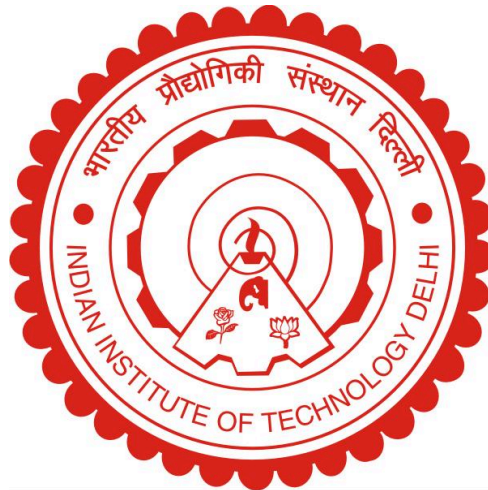


Assignment 3

Adding Custom Instruction in GEM-5

Submitted By : Manish Kumar(2023MCS2497)

Submitted On : **11th November, 2024**



DEPARTMENT OF COMPUTER SCIENCE & TECHNOLOGY
Indian Institute of Technology, Delhi
Hauz Khas, New Delhi, 110016

Academic Year: 2024-25

Problem Statement:

- **Implementation of Combination formula(nCr)** : As a custom instruction in GEM5.
- **Simulation Using gem5**: Clone the GEM5 repo. - built RISC-V. Also, cloned the RISC-V toolchain, to support custom instