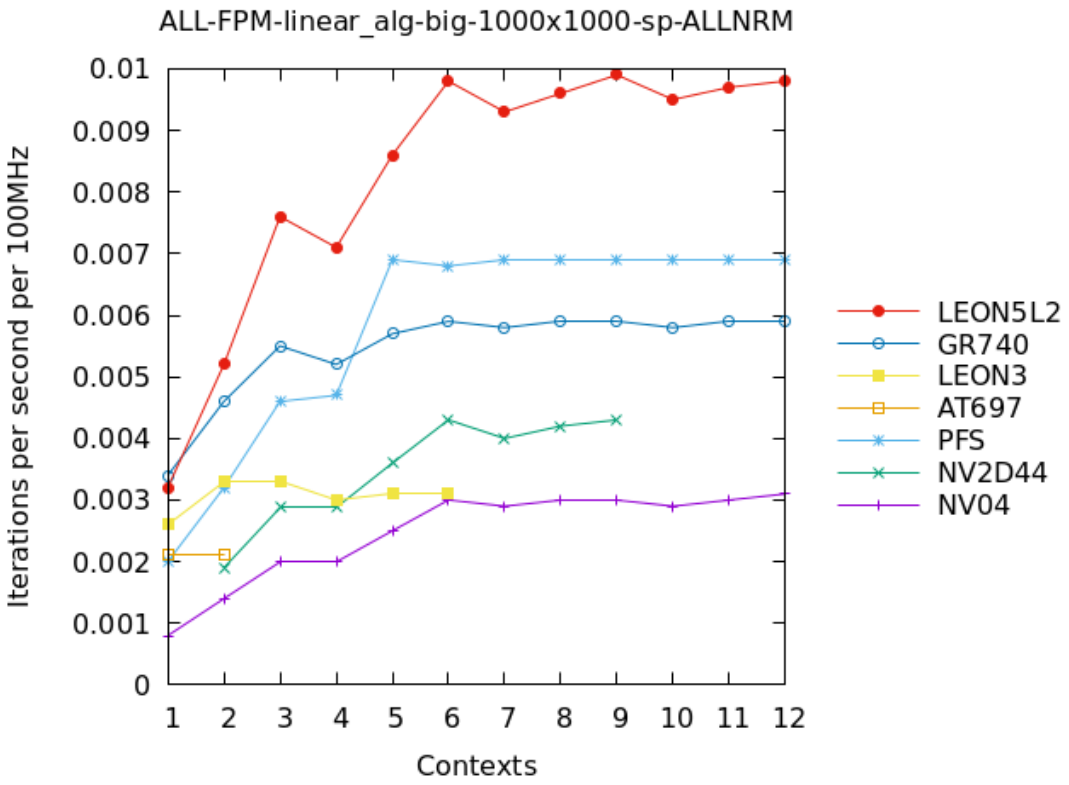

Figure 197: ALL - FPMark - linear_alg-big-1000x1000-sp, iterations per second at 100MHz. Higher values denote better performance.

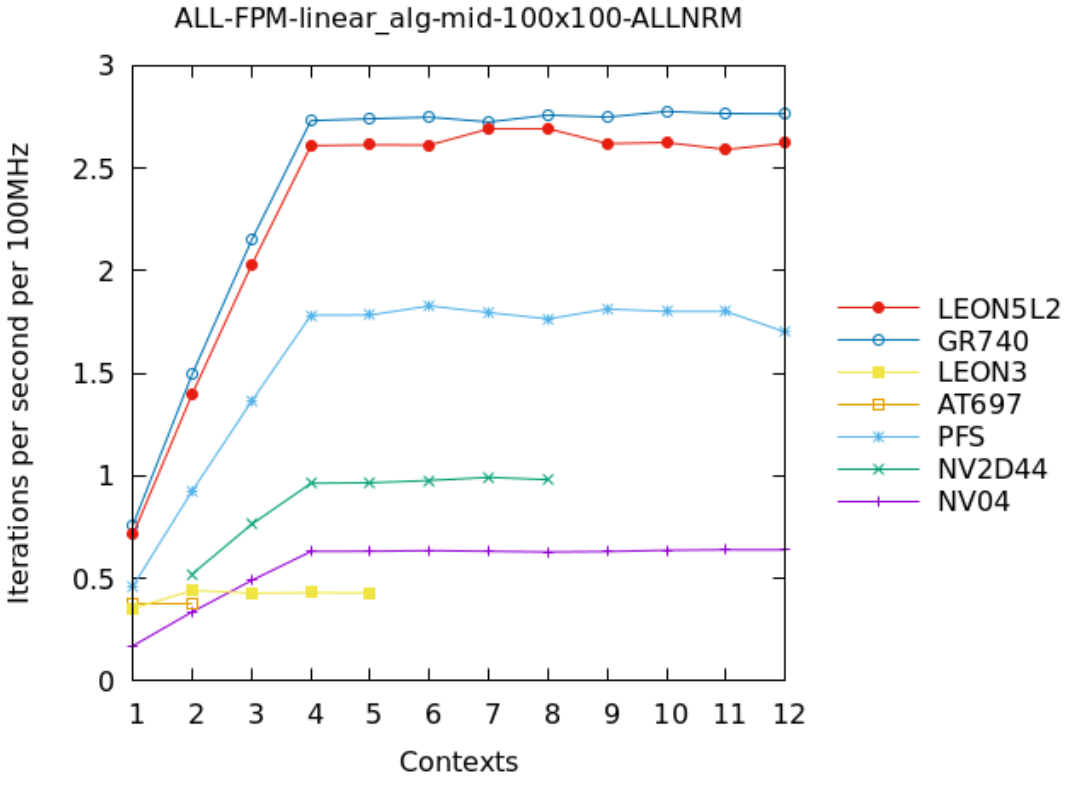

Figure 198: ALL - FPMark - linear_alg-mid-100x100, iterations per second at 100MHz. Higher values denote better performance.

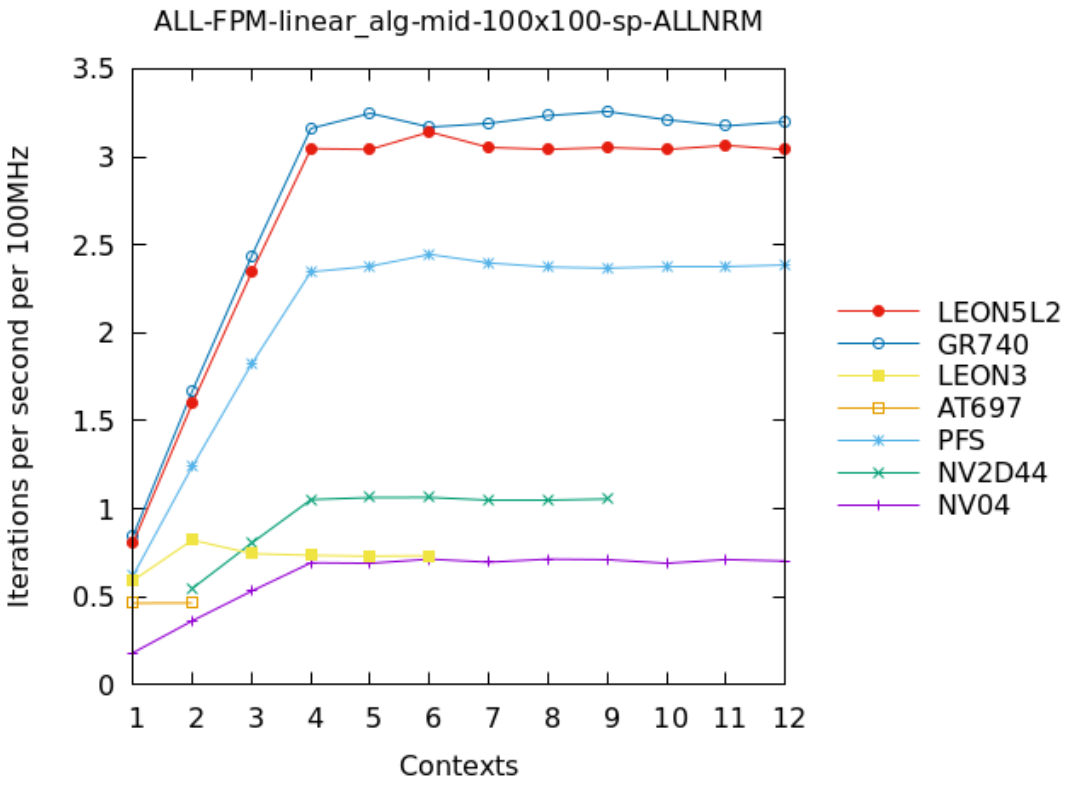

Figure 199: ALL - FPMark - linear_alg-mid-100x100-sp, iterations per second at 100MHz. Higher values denote better performance.

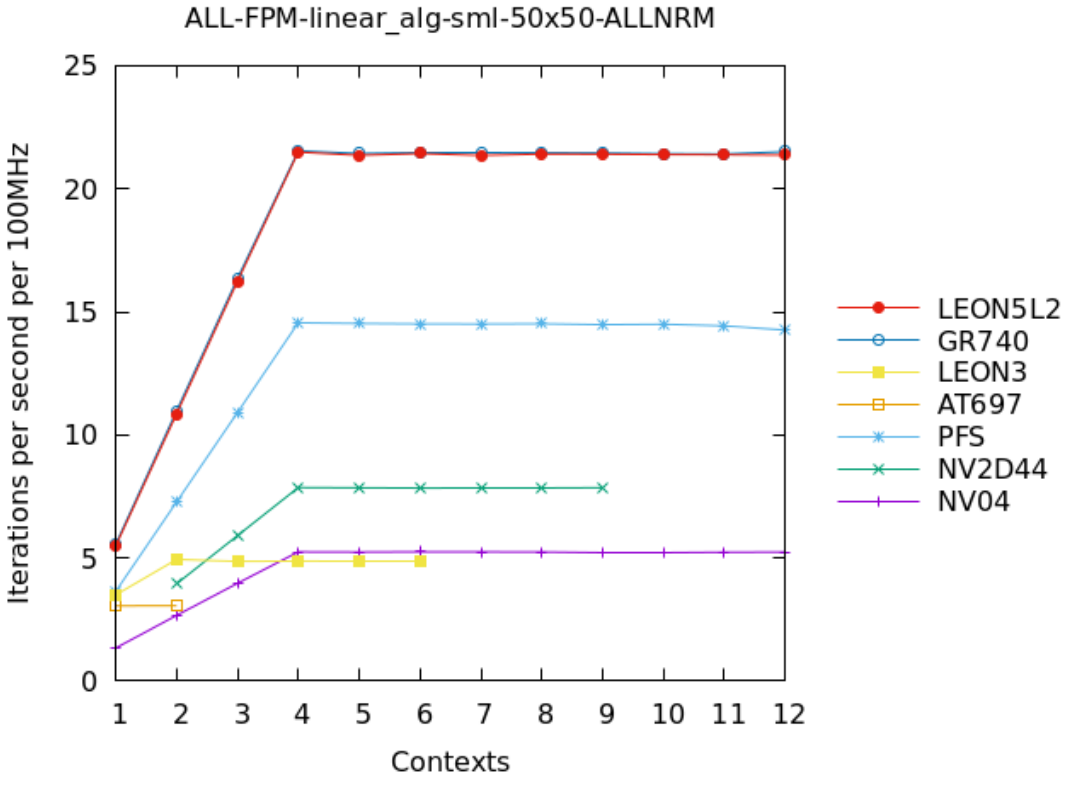

Figure 200: ALL - FPMark - linear_alg-sml-50x50, iterations per second at 100MHz. Higher values denote better performance.

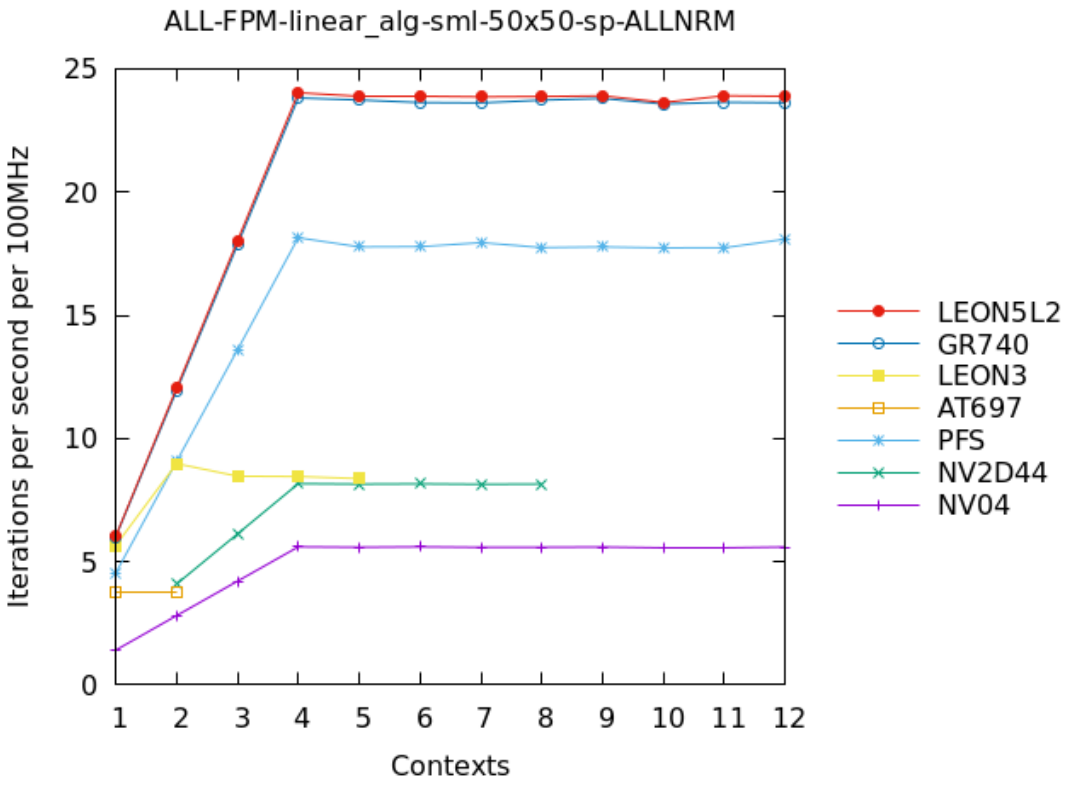

Figure 201: ALL - FPMark - linear_alg-sml-50x50-sp, iterations per second at 100MHz. Higher values denote better performance.

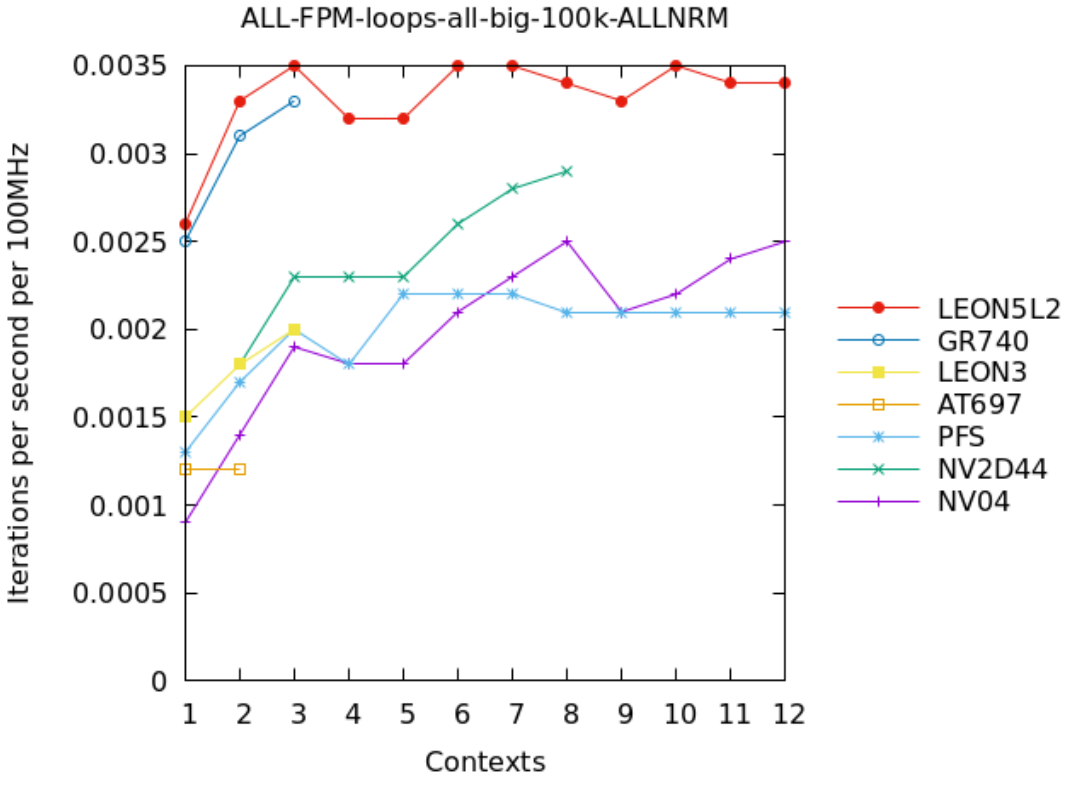

Figure 202: ALL - FPMark - loops-all-big-100k, iterations per second at 100MHz. Higher values denote better performance.

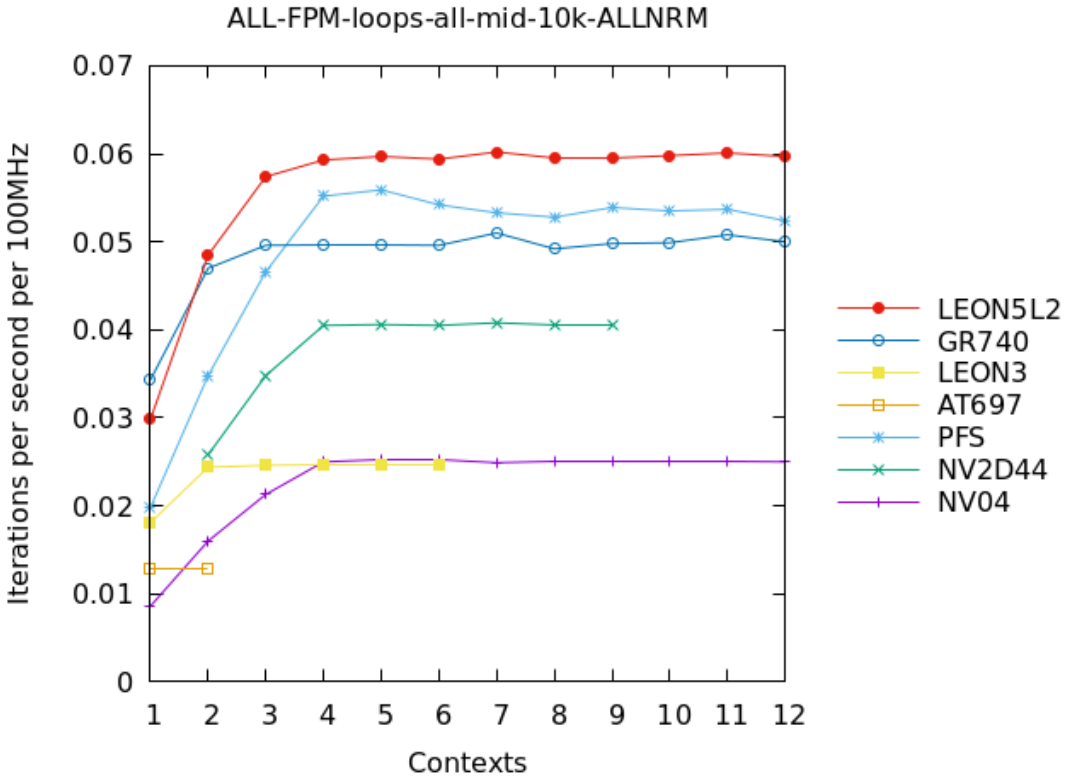

Figure 203: ALL - FPMark - loops-all-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

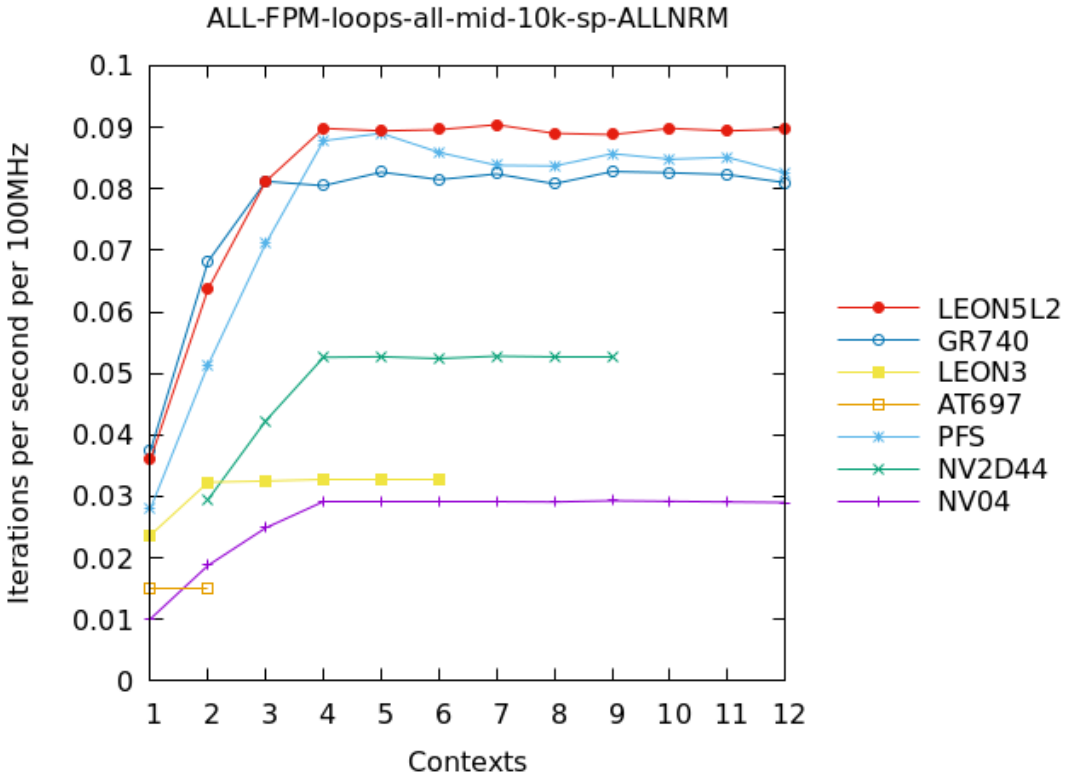

Figure 204: ALL - FPMark - loops-all-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

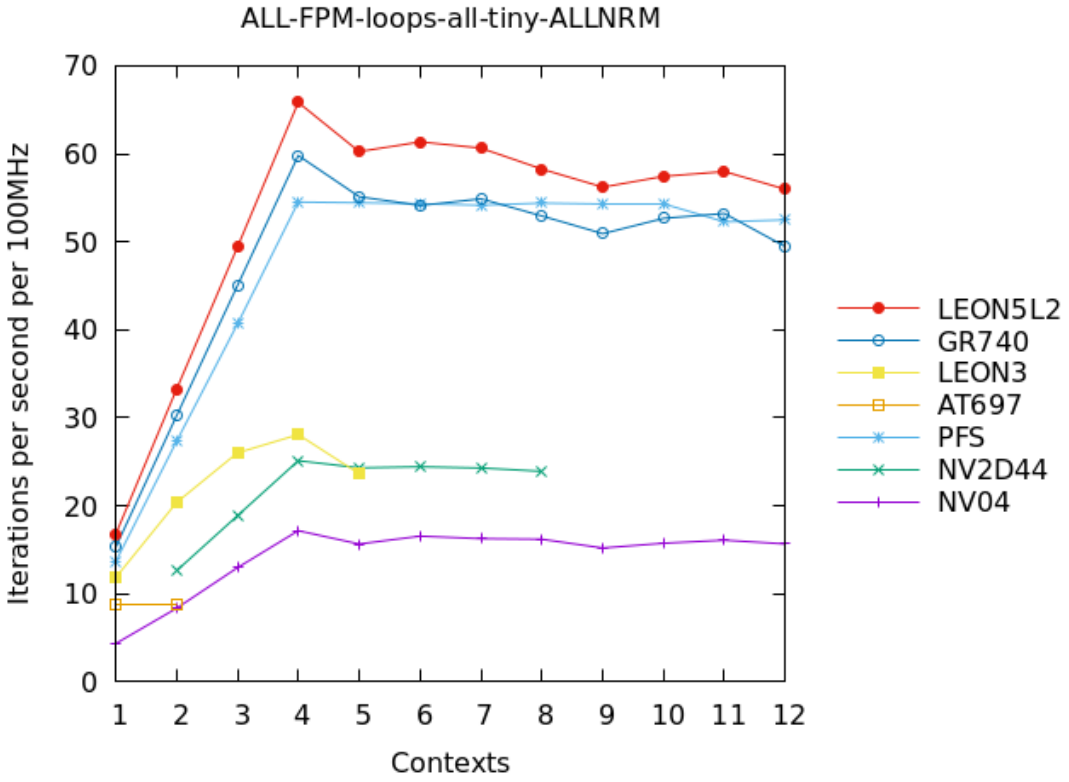

Figure 205: ALL - FPMark - loops-all-tiny, iterations per second at 100MHz. Higher values denote better performance.

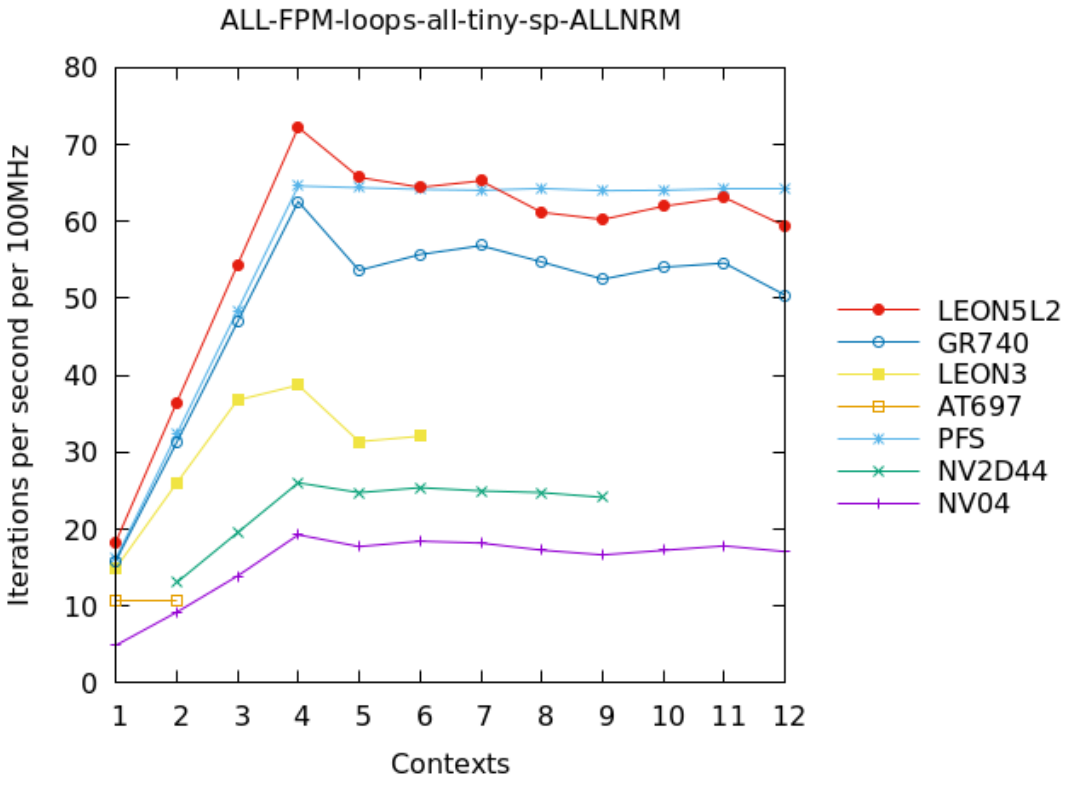

Figure 206: ALL - FPMark - loops-all-tiny-sp, iterations per second at 100MHz. Higher values denote better performance.

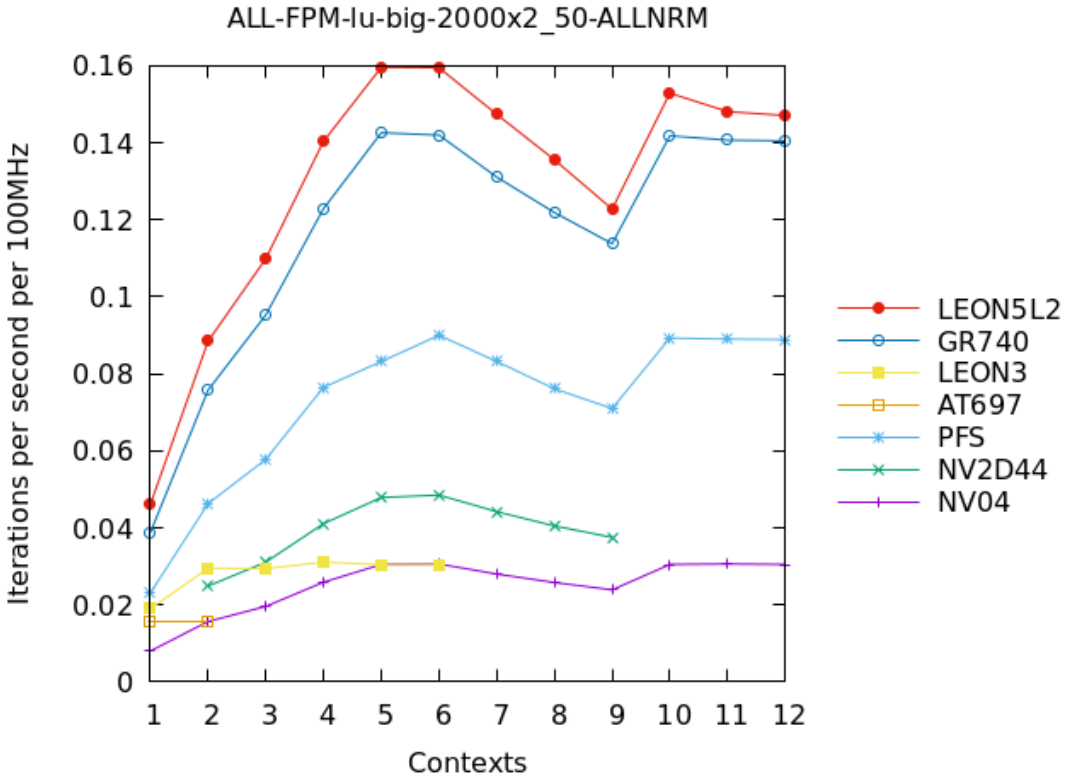

Figure 207: ALL - FPMark - lu-big-2000x2_50, iterations per second at 100MHz. Higher values denote better performance.

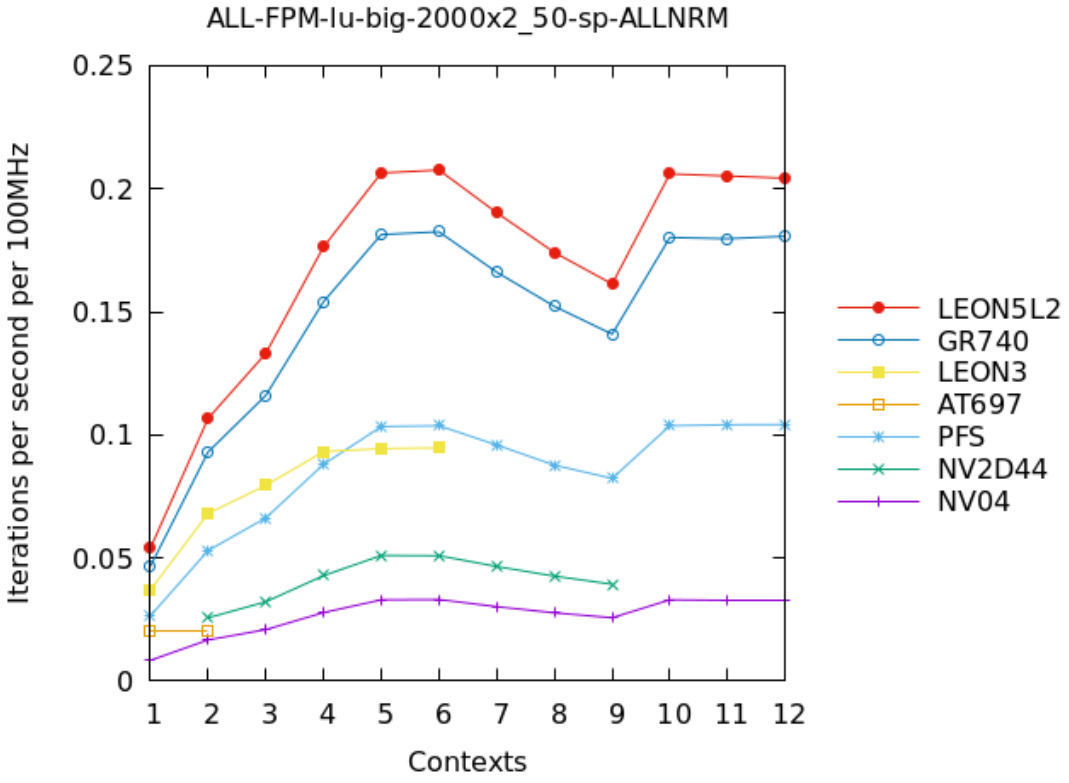

Figure 208: ALL - FPMark - lu-big-2000x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

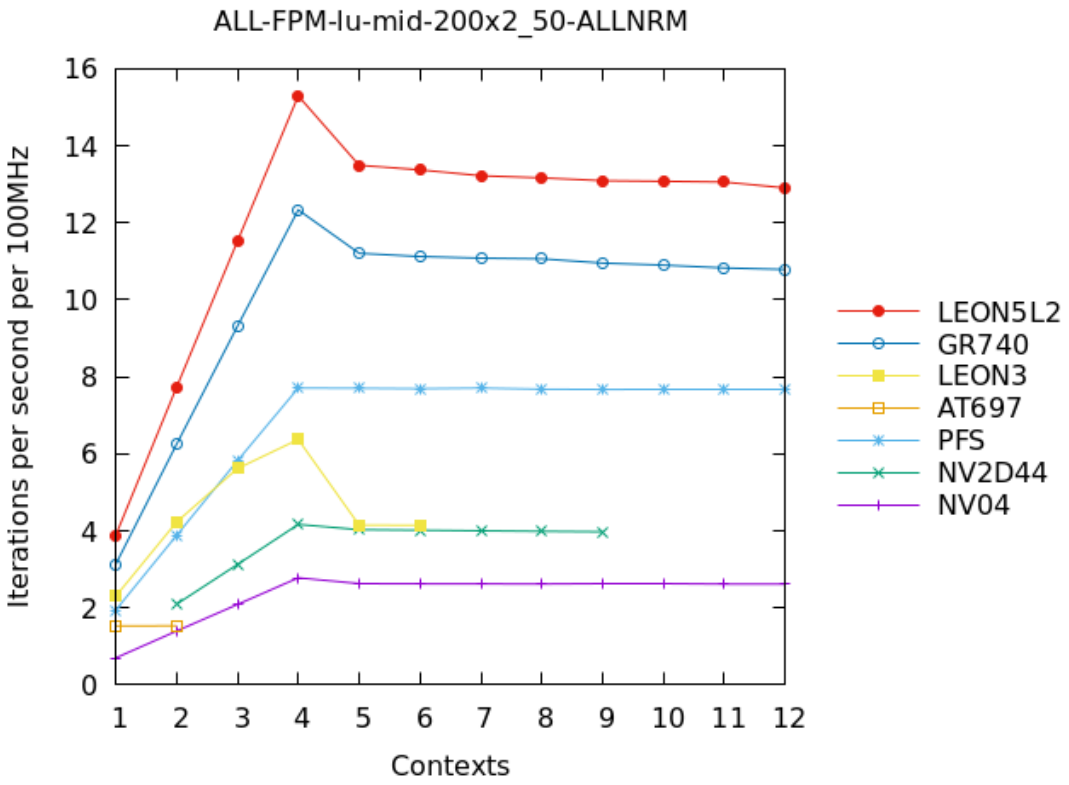

Figure 209: ALL - FPMark - lu-mid-200x2_50, iterations per second at 100MHz. Higher values denote better performance.

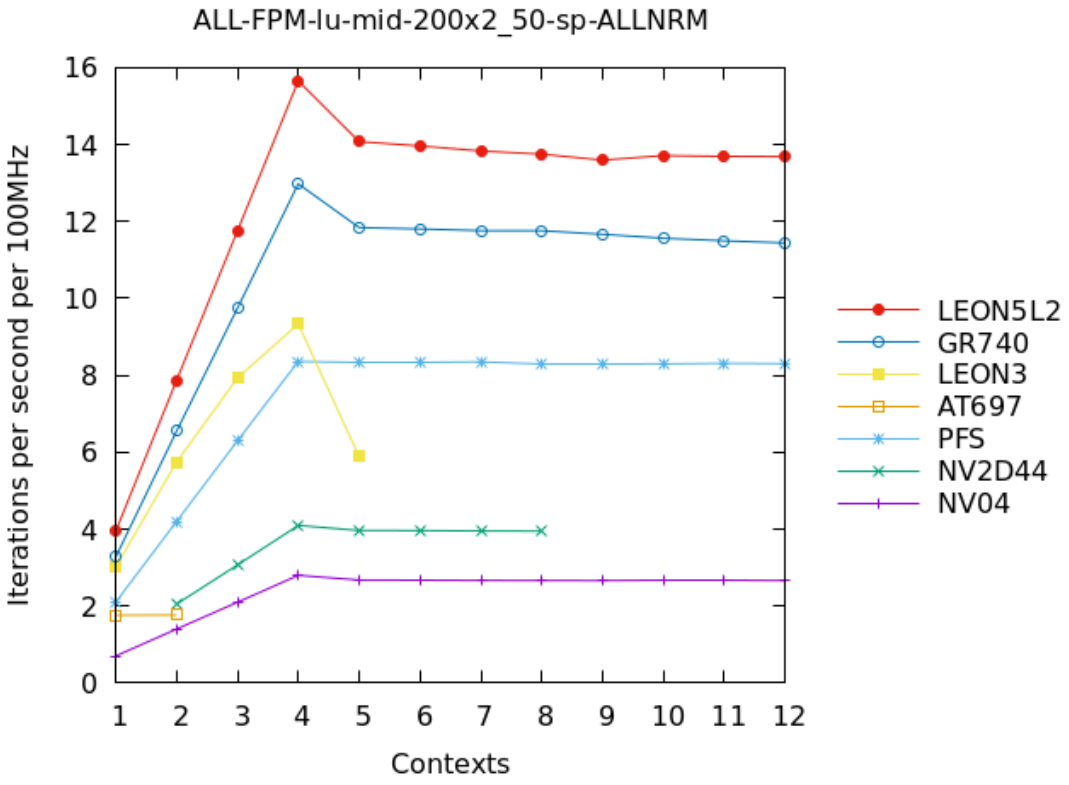

Figure 210: ALL - FPMark - lu-mid-200x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

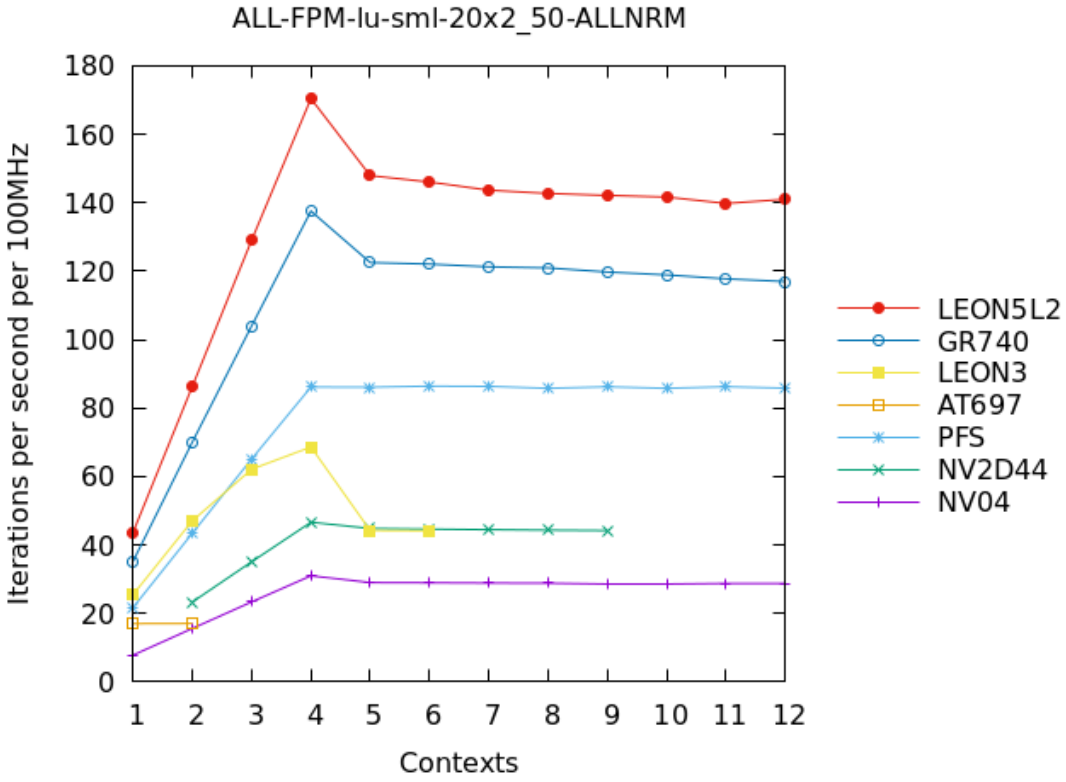

Figure 211: ALL - FPMark - lu-sml-20x2_50, iterations per second at 100MHz. Higher values denote better performance.

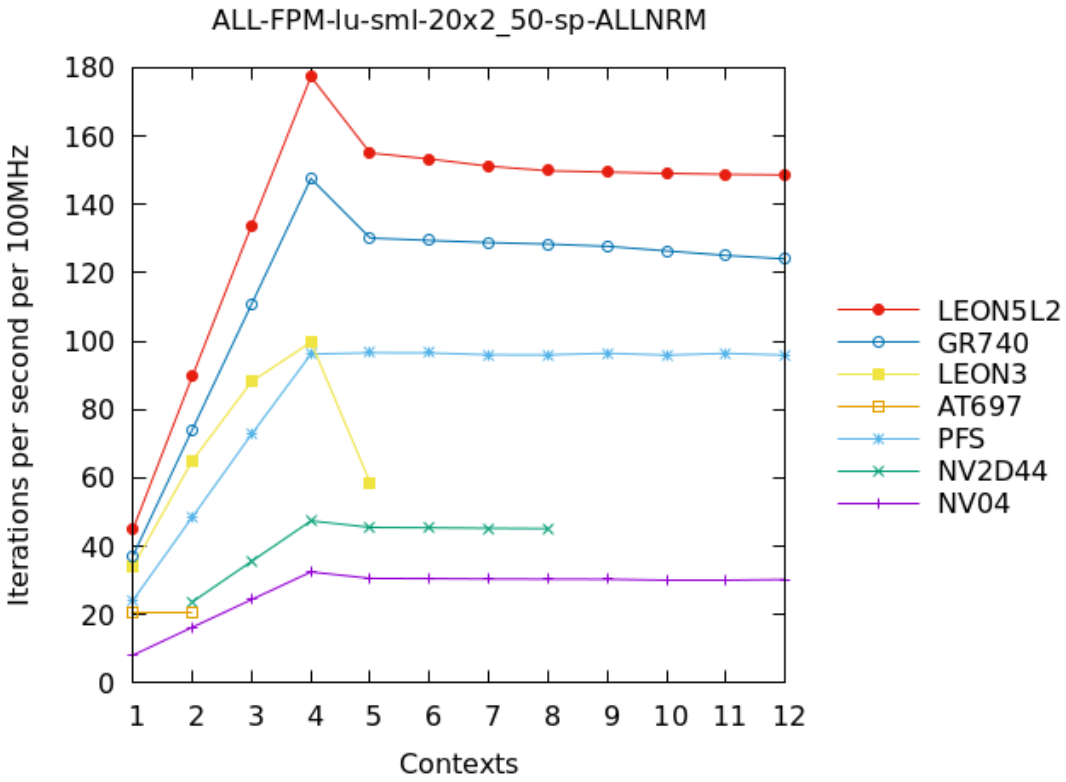

Figure 212: ALL - FPMark - lu-sml-20x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

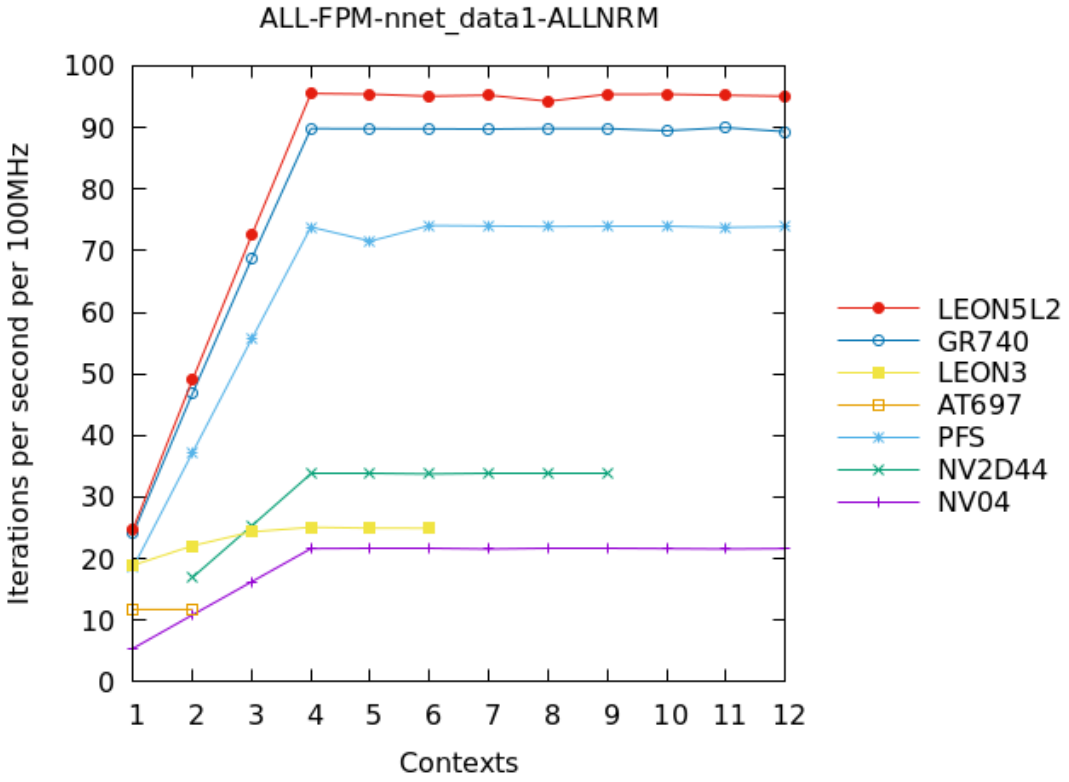

Figure 213: ALL - FPMark - nnet_data1, iterations per second at 100MHz. Higher values denote better performance.

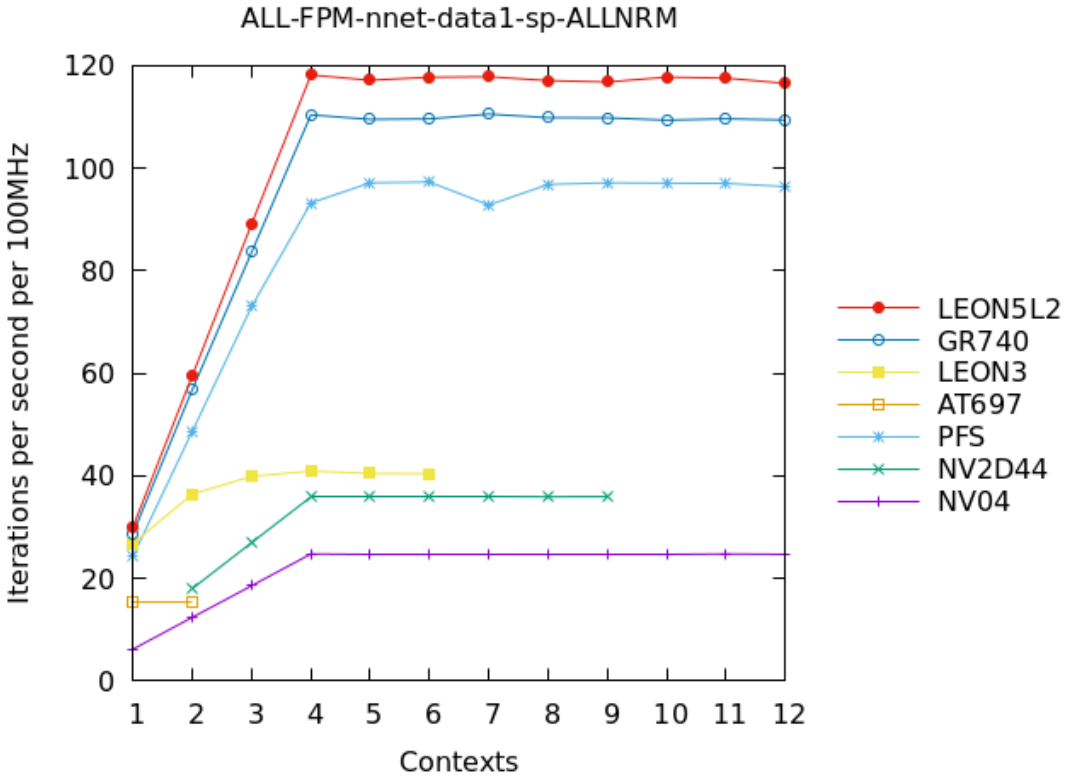

Figure 214: ALL - FPMark - nnet-data1-sp, iterations per second at 100MHz. Higher values denote better performance.

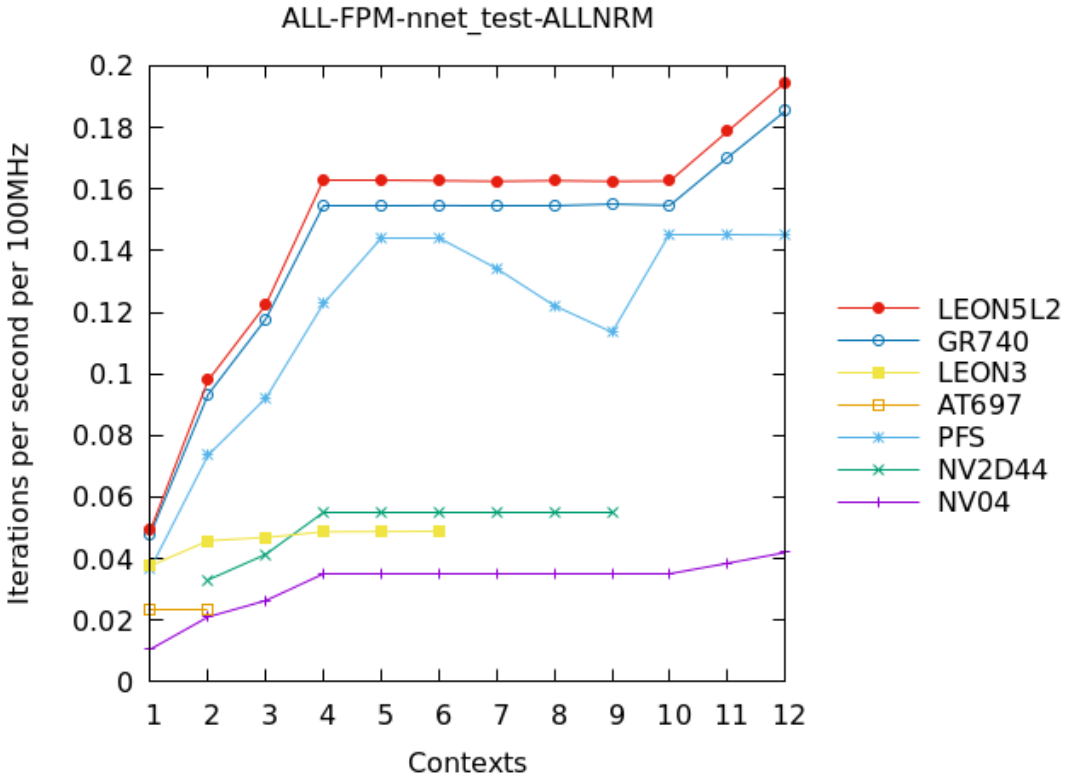

Figure 215: ALL - FPMark - nnet_test, iterations per second at 100MHz. Higher values denote better performance.

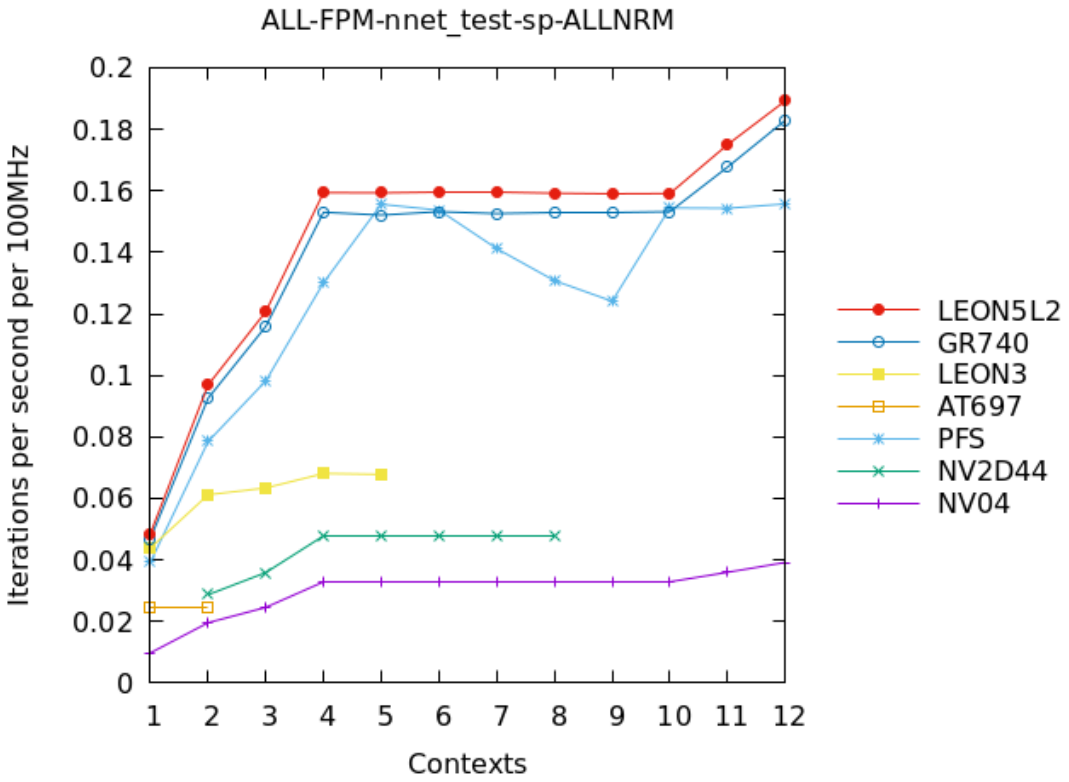

Figure 216: ALL - FPMark - nnet_test-sp, iterations per second at 100MHz. Higher values denote better performance.

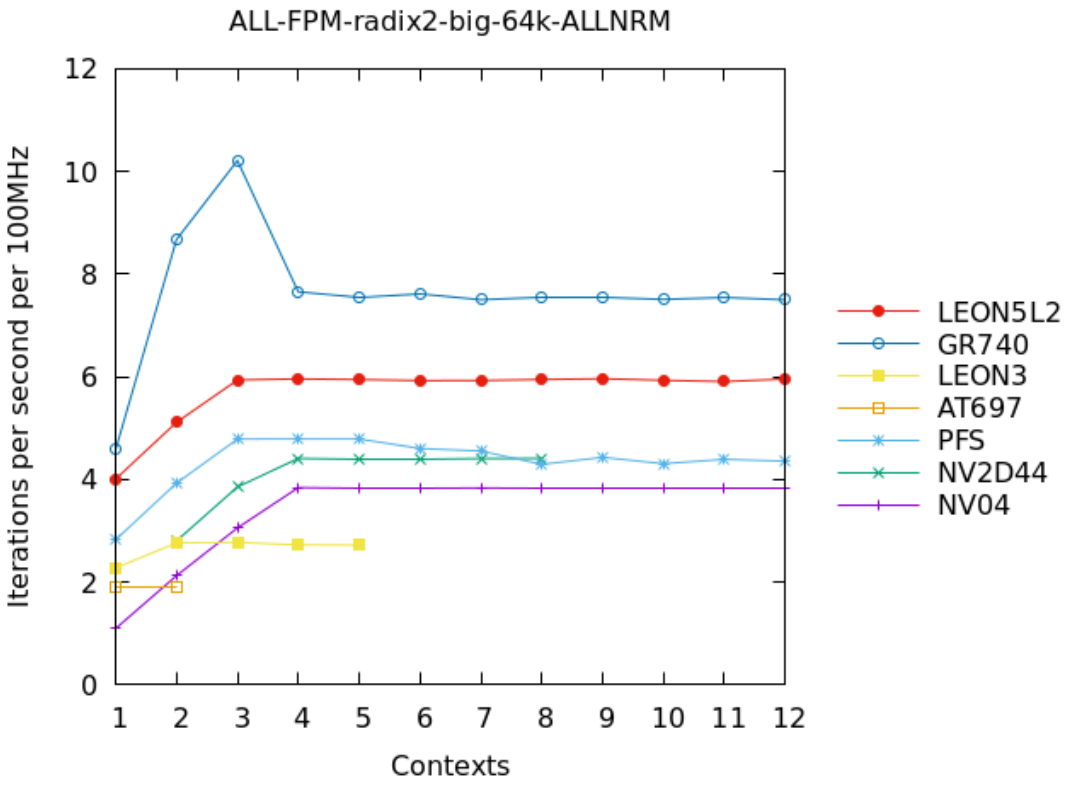

Figure 217: ALL - FPMark - radix2-big-64k, iterations per second at 100MHz. Higher values denote better performance.

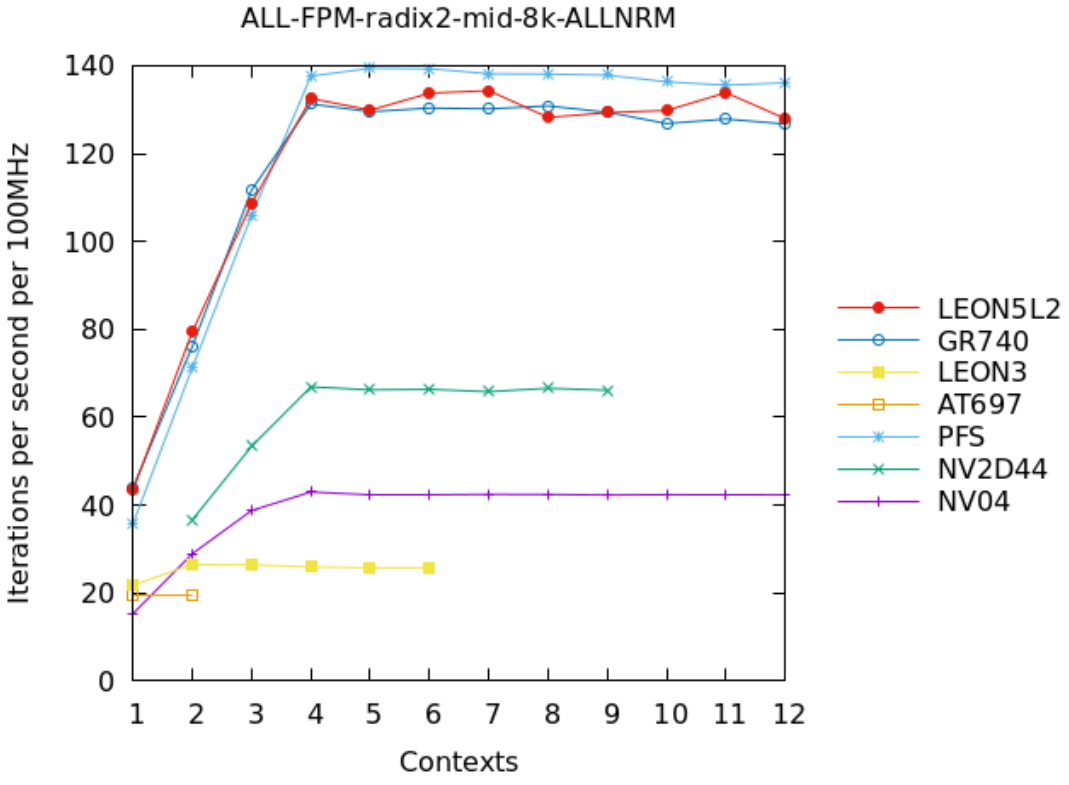

Figure 218: ALL - FPMark - radix2-mid-8k, iterations per second at 100MHz. Higher values denote better performance.

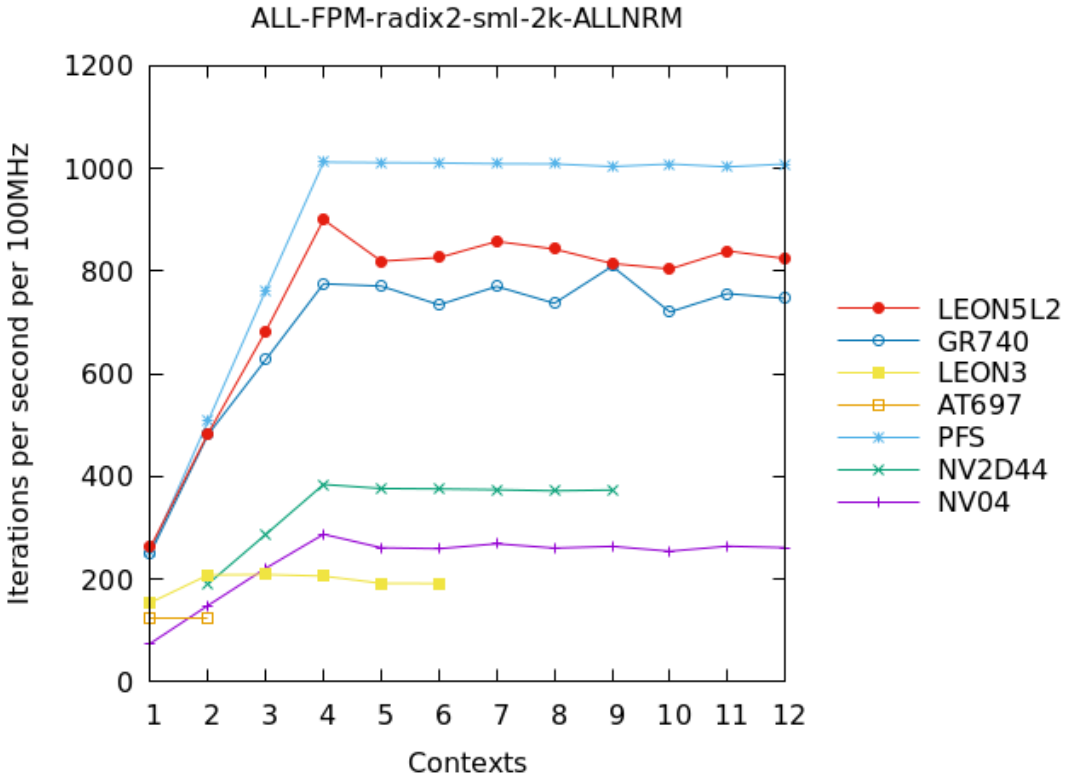

Figure 219: ALL - FPMark - radix2-sml-2k, iterations per second at 100MHz. Higher values denote better performance.

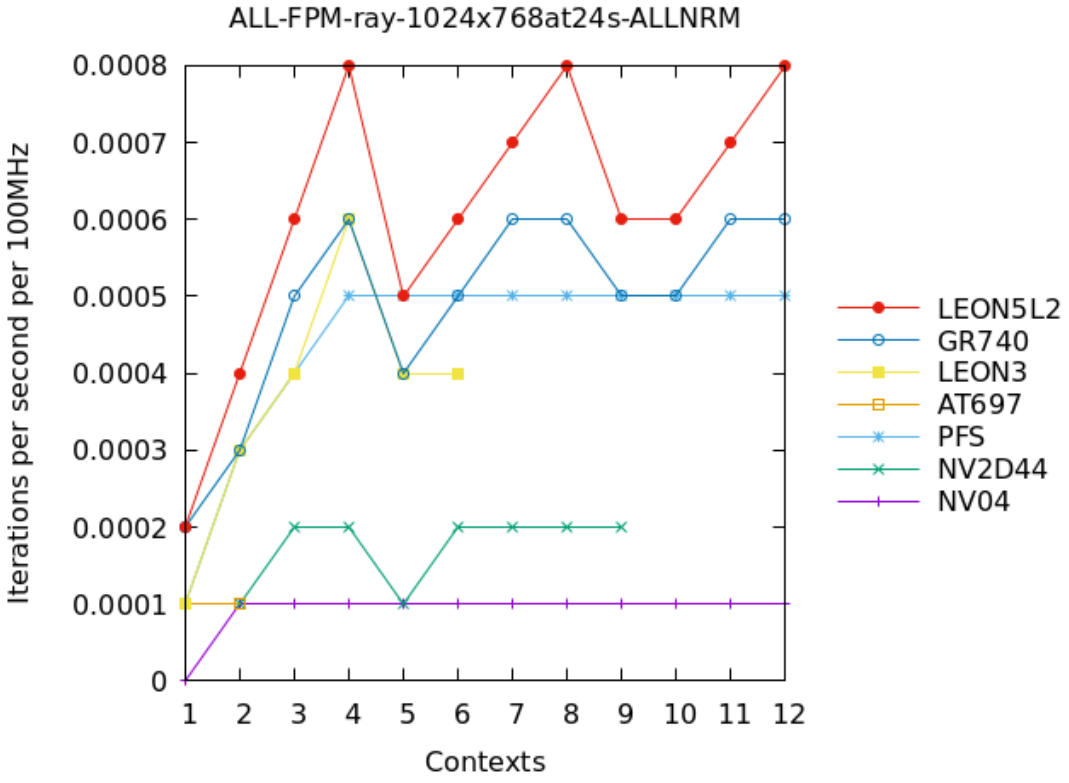

Figure 220: ALL - FPMark - ray-1024x768at24s, iterations per second at 100MHz. Higher values denote better performance.

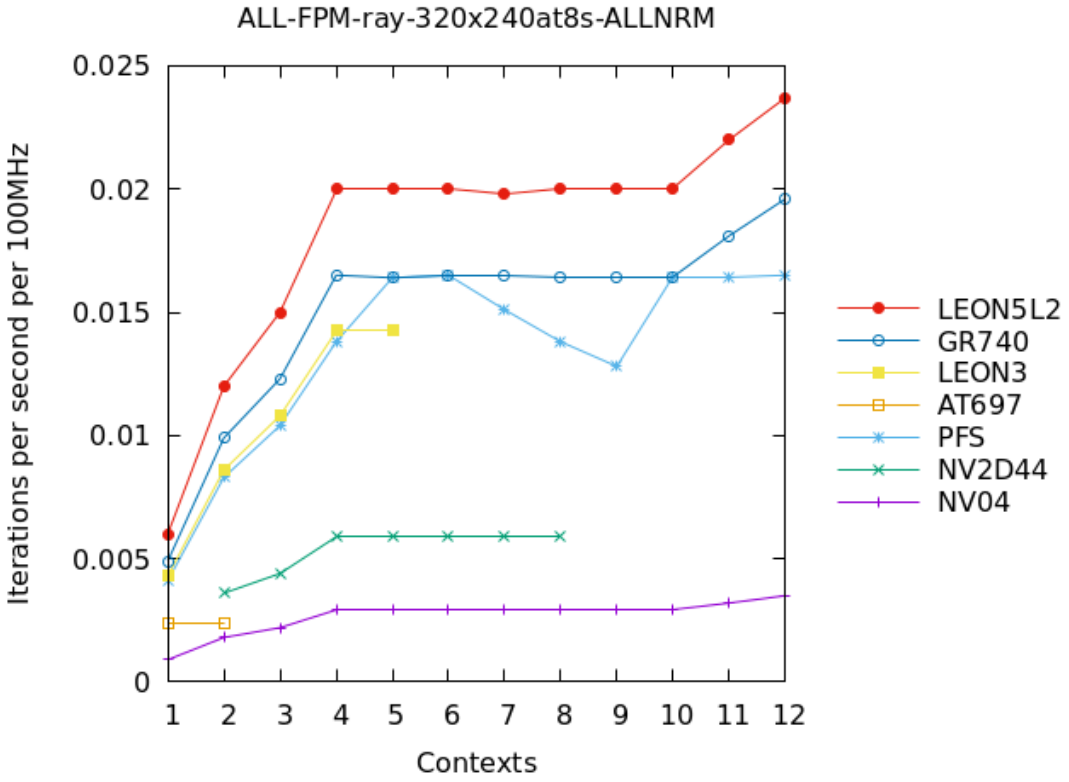

Figure 221: ALL - FPMark - ray-320x240at8s, iterations per second at 100MHz. Higher values denote better performance.

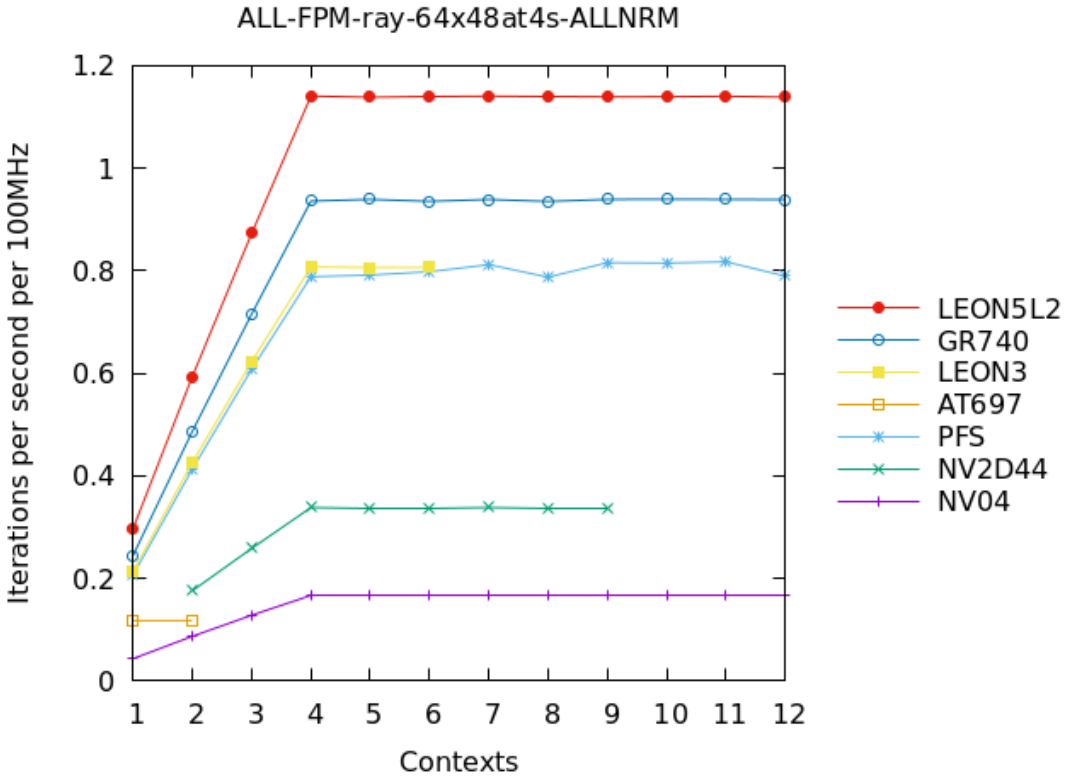

Figure 222: ALL - FPMark - ray-64x48at4s, iterations per second at 100MHz. Higher values denote better performance.

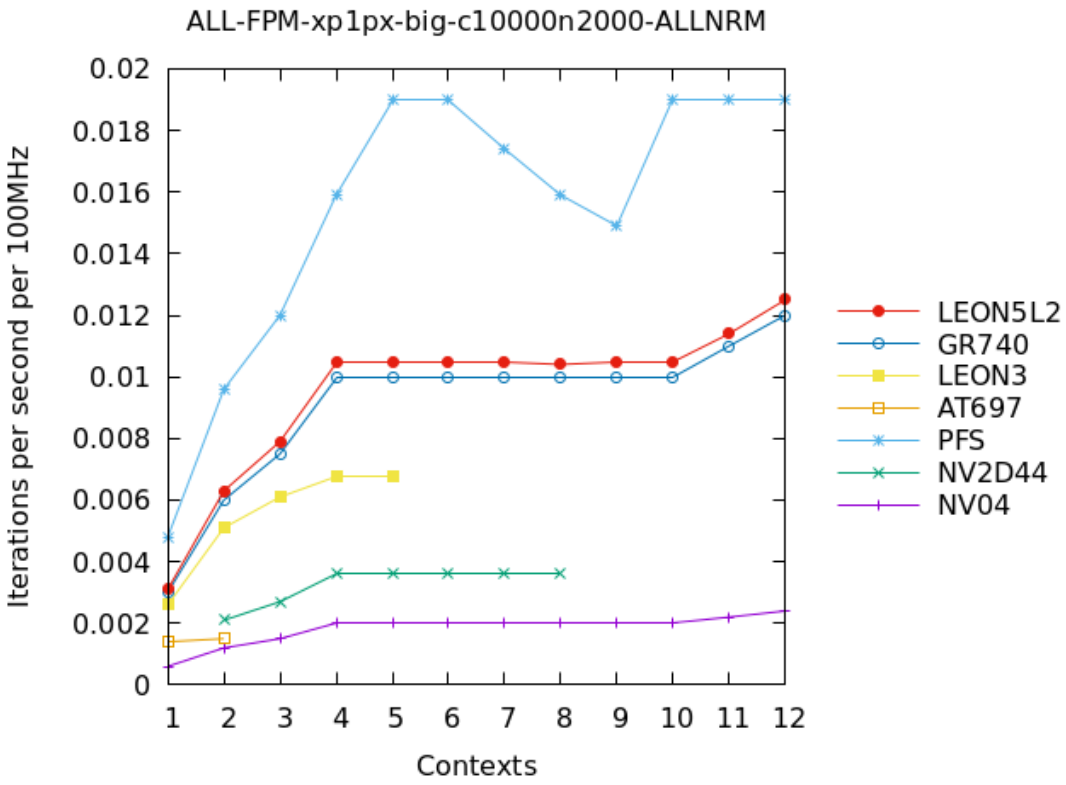

Figure 223: ALL - FPMark - xp1px-big-c10000n2000, iterations per second at 100MHz. Higher values denote better performance.

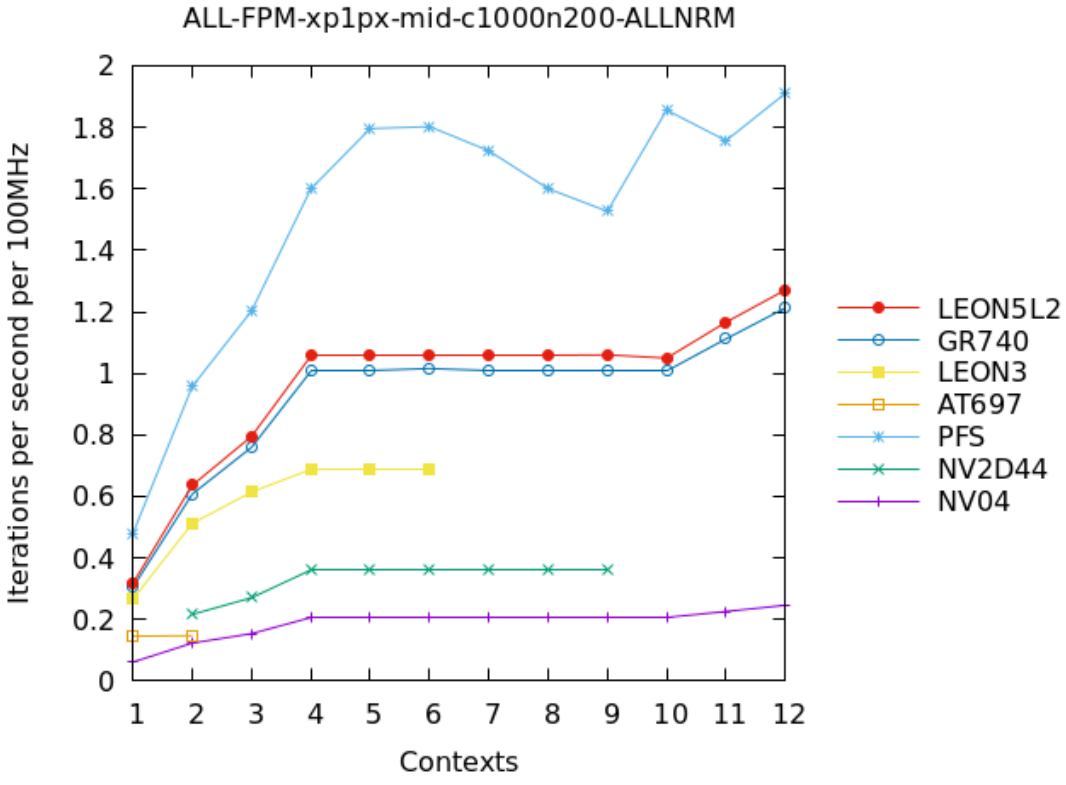

Figure 224: ALL - FPMark - xp1px-mid-c1000n200, iterations per second at 100MHz. Higher values denote better performance.

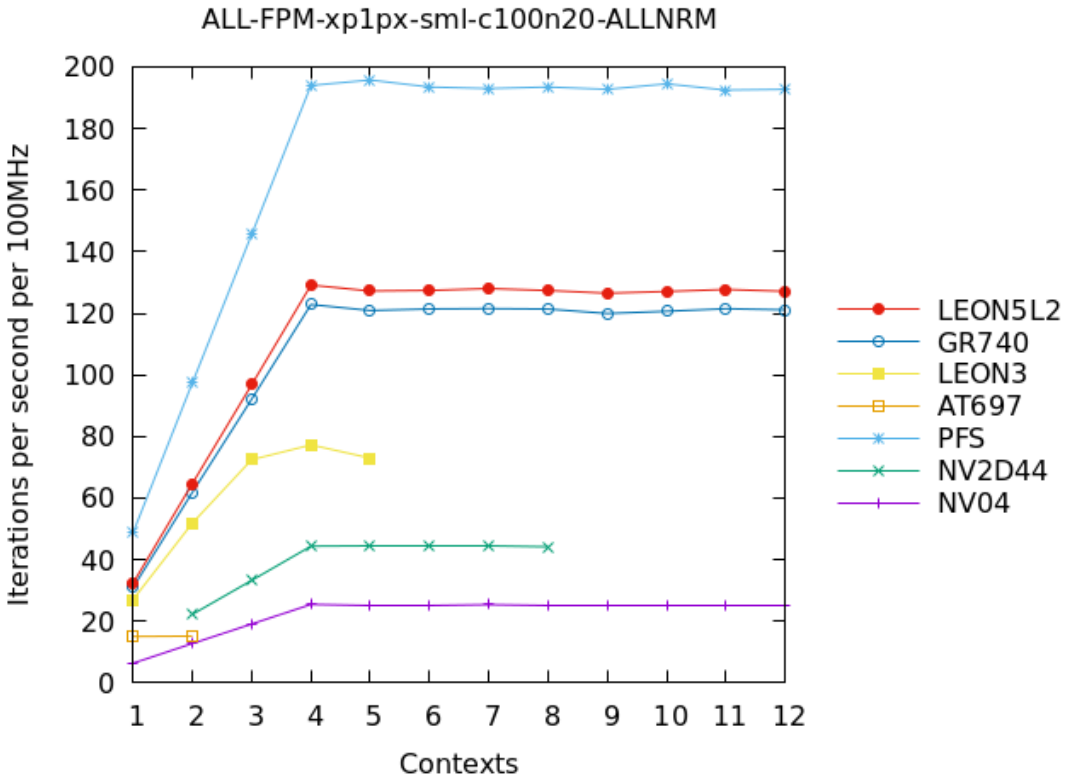

Figure 225: ALL - FPMark - xp1px-sml-c100n20, iterations per second at 100MHz. Higher values denote better performance.

10.4.2 Scaling plots

Figures 226 to 278 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.

Poor scaling can be observed for *atan-1k*, *horner-sml*, *inner-product-sml*, *loops-all-tiny*, *lu-mid*, *lu-sml* and *radix2*.

A clear memory-bound nature is demonstrated for *atan-64k*, *horner-big*, *horner-mid*, *linear_alg-mid*, *linear_alg-sml*, *nnet_data1*, *ray-64x48at4s*, *xp1px-sml*.

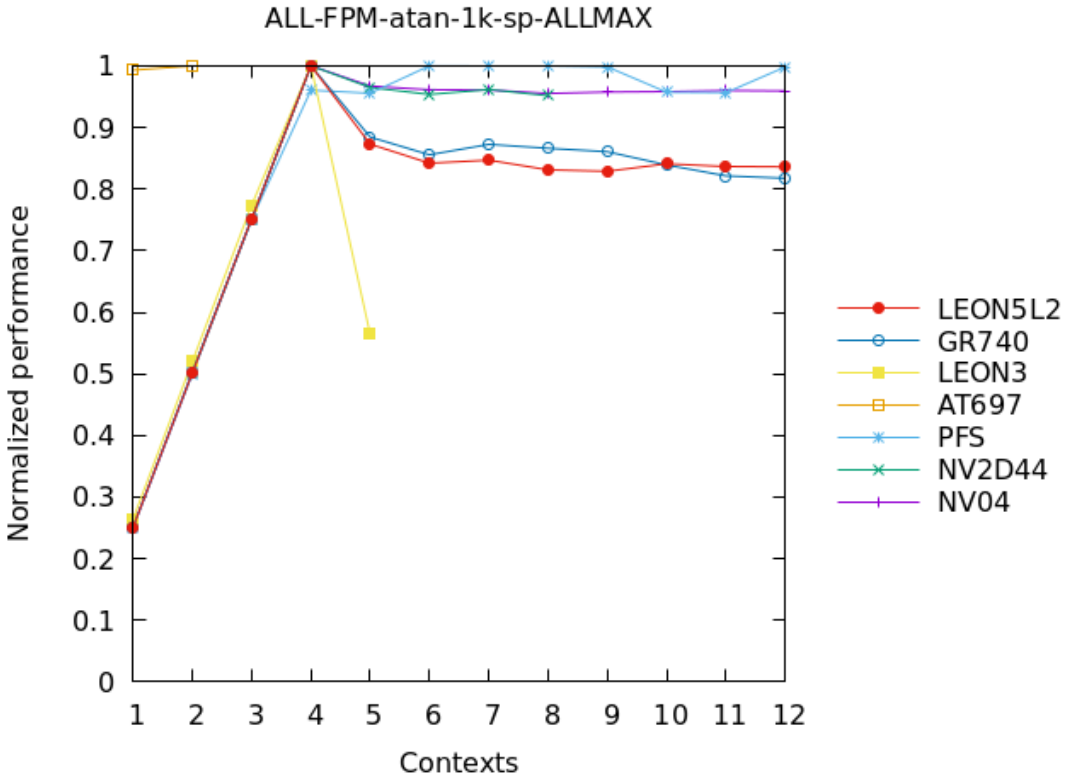

Figure 226: ALL - FPMark - atan-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

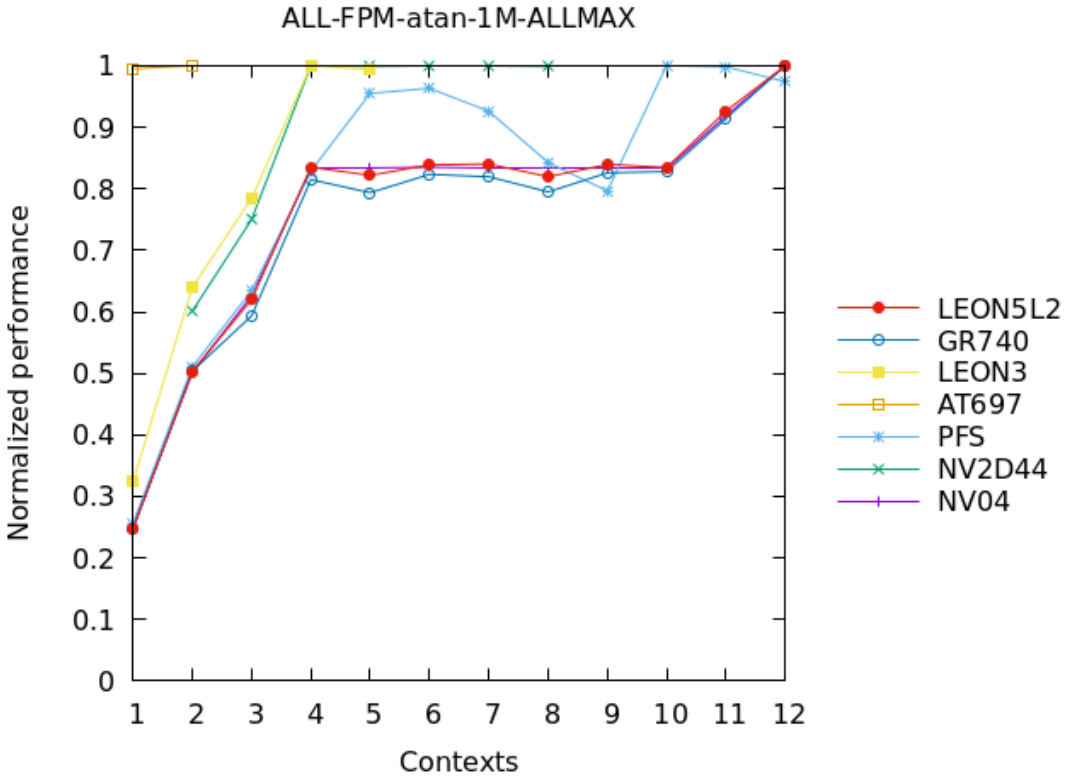

Figure 227: ALL - FPMark - atan-1M, iterations per second at 100MHz. Higher values denote better performance.

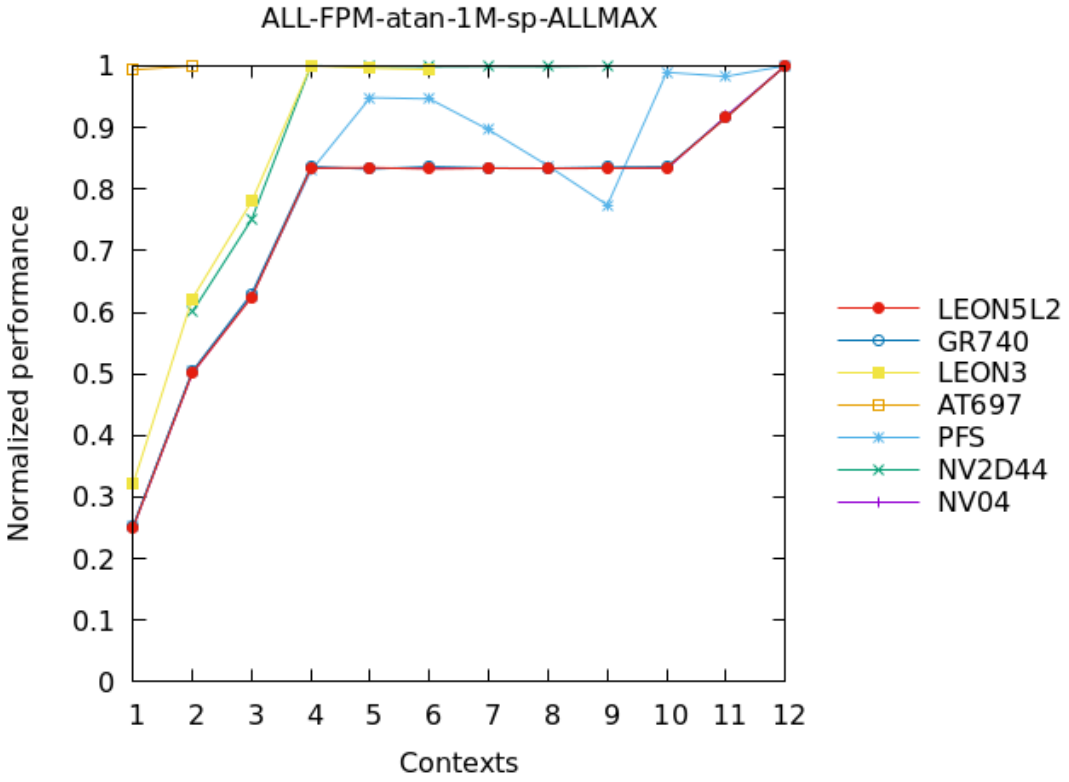

Figure 228: ALL - FPMark - atan-1M-sp, iterations per second at 100MHz. Higher values denote better performance.

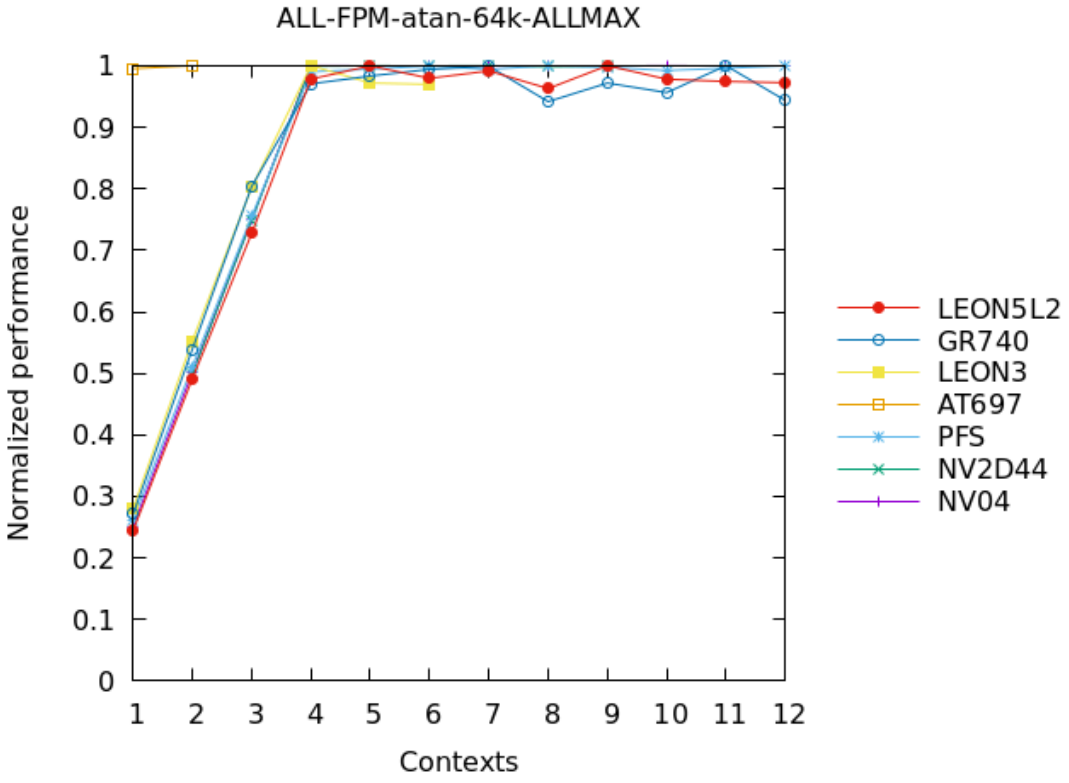

Figure 229: ALL - FPMark - atan-64k, iterations per second at 100MHz. Higher values denote better performance.

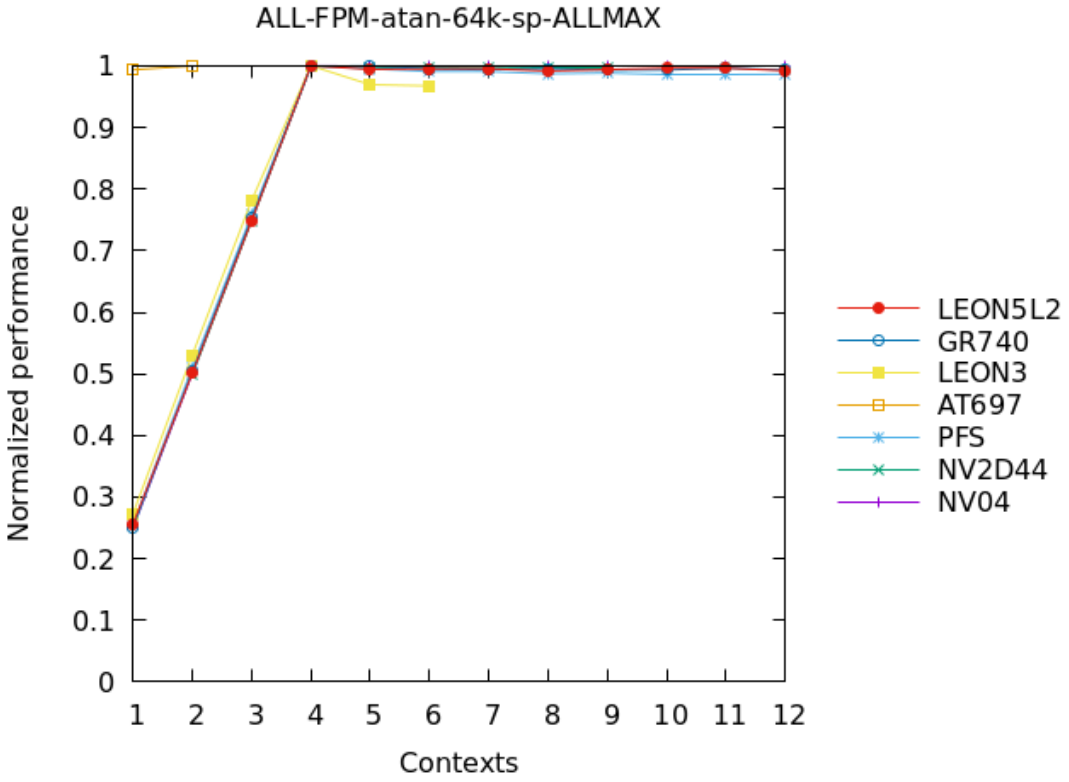

Figure 230: ALL - FPMark - atan-64k-sp, iterations per second at 100MHz. Higher values denote better performance.

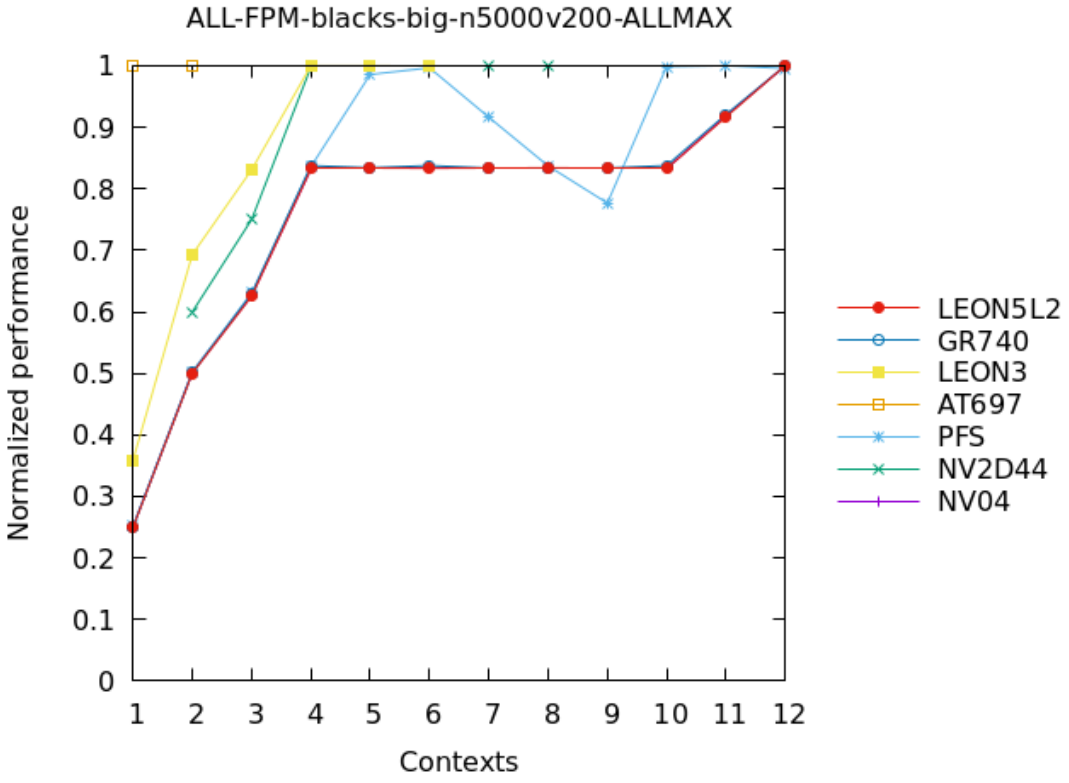

Figure 231: ALL - FPMark - blacks-big-n5000v200, iterations per second at 100MHz. Higher values denote better performance.

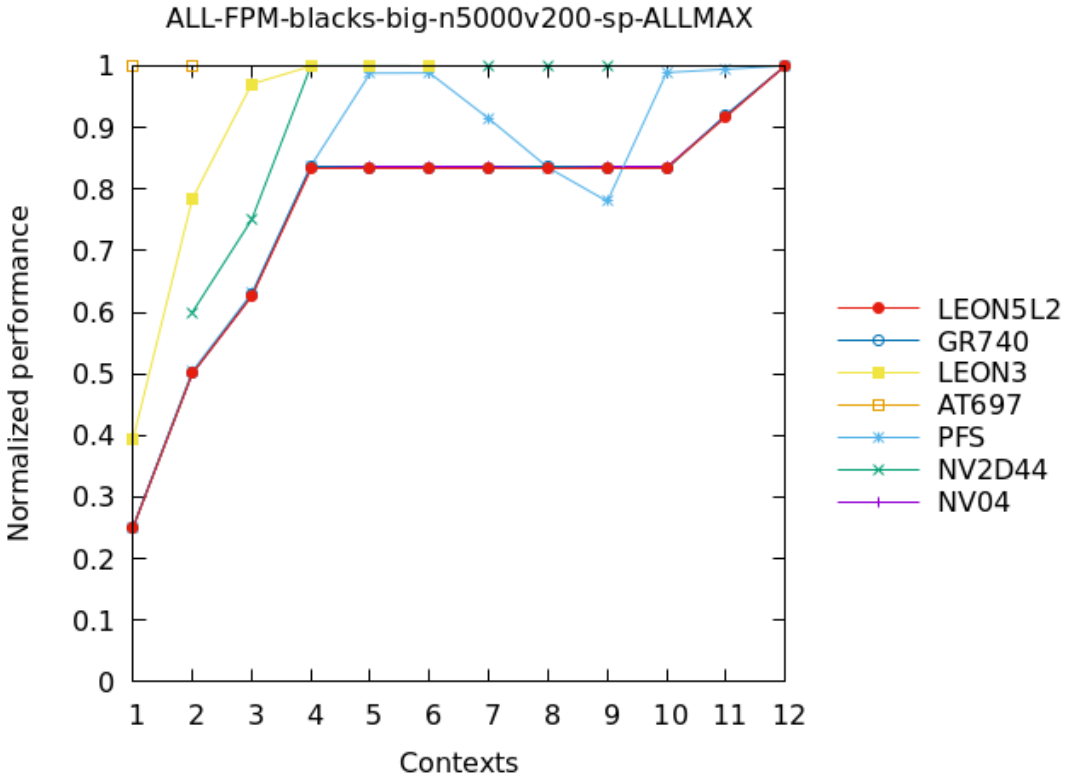

Figure 232: ALL - FPMark - blacks-big-n5000v200-sp, iterations per second at 100MHz. Higher values denote better performance.

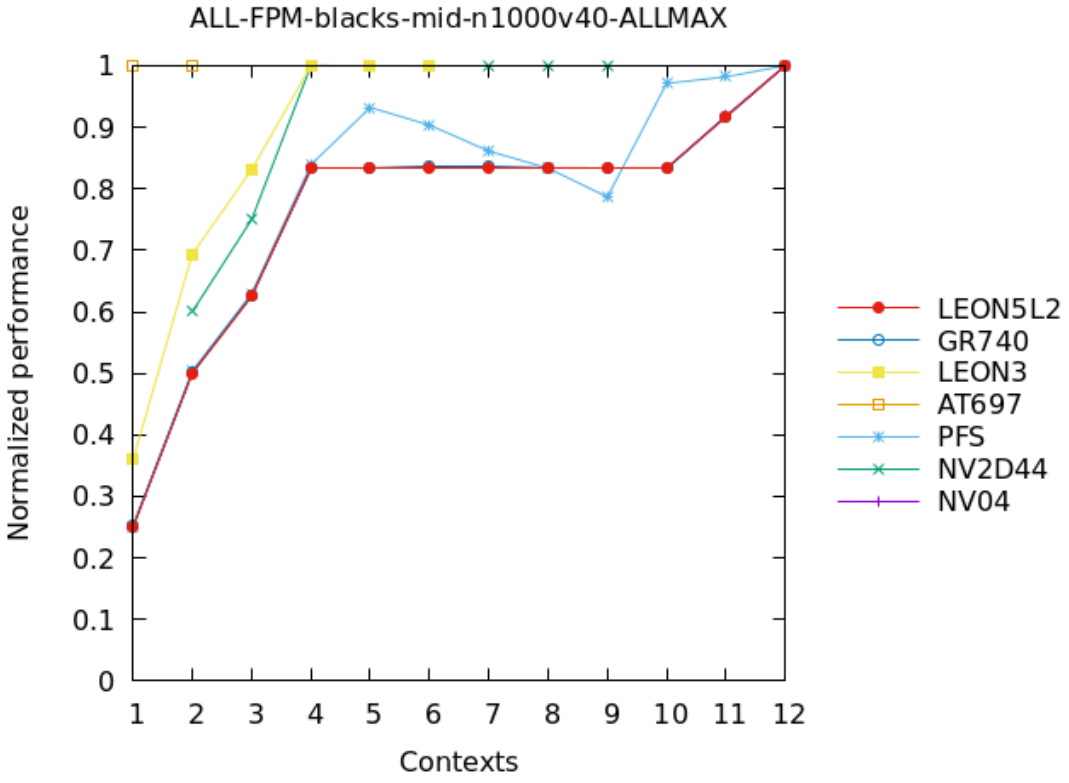

Figure 233: ALL - FPMark - blacks-mid-n1000v40, iterations per second at 100MHz. Higher values denote better performance.

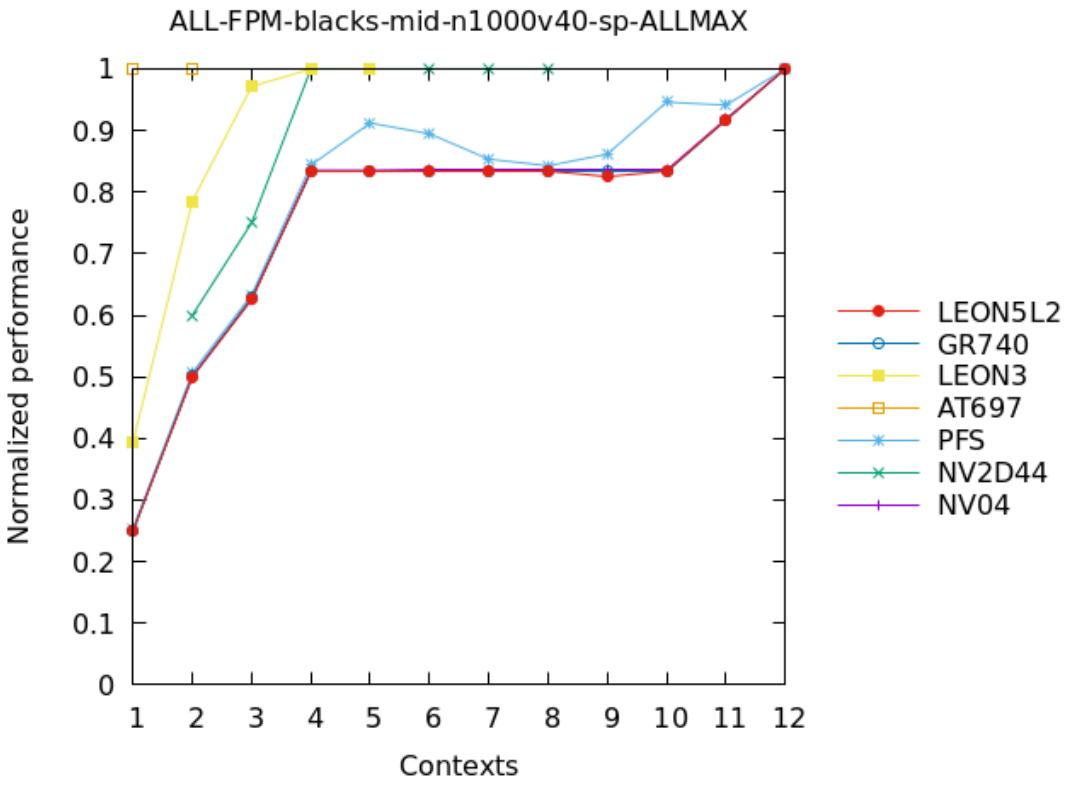

Figure 234: ALL - FPMark - blacks-mid-n1000v40-sp, iterations per second at 100MHz. Higher values denote better performance.

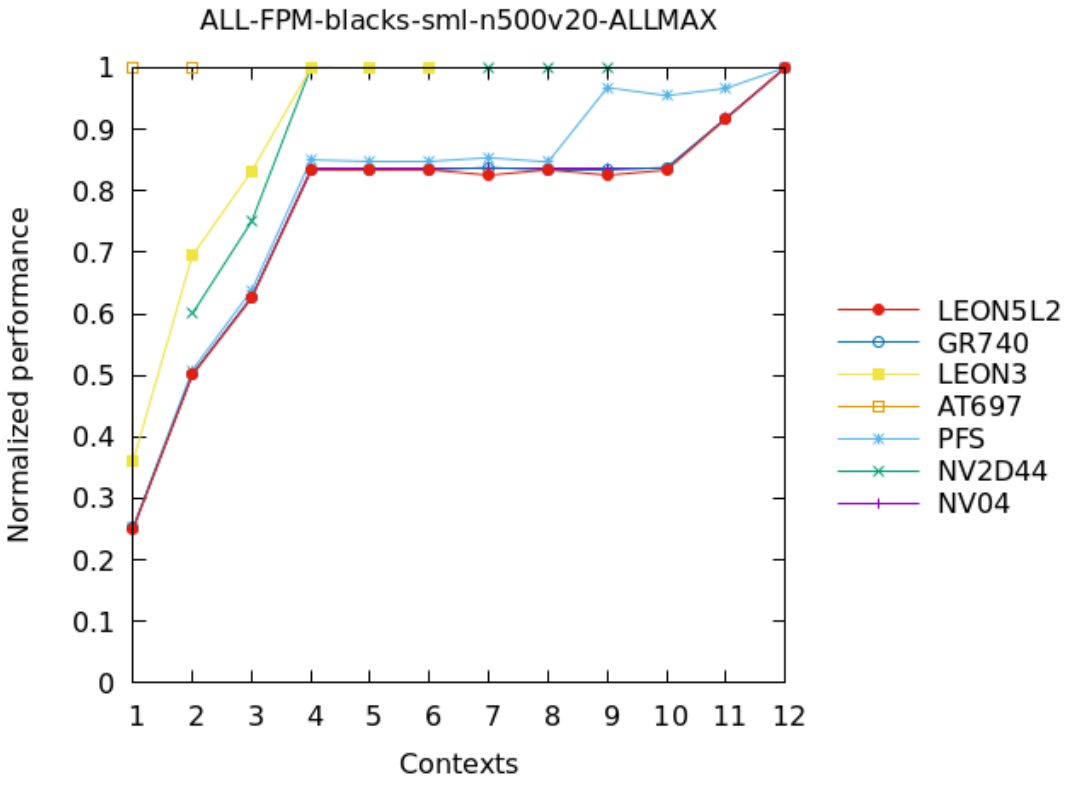

Figure 235: ALL - FPMark - blacks-sml-n500v20, iterations per second at 100MHz. Higher values denote better performance.

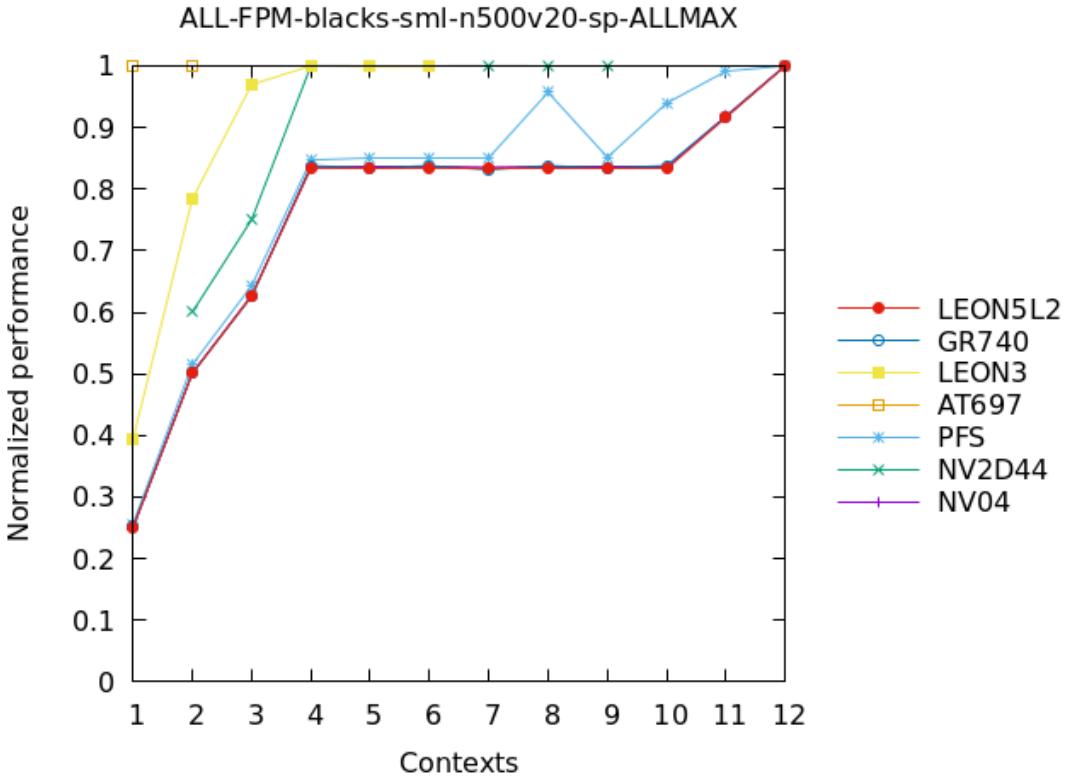

Figure 236: ALL - FPMark - blacks-sml-n500v20-sp, iterations per second at 100MHz. Higher values denote better performance.

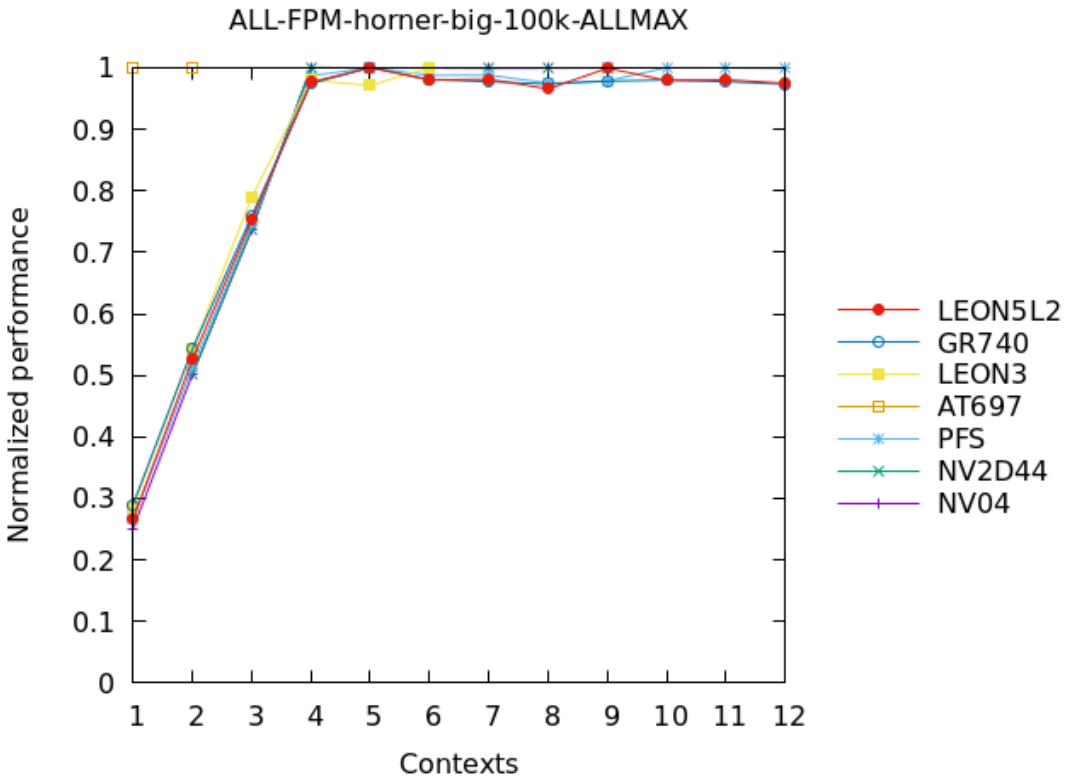

Figure 237: ALL - FPMark - horner-big-100k, iterations per second at 100MHz. Higher values denote better performance.

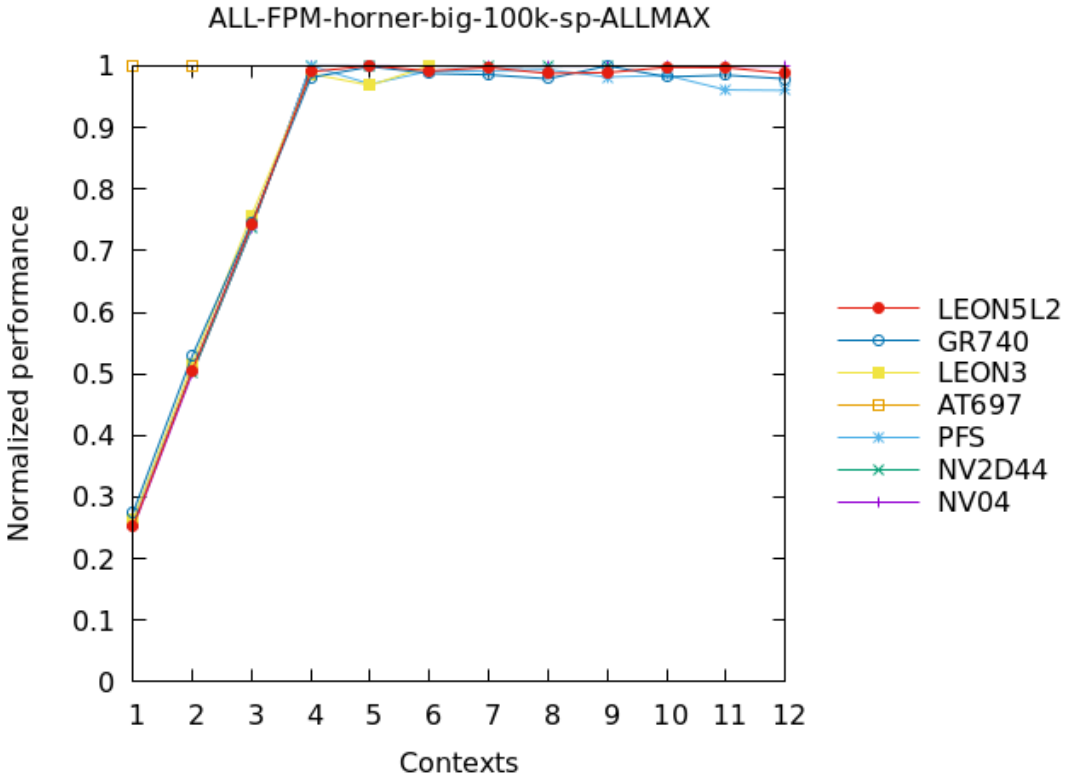

Figure 238: ALL - FPMark - horner-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

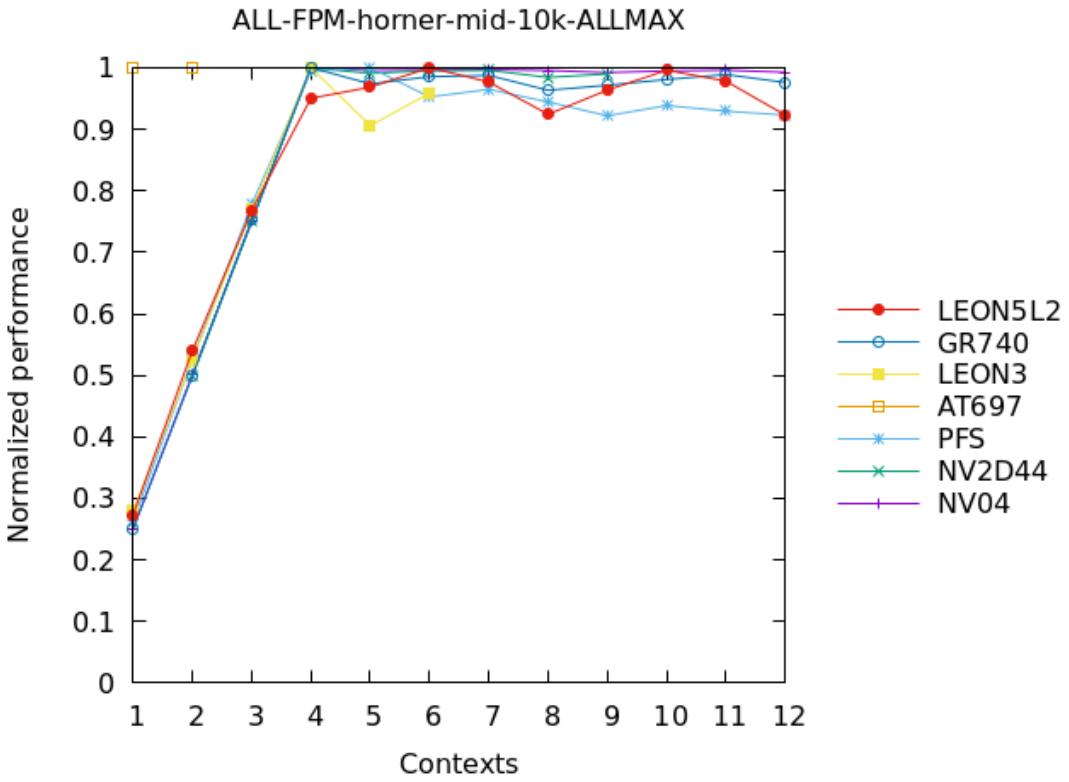

Figure 239: ALL - FPMark - horner-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

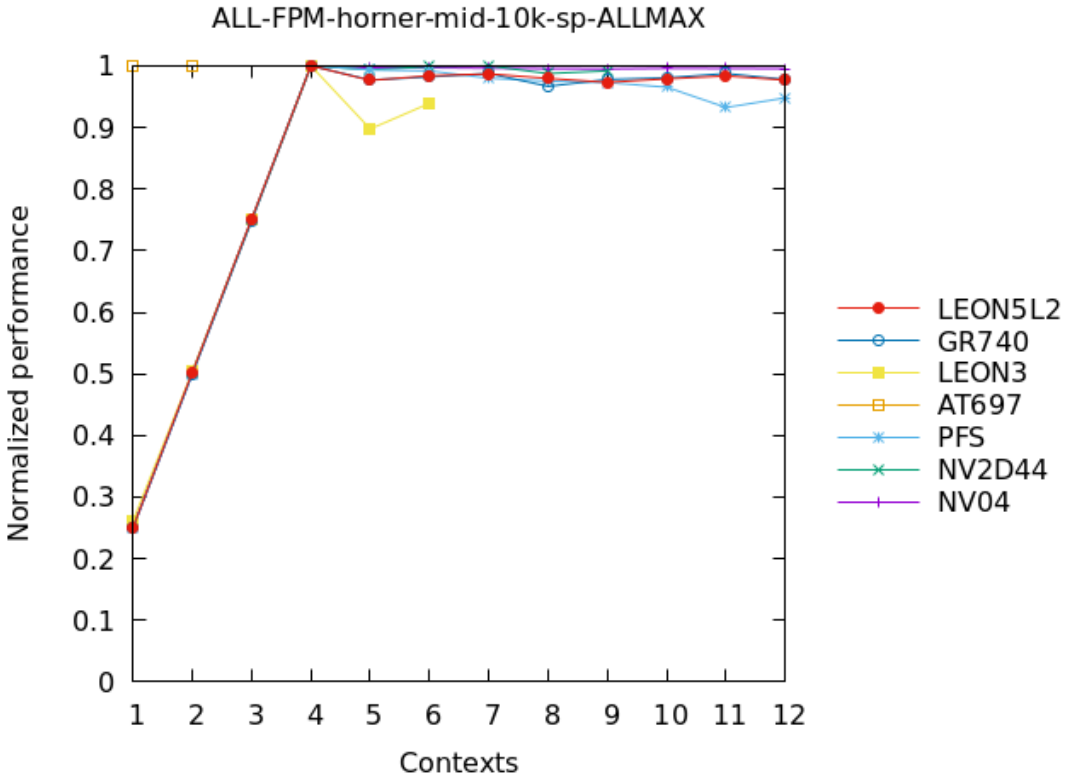

Figure 240: ALL - FPMark - horner-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

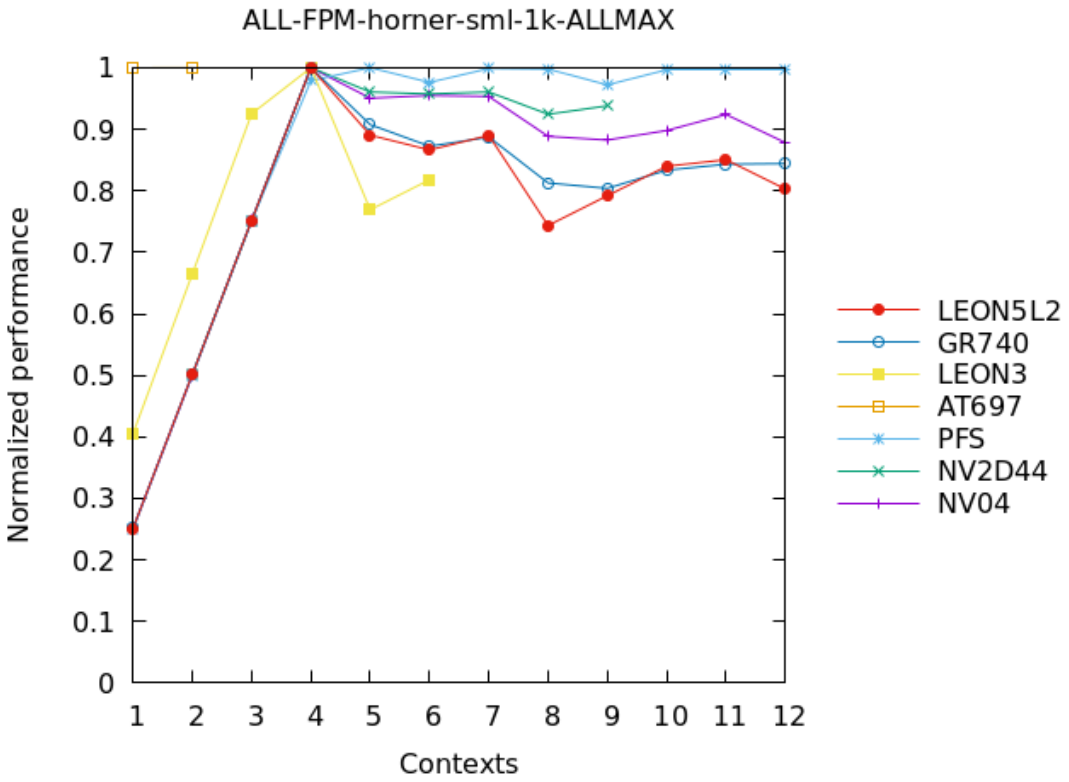

Figure 241: ALL - FPMark - horner-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

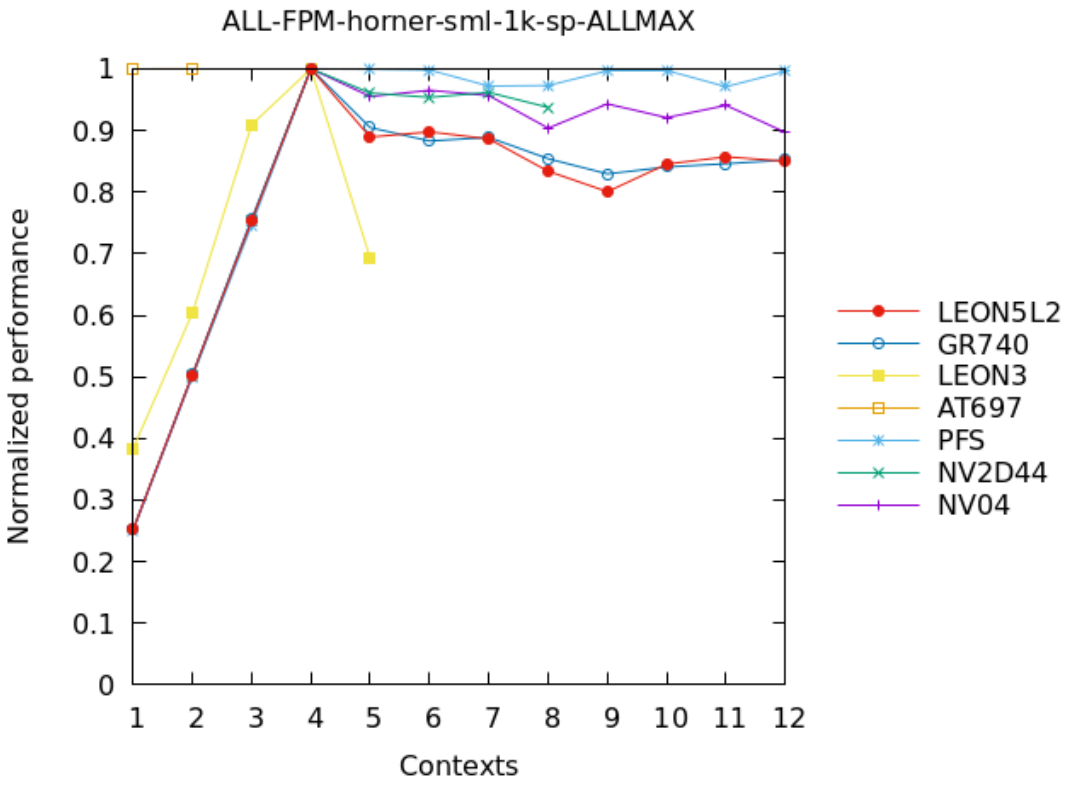

Figure 242: ALL - FPMark - horner-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

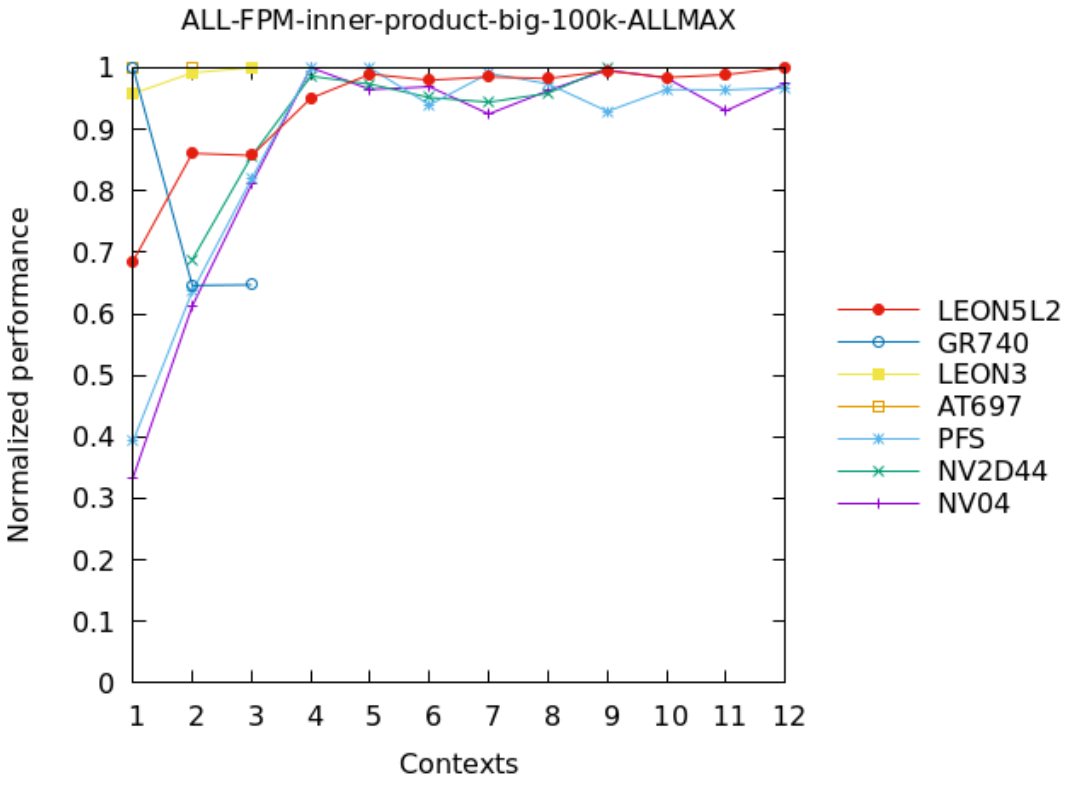

Figure 243: ALL - FPMark - inner-product-big-100k, iterations per second at 100MHz. Higher values denote better performance.

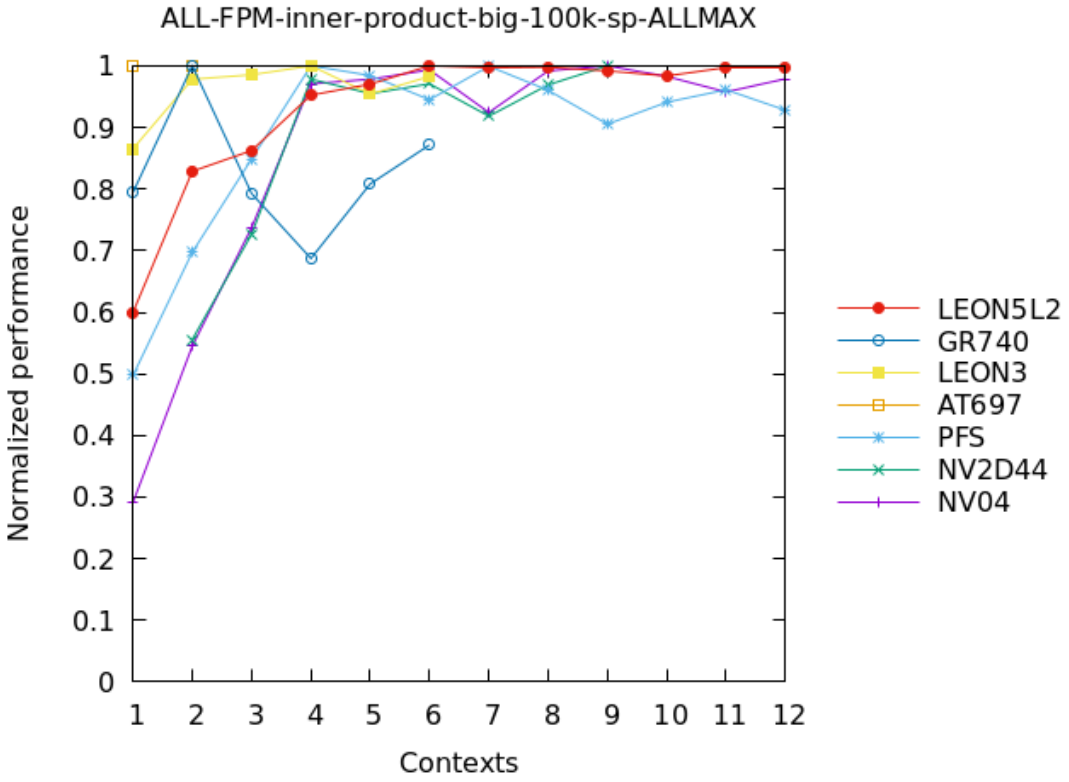

Figure 244: ALL - FPMark - inner-product-big-100k-sp, iterations per second at 100MHz. Higher values denote better performance.

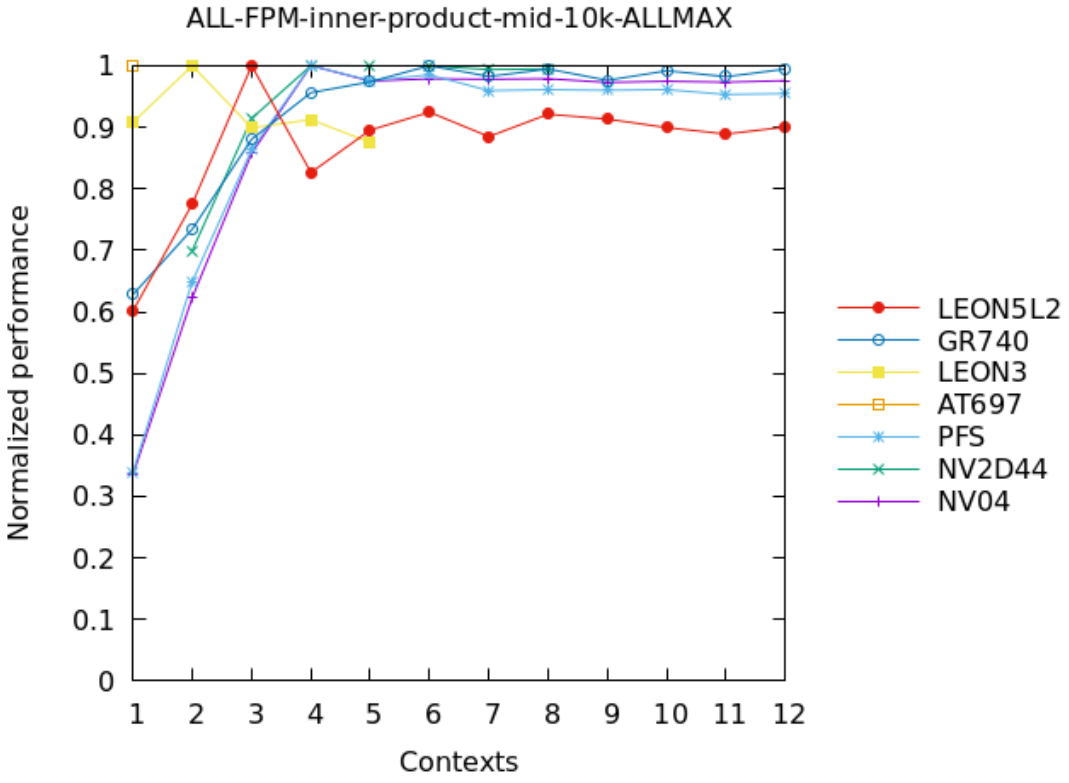

Figure 245: ALL - FPMark - inner-product-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

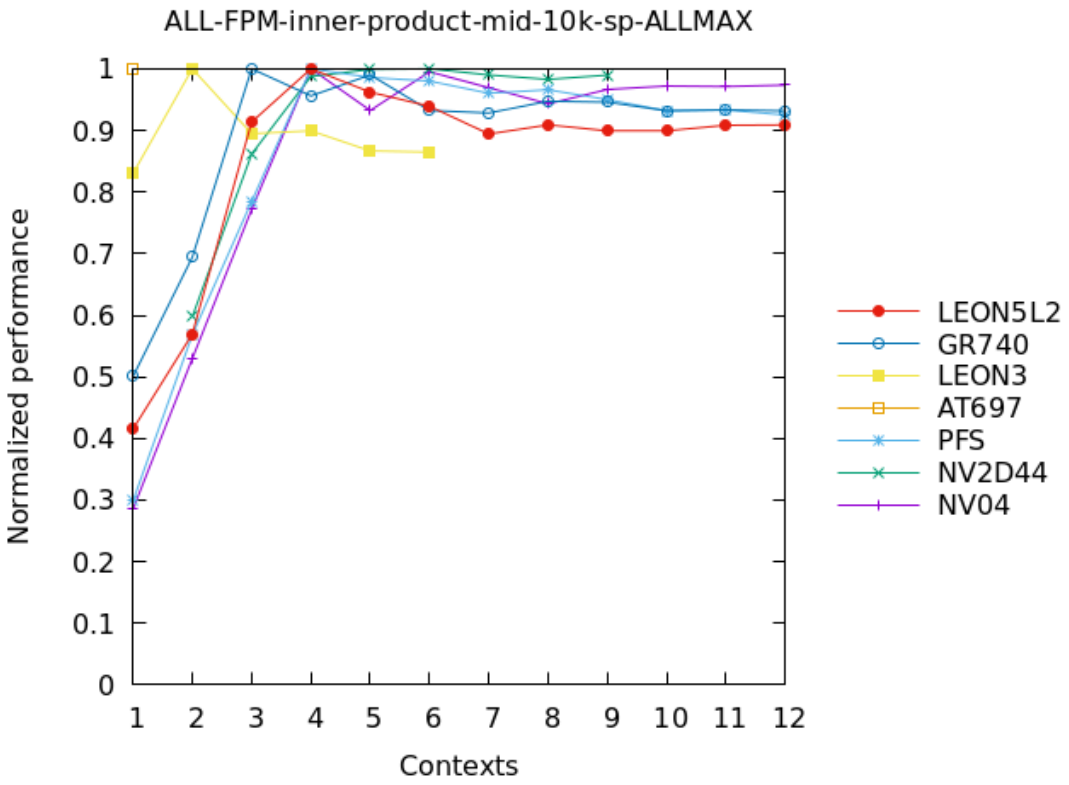

Figure 246: ALL - FPMark - inner-product-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

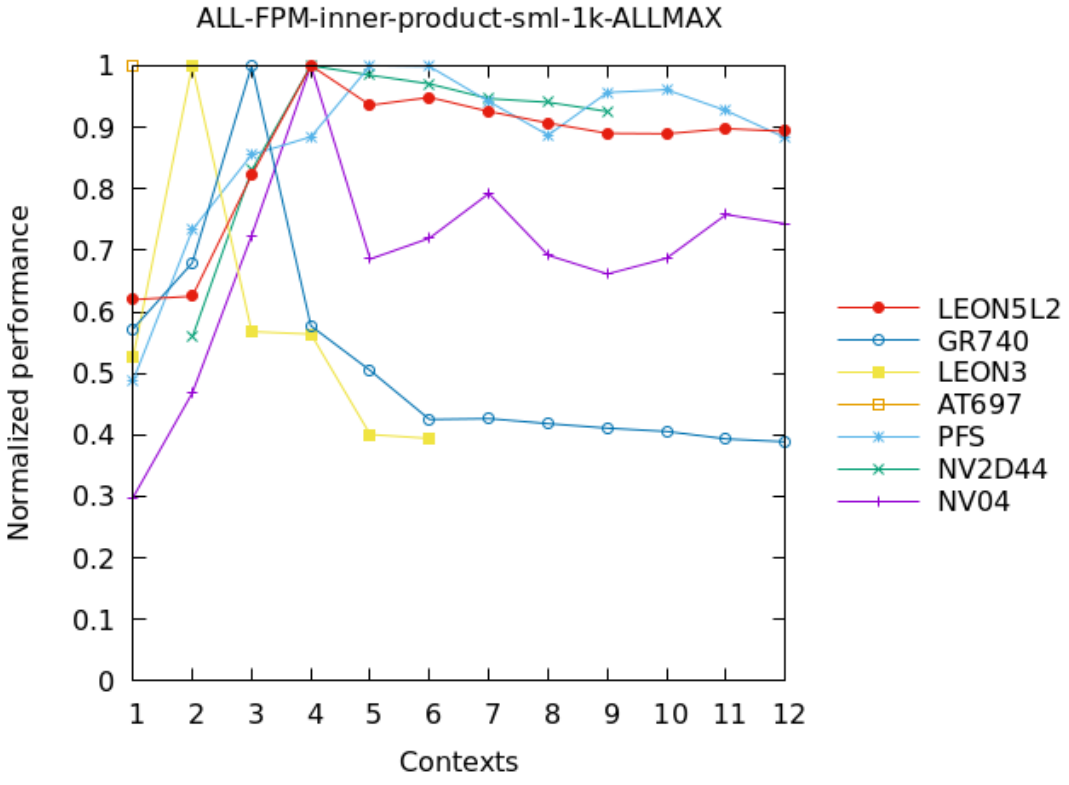

Figure 247: ALL - FPMark - inner-product-sml-1k, iterations per second at 100MHz. Higher values denote better performance.

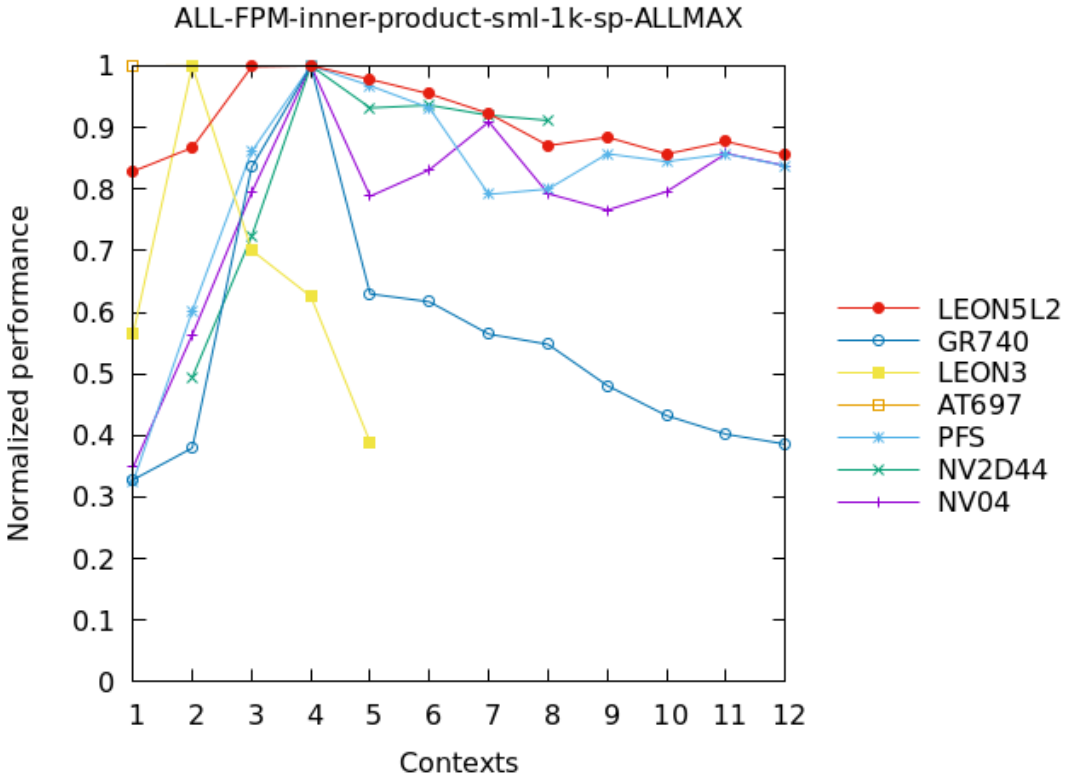

Figure 248: ALL - FPMark - inner-product-sml-1k-sp, iterations per second at 100MHz. Higher values denote better performance.

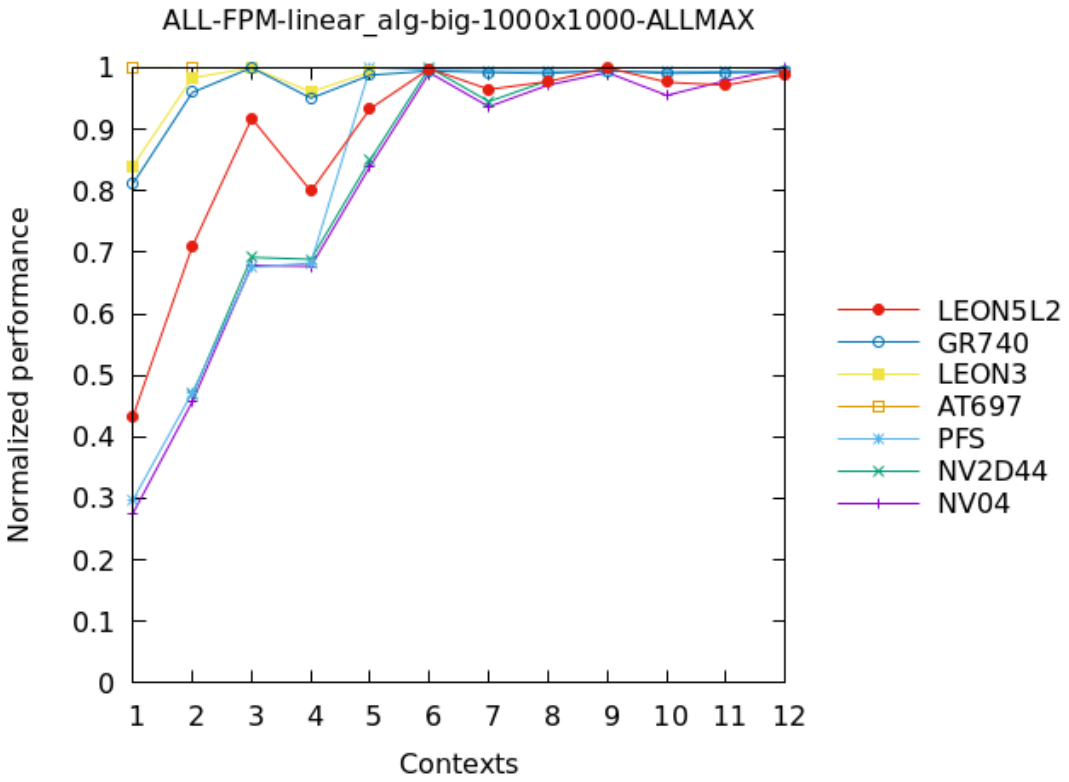

Figure 249: ALL - FPMark - linear_alg-big-1000x1000, iterations per second at 100MHz. Higher values denote better performance.

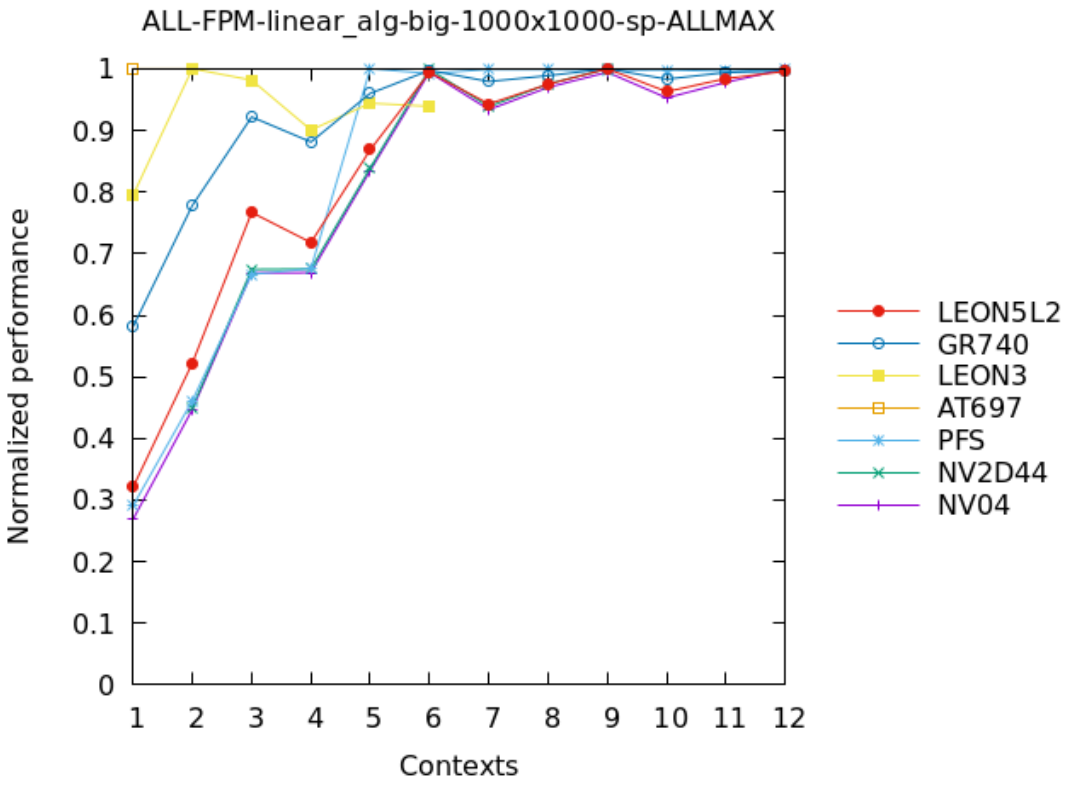

Figure 250: ALL - FPMark - linear_alg-big-1000x1000-sp, iterations per second at 100MHz. Higher values denote better performance.

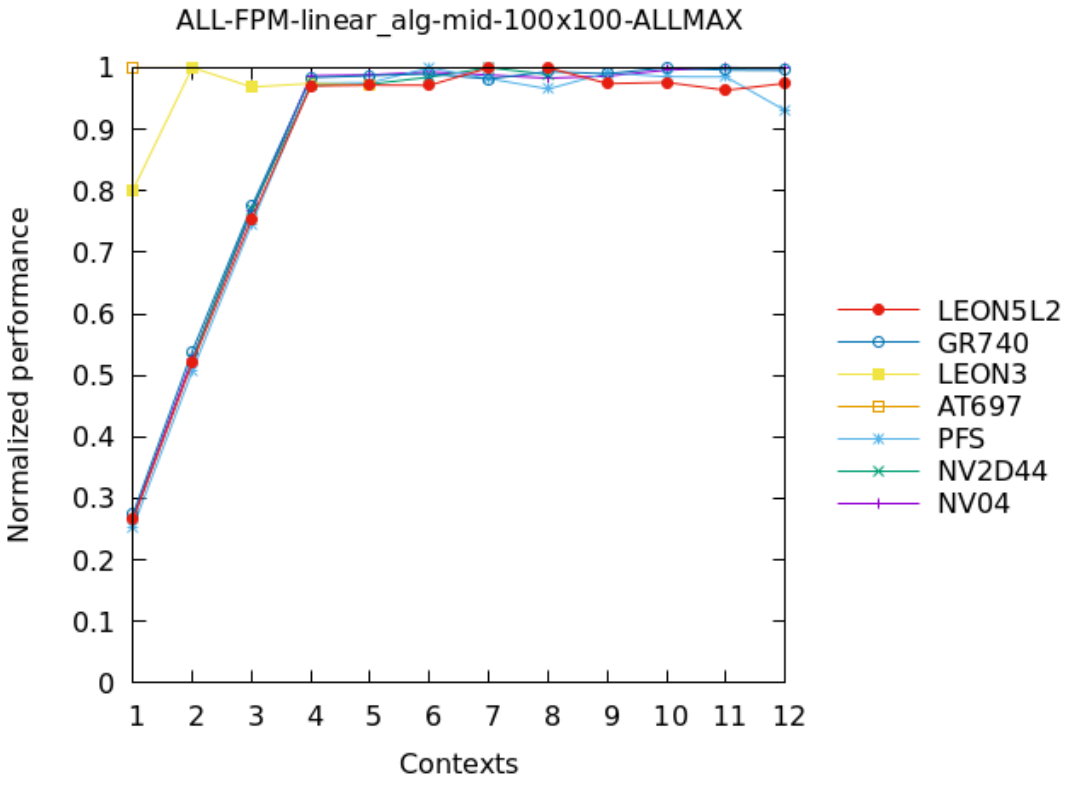

Figure 251: ALL - FPMark - linear_alg-mid-100x100, iterations per second at 100MHz. Higher values denote better performance.

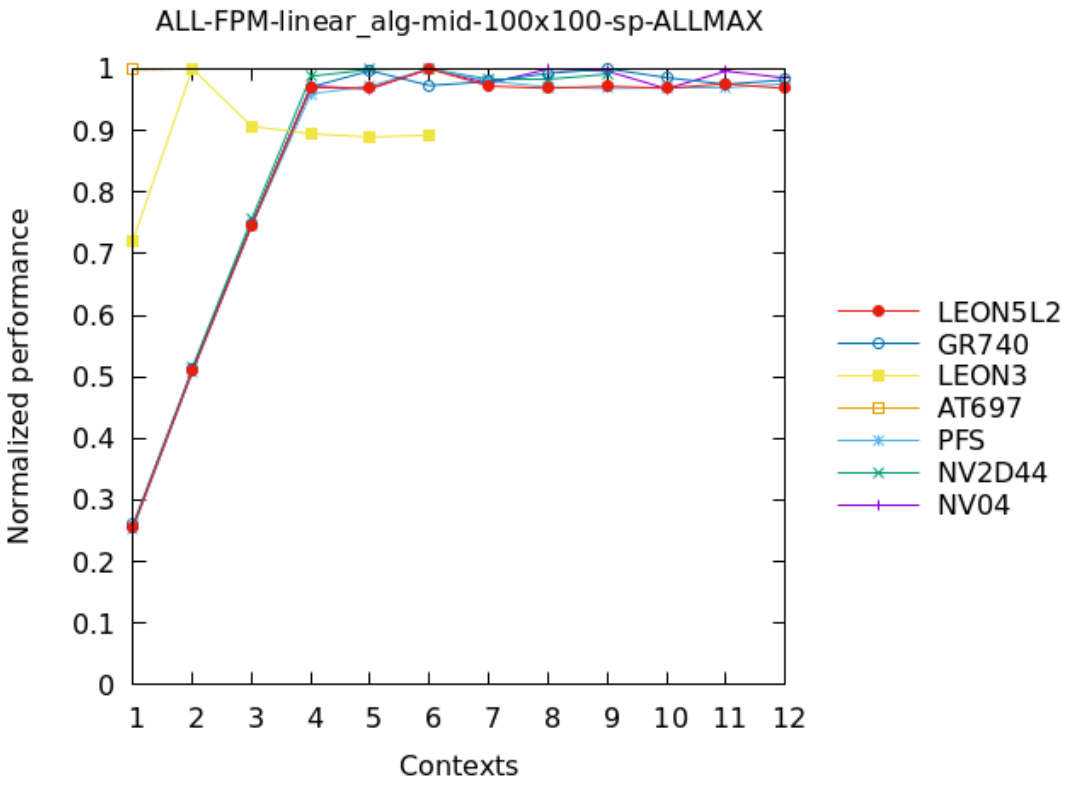

Figure 252: ALL - FPMark - linear_alg-mid-100x100-sp, iterations per second at 100MHz. Higher values denote better performance.

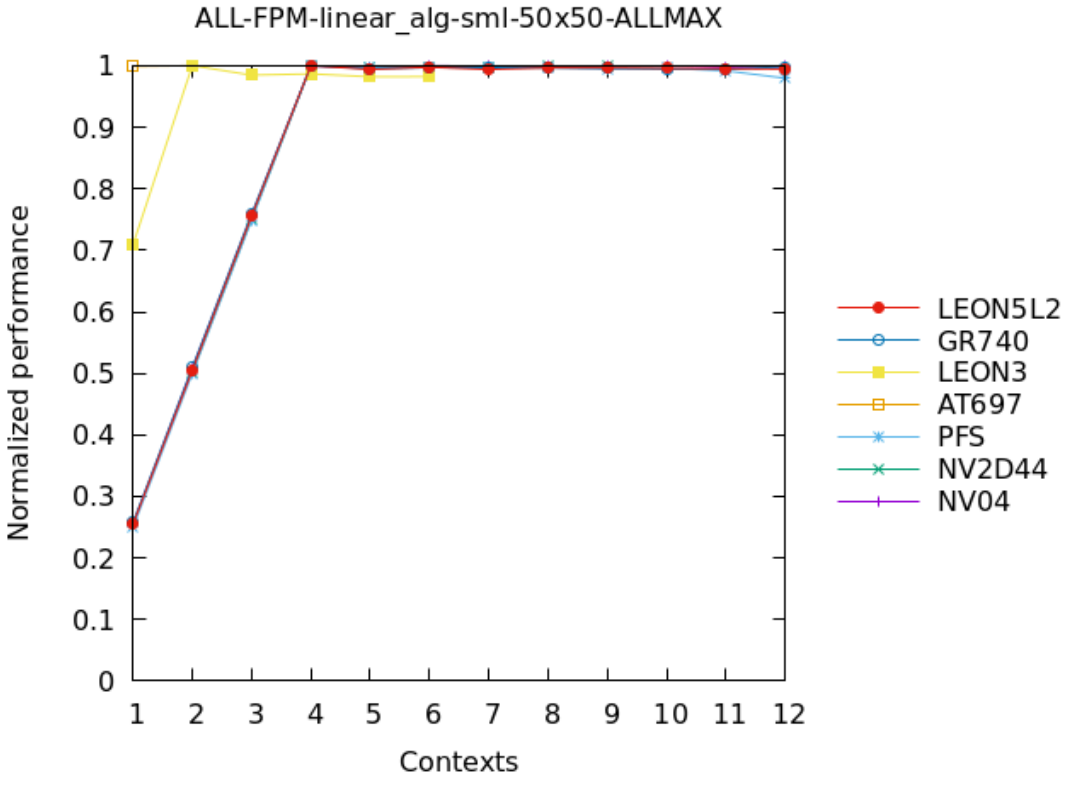

Figure 253: ALL - FPMark - linear_alg-sml-50x50, iterations per second at 100MHz. Higher values denote better performance.

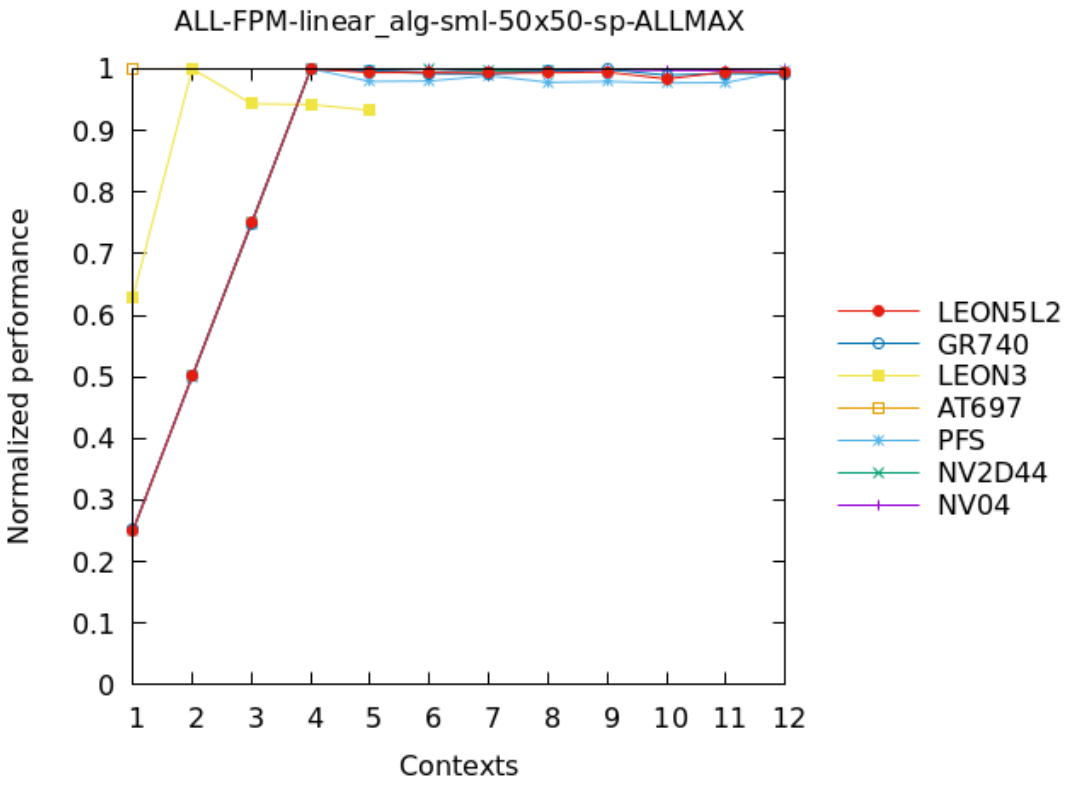

Figure 254: ALL - FPMark - linear_alg-sml-50x50-sp, iterations per second at 100MHz. Higher values denote better performance.

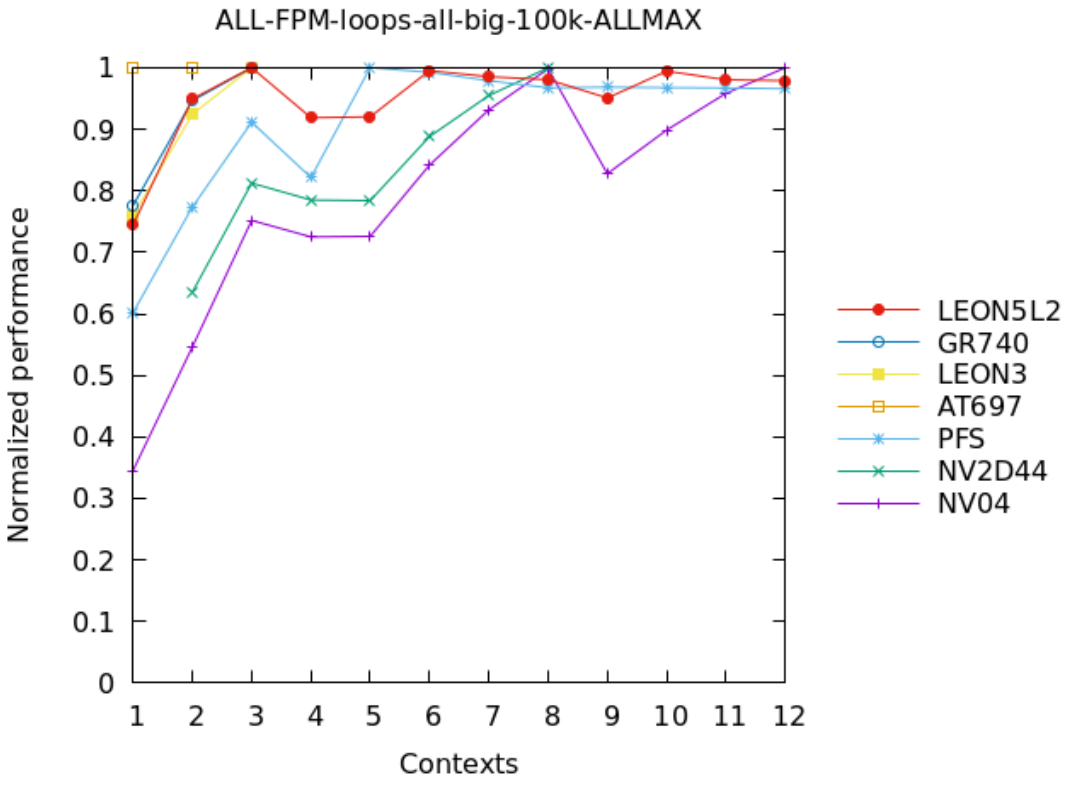

Figure 255: ALL - FPMark - loops-all-big-100k, iterations per second at 100MHz. Higher values denote better performance.

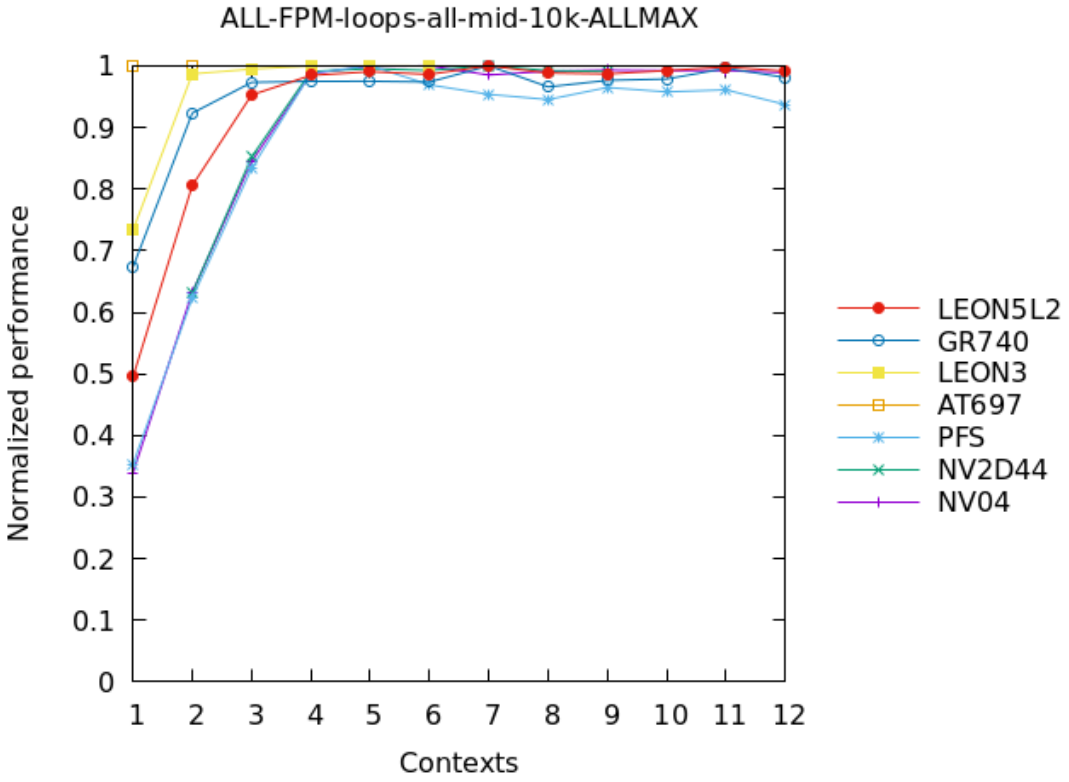

Figure 256: ALL - FPMark - loops-all-mid-10k, iterations per second at 100MHz. Higher values denote better performance.

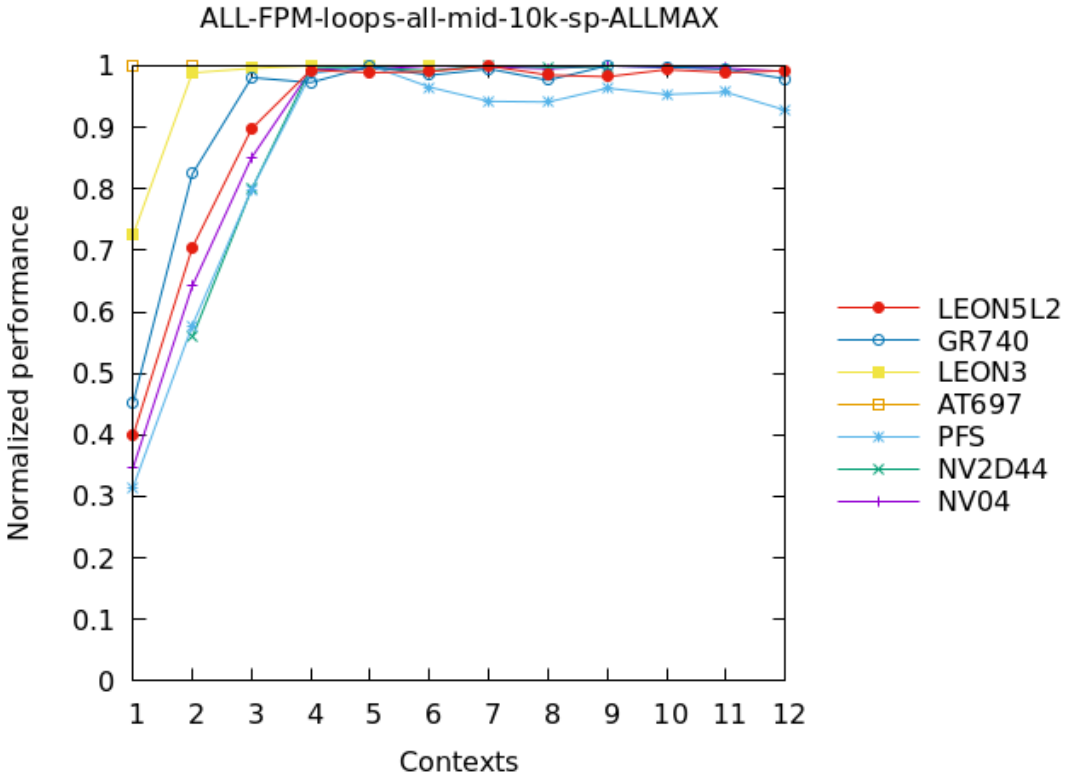

Figure 257: ALL - FPMark - loops-all-mid-10k-sp, iterations per second at 100MHz. Higher values denote better performance.

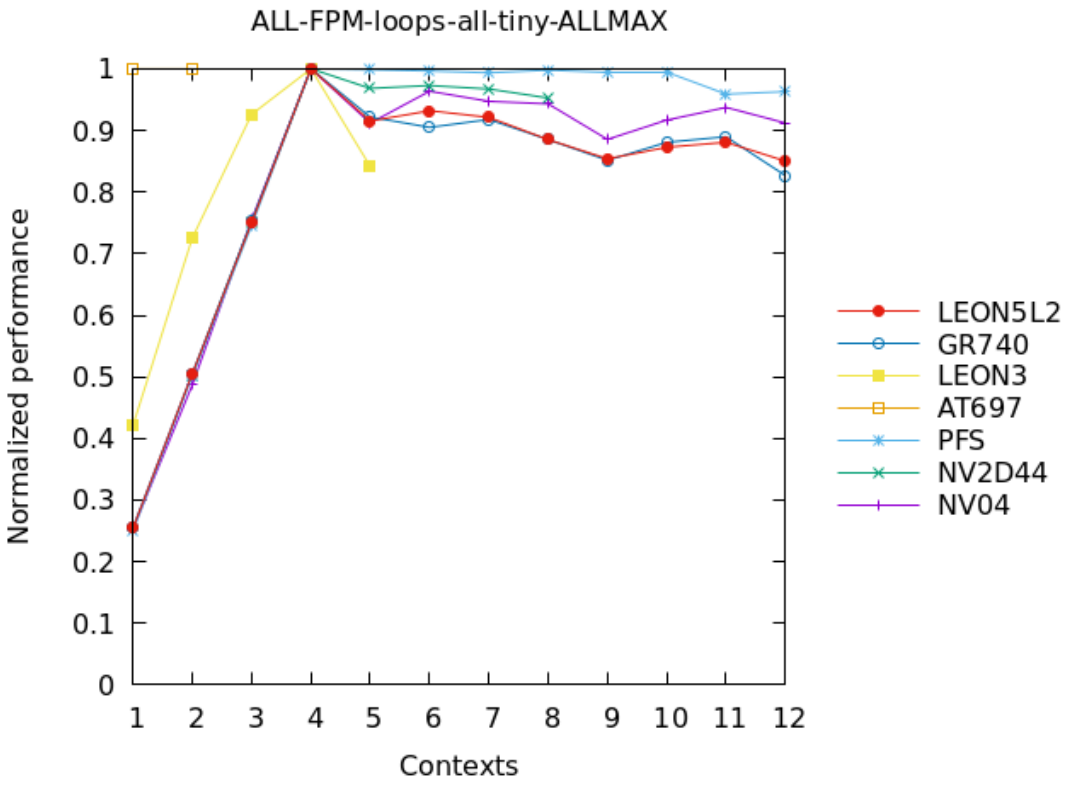

Figure 258: ALL - FPMark - loops-all-tiny, iterations per second at 100MHz. Higher values denote better performance.

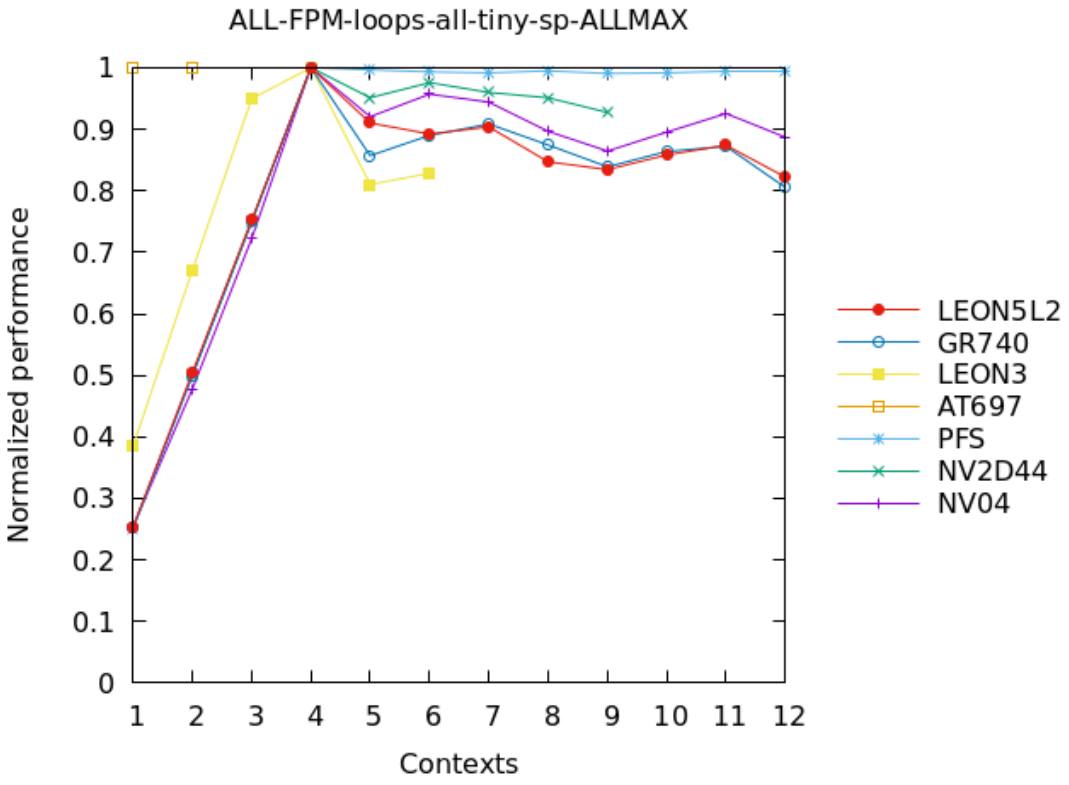

Figure 259: ALL - FPMark - loops-all-tiny-sp, iterations per second at 100MHz. Higher values denote better performance.

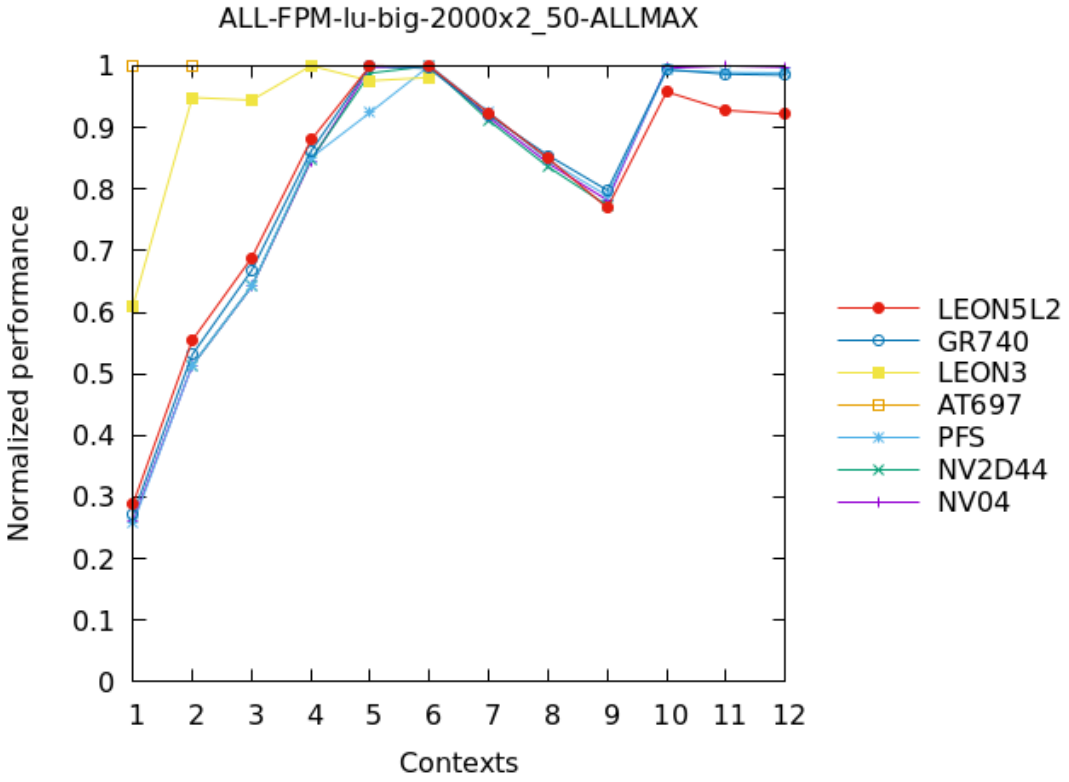

Figure 260: ALL - FPMark - lu-big-2000x2_50, iterations per second at 100MHz. Higher values denote better performance.

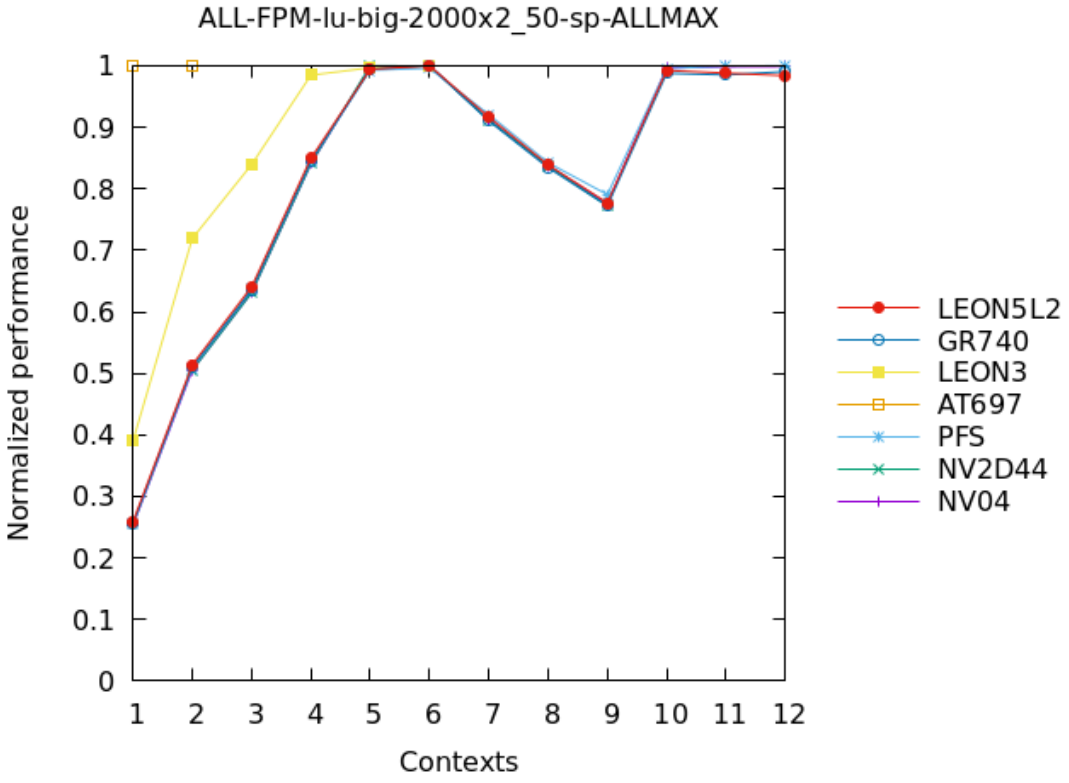

Figure 261: ALL - FPMark - lu-big-2000x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

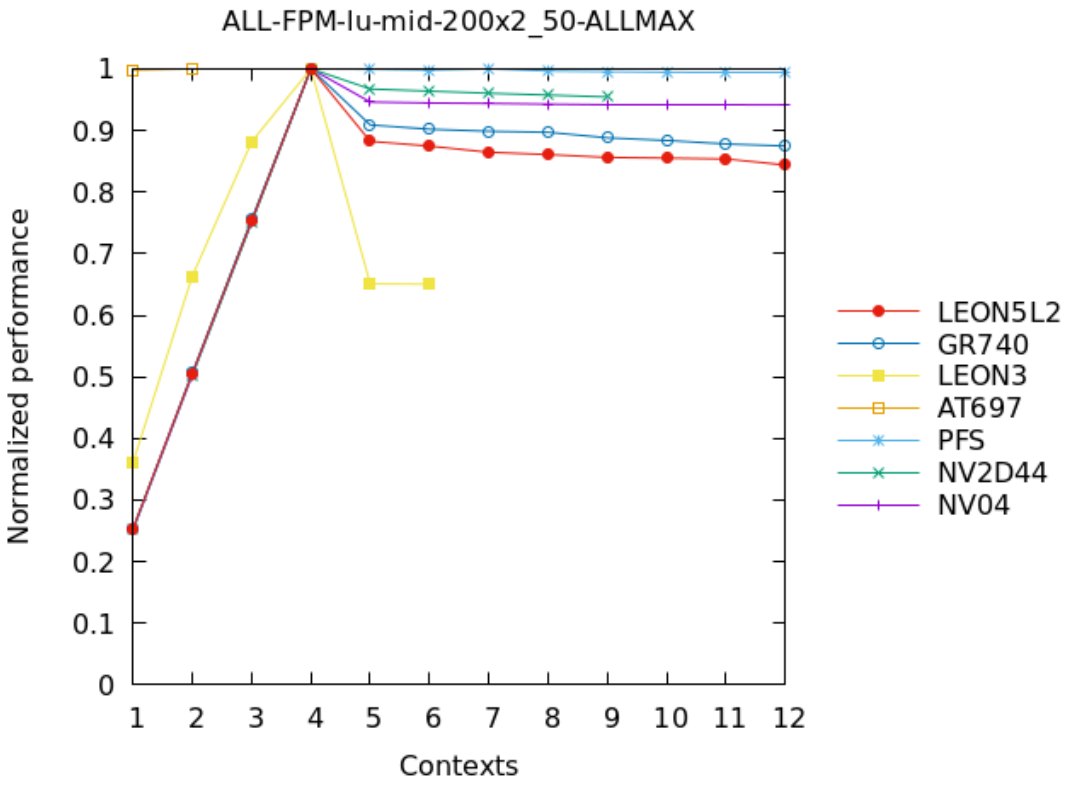

Figure 262: ALL - FPMark - lu-mid-200x2_50, iterations per second at 100MHz. Higher values denote better performance.

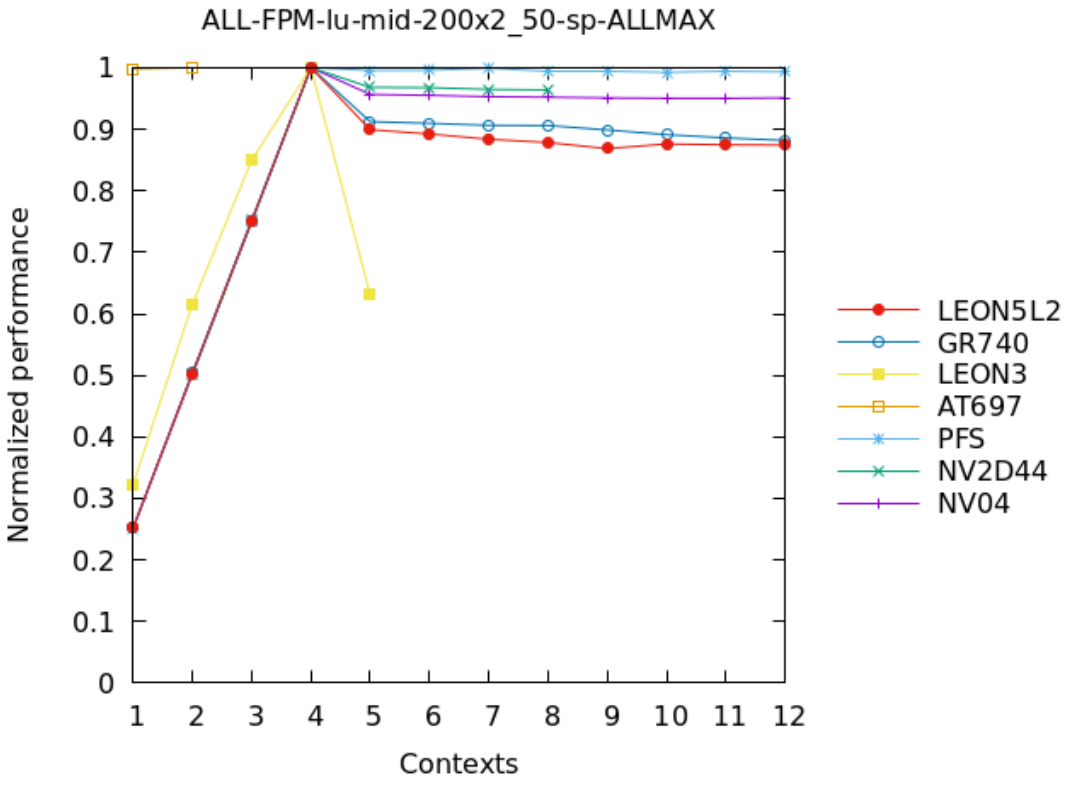

Figure 263: ALL - FPMark - lu-mid-200x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

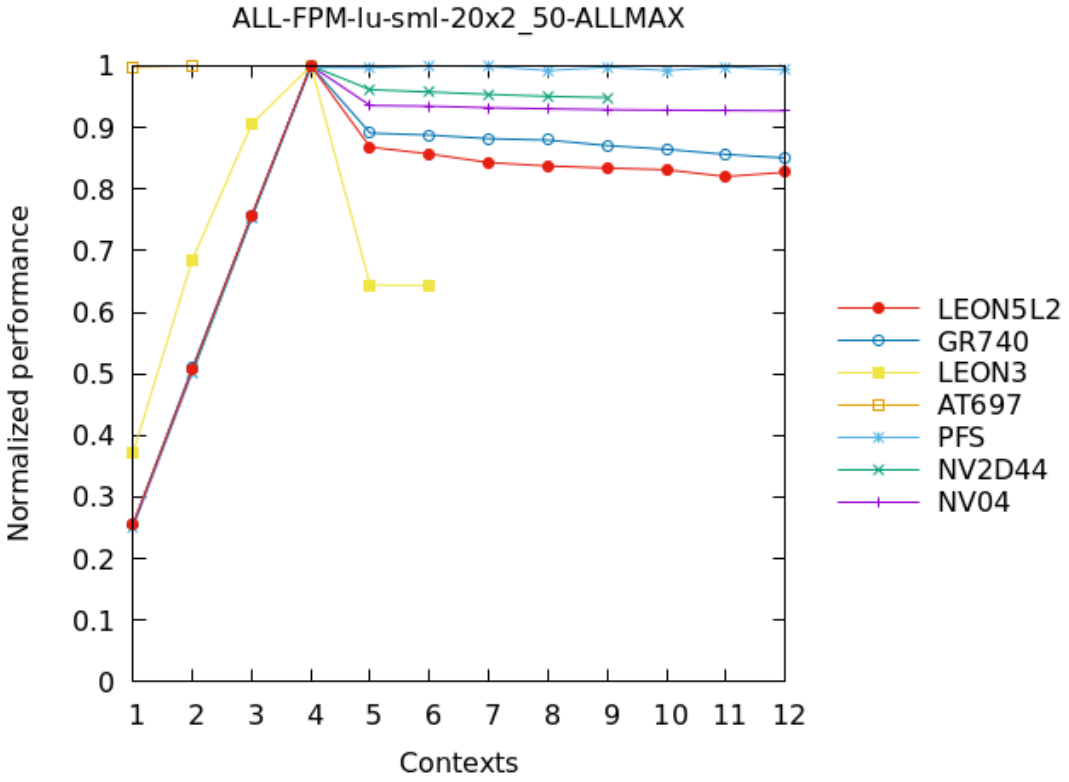

Figure 264: ALL - FPMark - lu-sml-20x2_50, iterations per second at 100MHz. Higher values denote better performance.

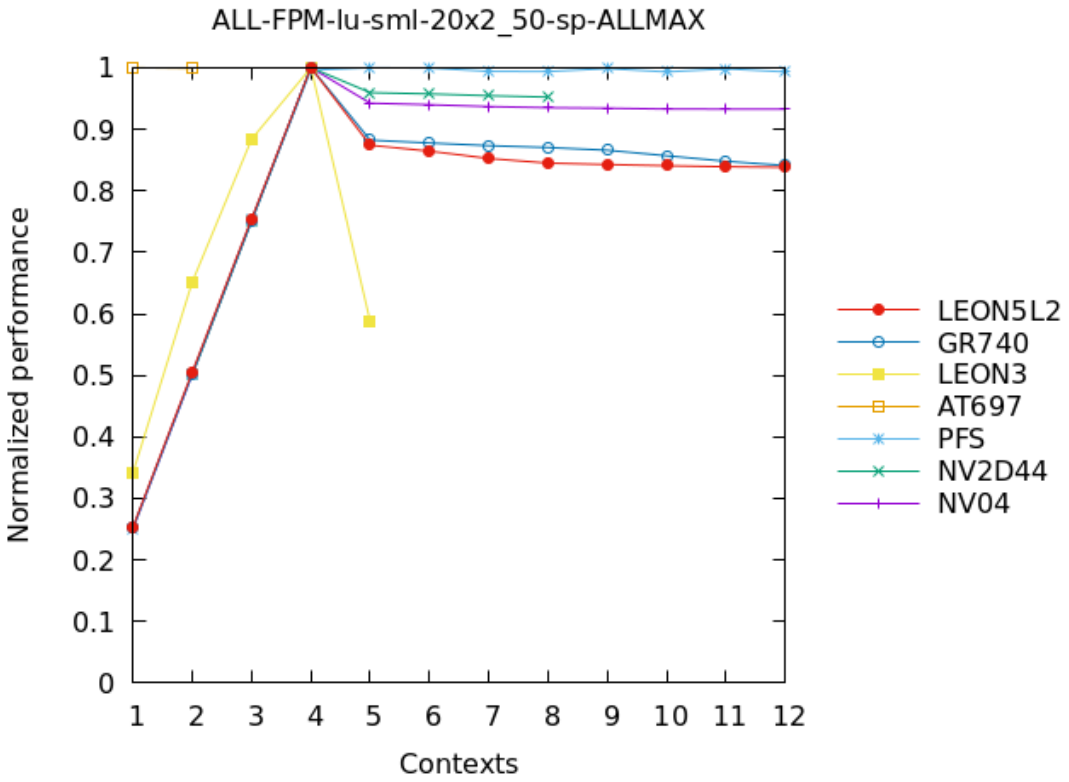

Figure 265: ALL - FPMark - lu-sml-20x2_50-sp, iterations per second at 100MHz. Higher values denote better performance.

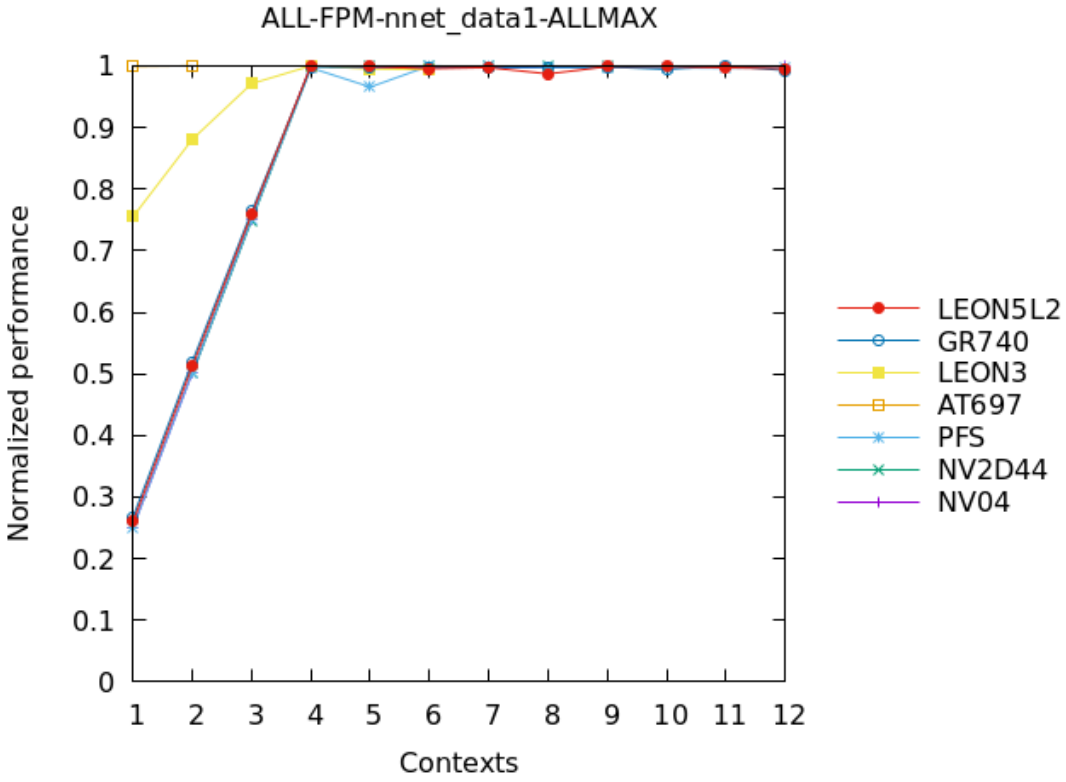

Figure 266: ALL - FPMark - nnet_data1, iterations per second at 100MHz. Higher values denote better performance.

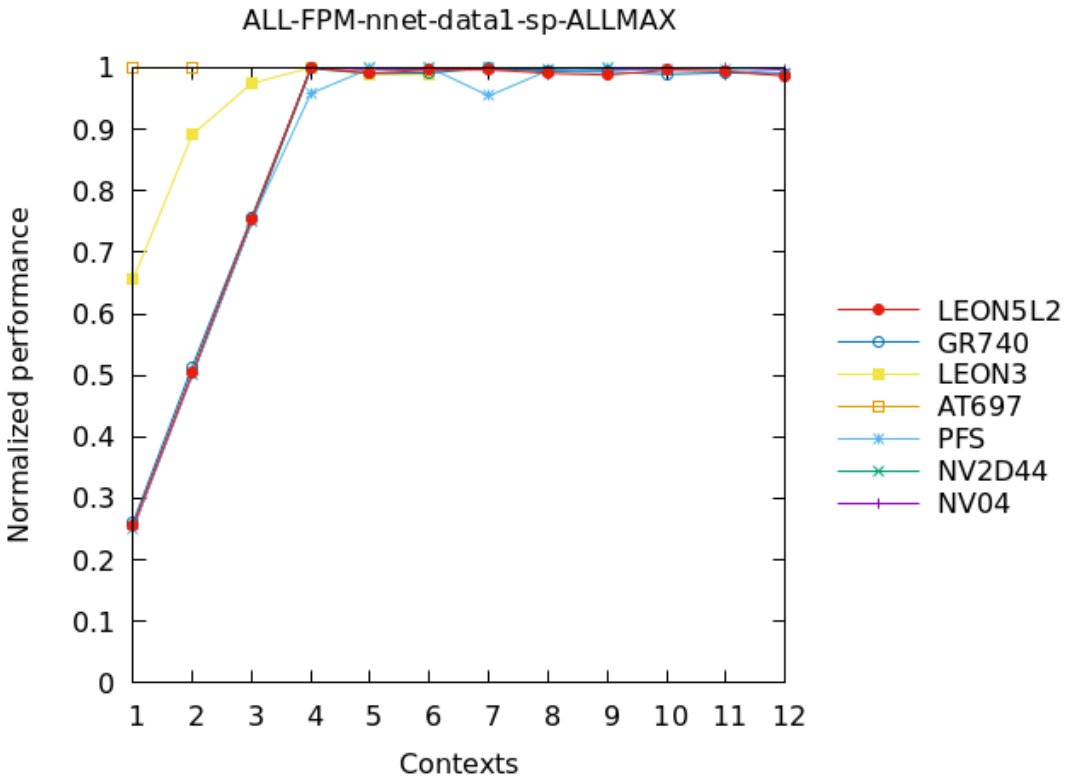

Figure 267: ALL - FPMark - nnet-data1-sp, iterations per second at 100MHz. Higher values denote better performance.

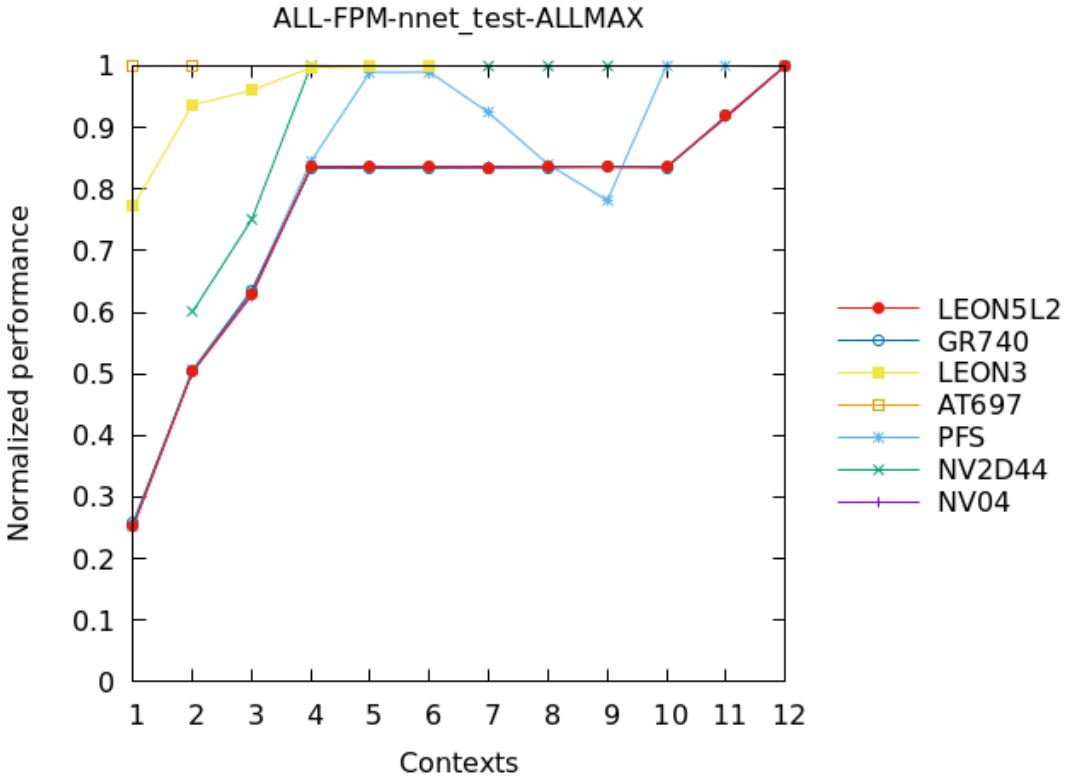

Figure 268: ALL - FPMark - nnet_test, iterations per second at 100MHz. Higher values denote better performance.

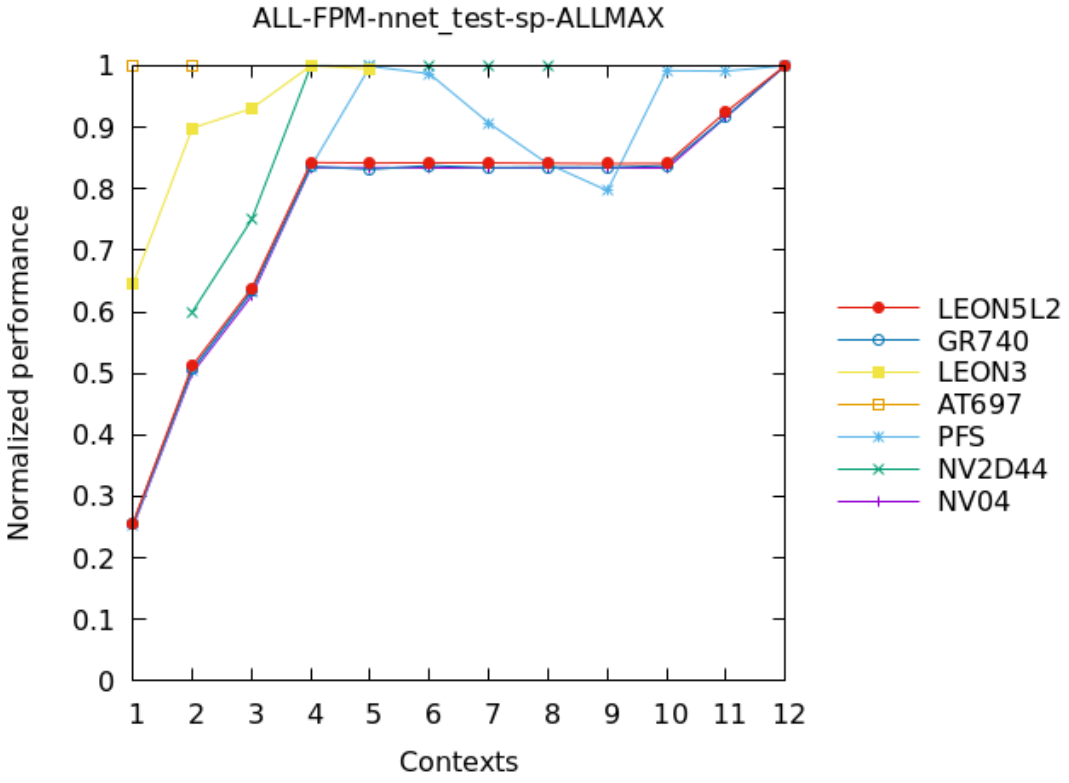

Figure 269: ALL - FPMark - nnet_test-sp, iterations per second at 100MHz. Higher values denote better performance.

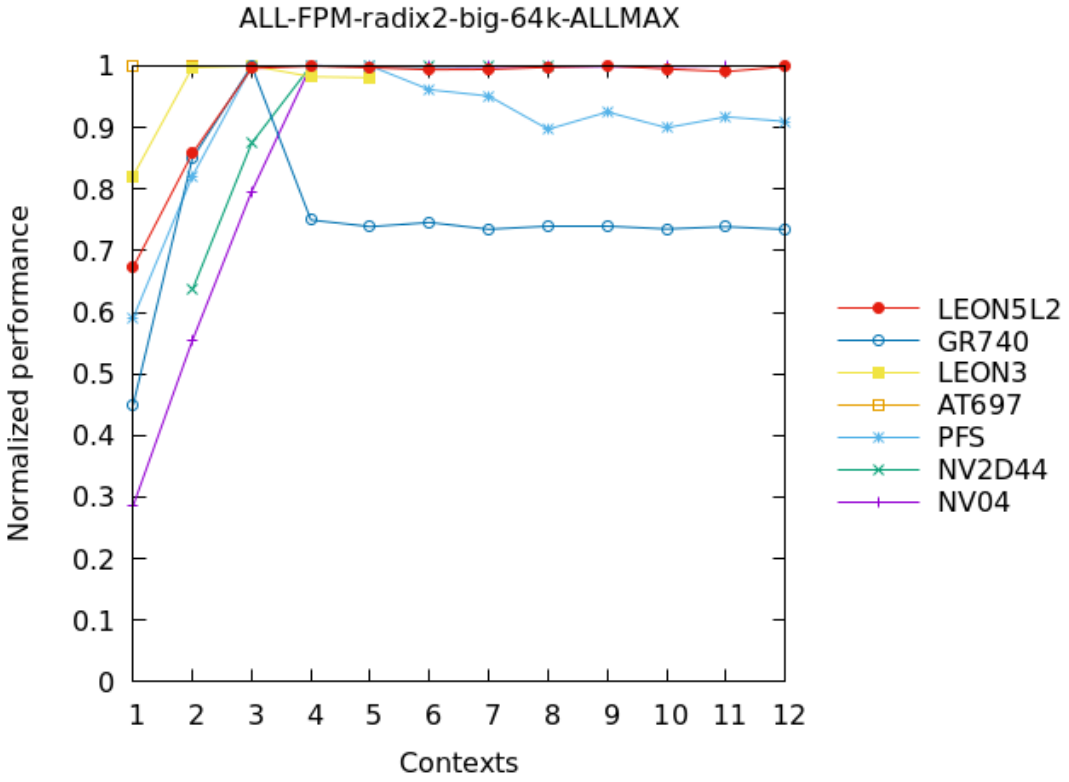

Figure 270: ALL - FPMark - radix2-big-64k, iterations per second at 100MHz. Higher values denote better performance.

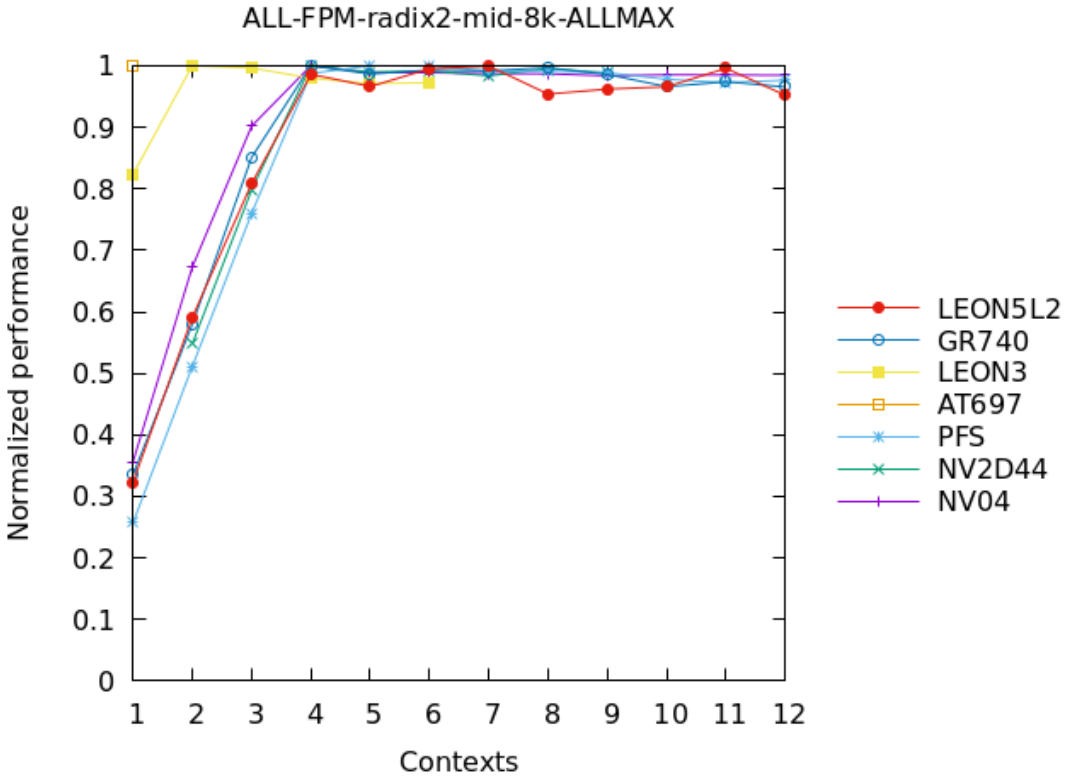

Figure 271: ALL - FPMark - radix2-mid-8k, iterations per second at 100MHz. Higher values denote better performance.

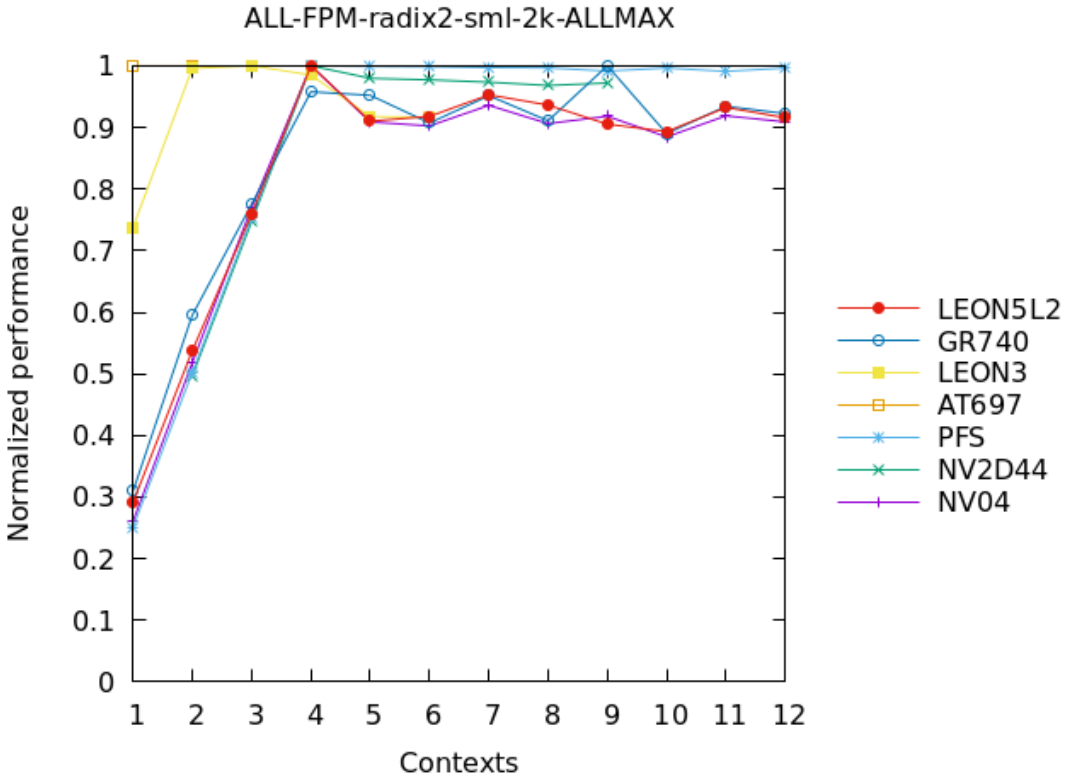

Figure 272: ALL - FPMark - radix2-sml-2k, iterations per second at 100MHz. Higher values denote better performance.

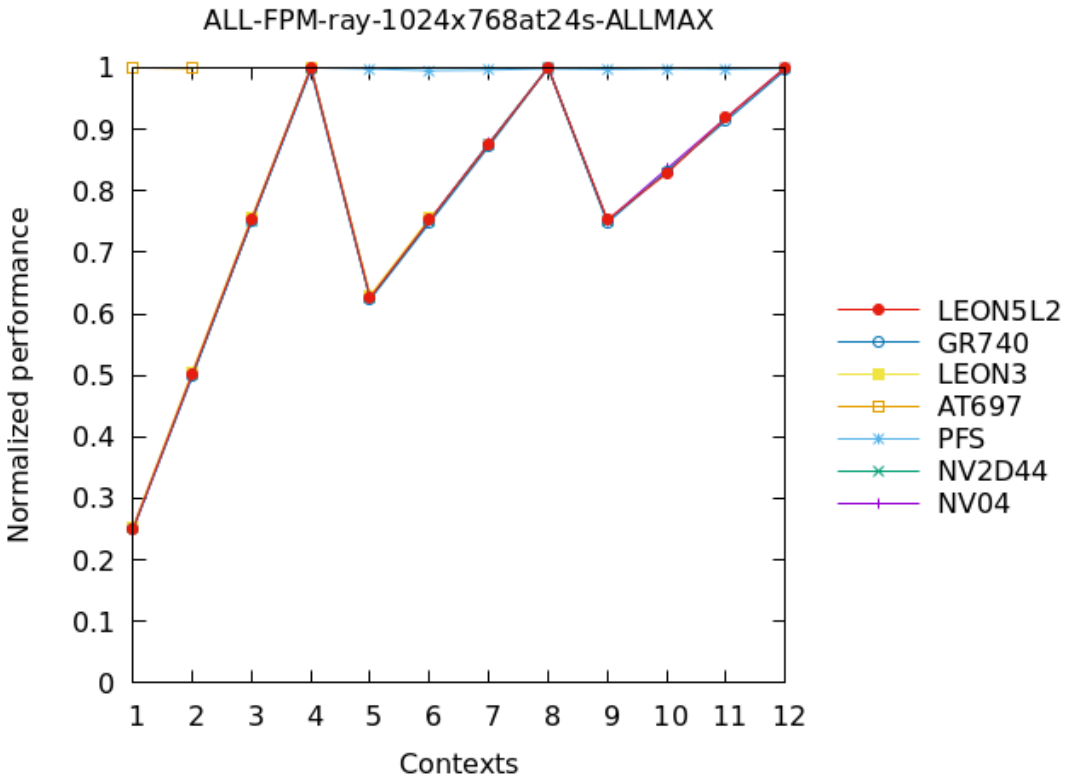

Figure 273: ALL - FPMark - ray-1024x768at24s, iterations per second at 100MHz. Higher values denote better performance.

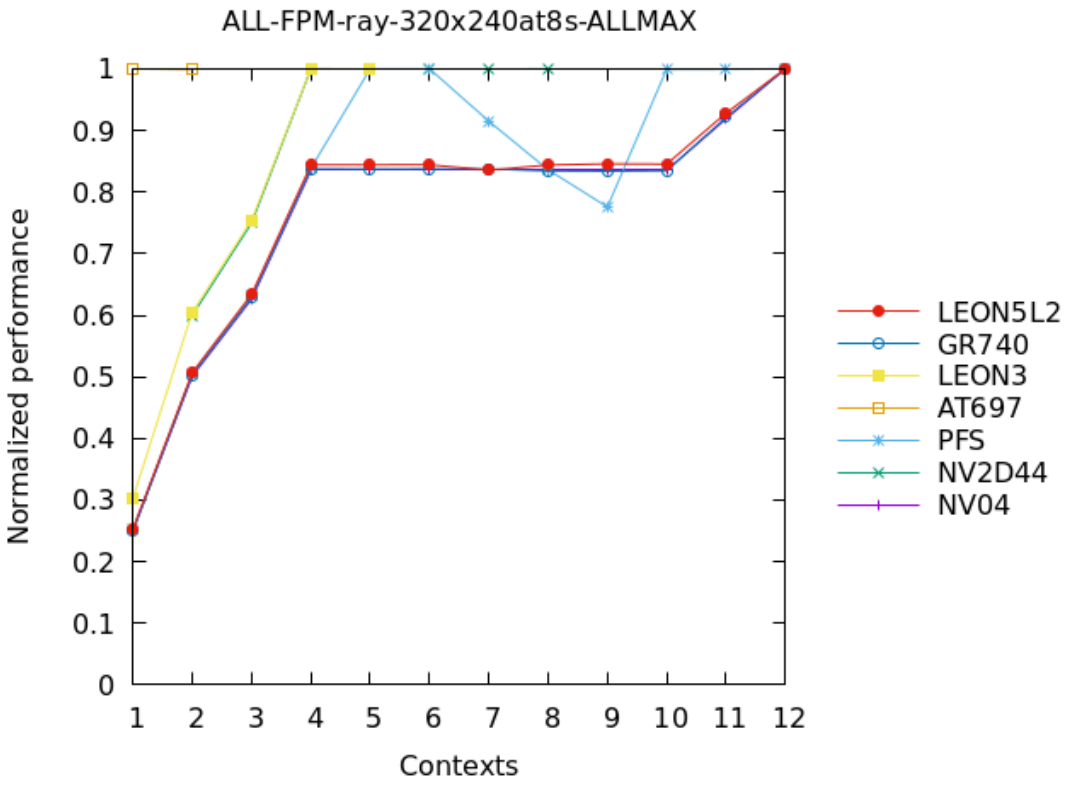

Figure 274: ALL - FPMark - ray-320x240at8s, iterations per second at 100MHz. Higher values denote better performance.

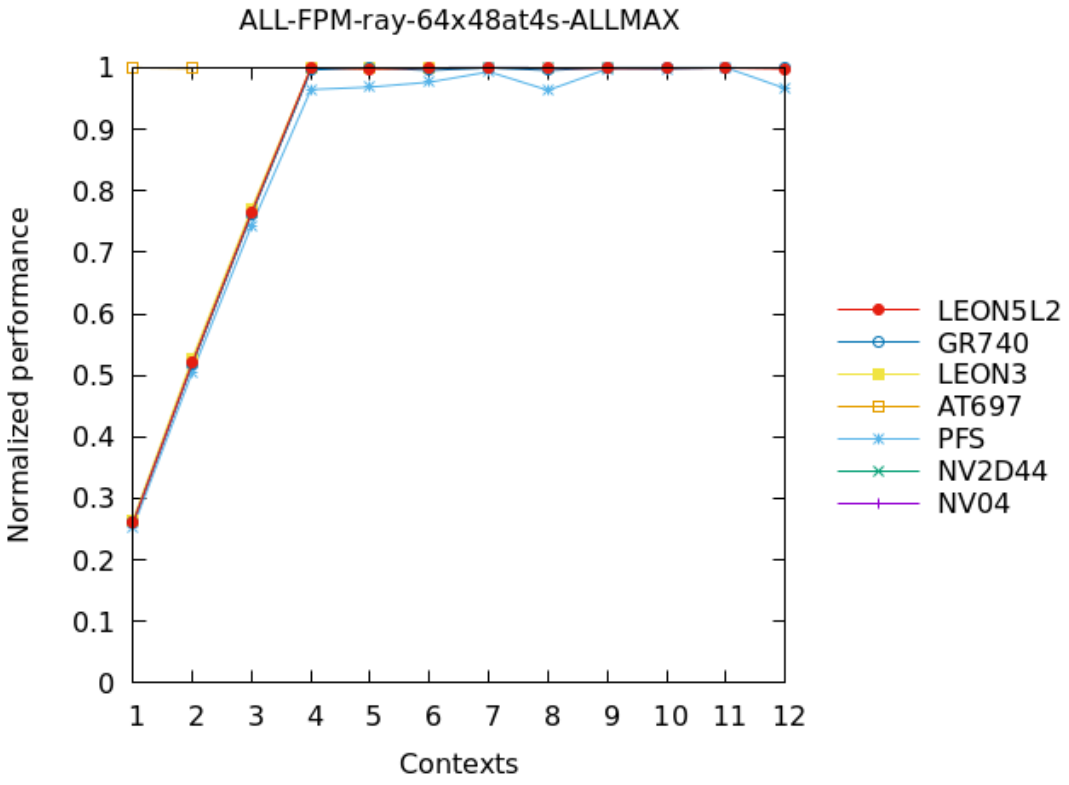

Figure 275: ALL - FPMark - ray-64x48at4s, iterations per second at 100MHz. Higher values denote better performance.

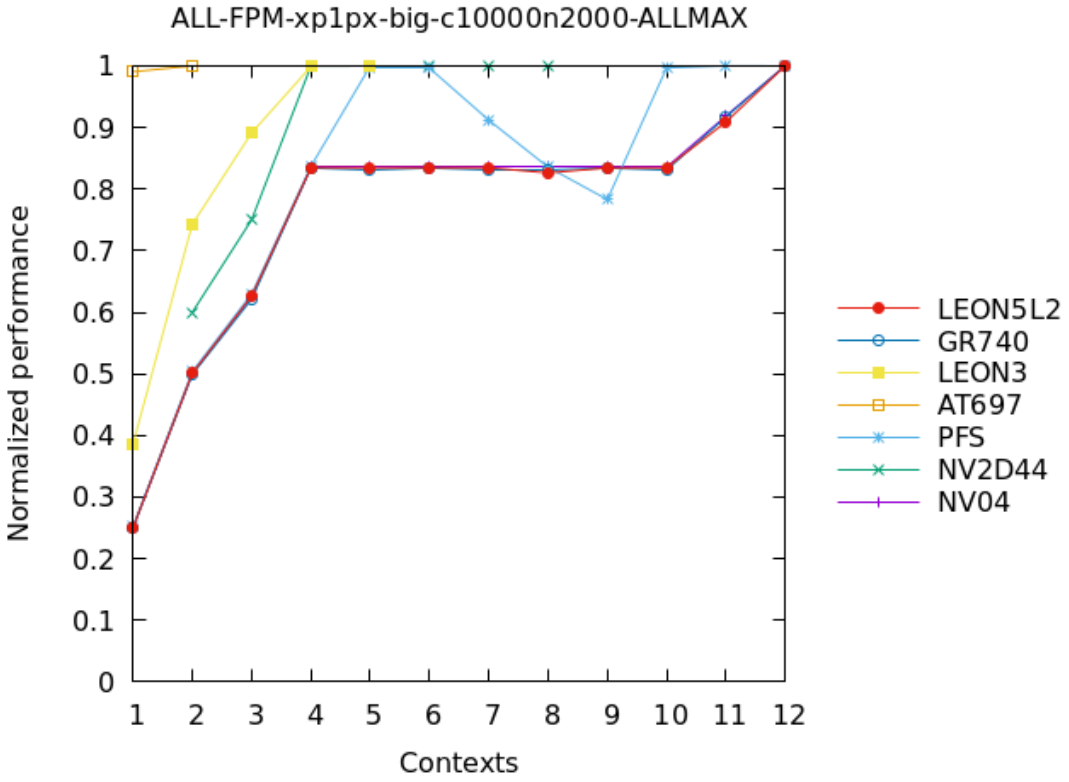

Figure 276: ALL - FPMark - xp1px-big-c10000n2000, iterations per second at 100MHz. Higher values denote better performance.

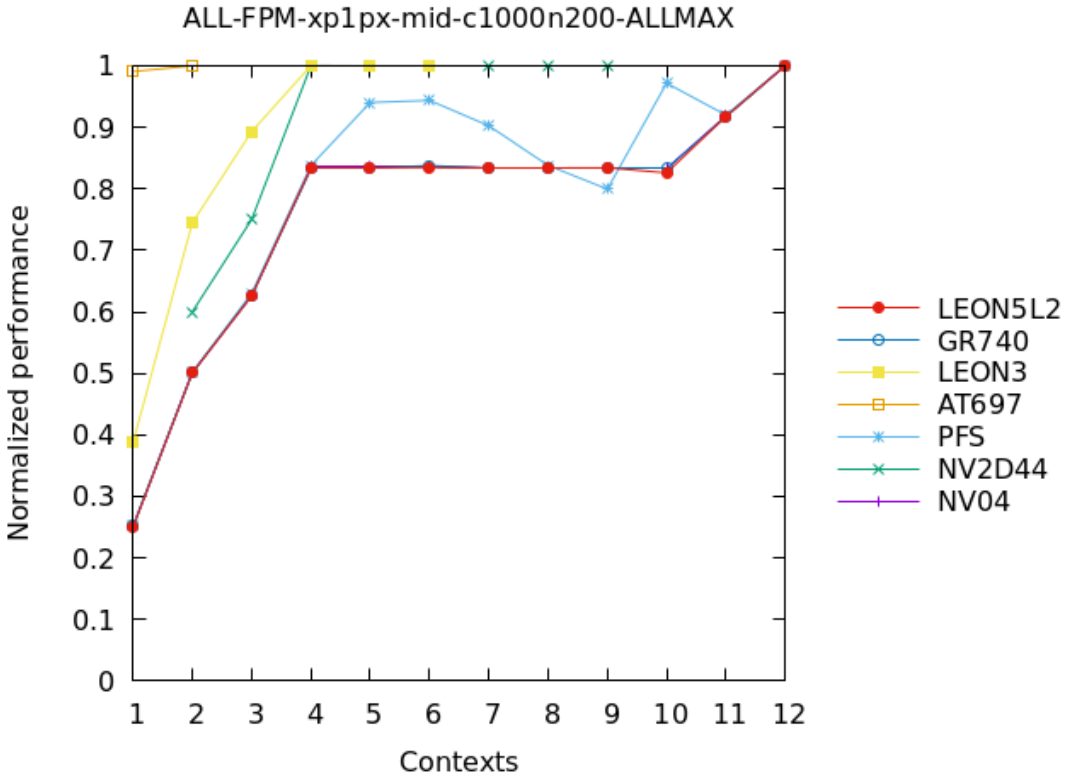

Figure 277: ALL - FPMark - xp1px-mid-c1000n200, iterations per second at 100MHz. Higher values denote better performance.

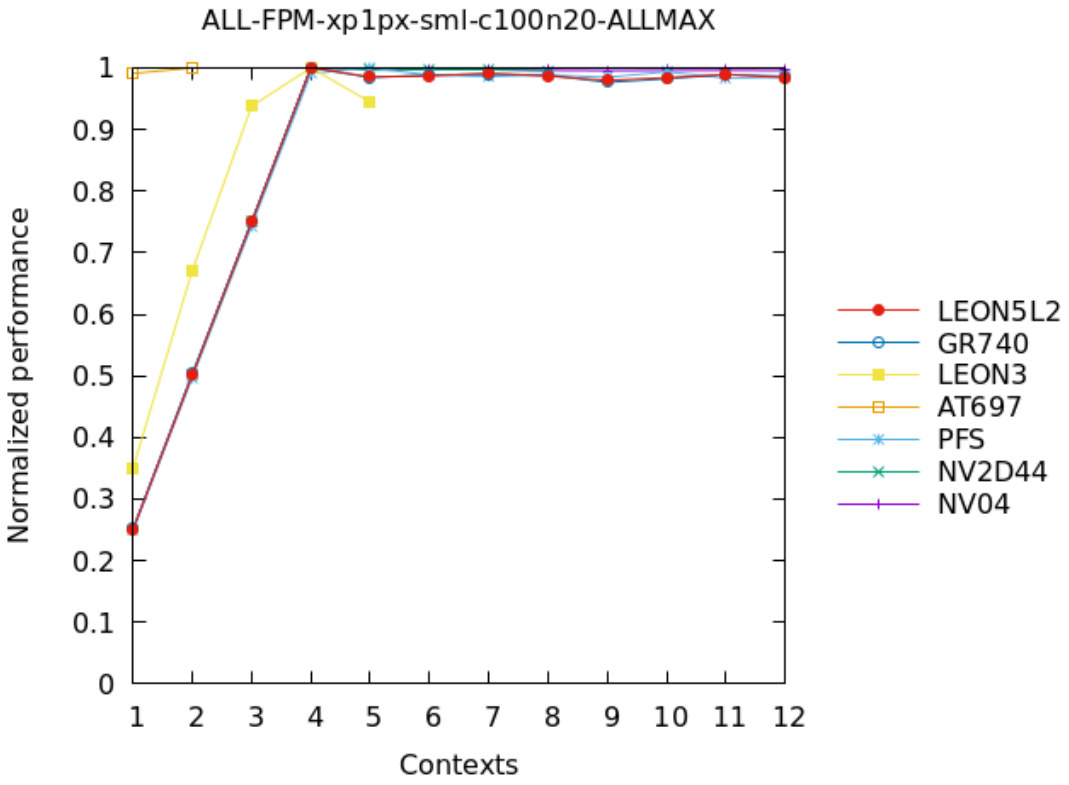

Figure 278: ALL - FPMark - xp1px-sml-c100n20, iterations per second at 100MHz. Higher values denote better performance.

11 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 normalized performance difference between NOEL-V *NV04* and *NV14* is very small.
2. Cache size: The normalized performance difference between NOEL-V *NV24* and *NV34* 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 *NV04* brings about a 20-40% performance increase compared to configuration *NV24*, in other cases the normalized performance is almost identical.
4. No caches: The lack of caches in NOEL-V configurations *NV51* and *NV63* 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* - *nv04* to *nv34*. 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.
7. *daiFPUrv* further improves the NOEL-V normalized floating performance over *nanoFPU* by a factor of 2.
8. In most cases *PolarFire SoC RISC-V* has better normalized floating-point performance than any NOEL-V configuration.
9. For integer workloads *NOEL-V* has better normalized performance than *PolarFire SoC RISC-V*.

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.
2. In general it is good to disable generation of fused floating-point operations for floating-point benchmarks[1] .

[1] FPMark has problems executing RISC-V binaries with fused operations. Paranoia reports rounding errors when compiled with fused operations enabled.

12 List of tables

13 List of figures

14 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.

[FPM] FPMark - An EEMBC Benchmark. https://www.eembc.org/fpmark/. Accessed: 2021-11-07.

[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.

[PFS] PolarFire SoC Product Overview. https://www.microsemi.com/document-portal/doc_download/1244584-polarfire-soc-product-overview. Accessed: 2021-11-07.

[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
2021/11/03	M.Daněk	Added FPMark data and analysis

Disclaimer: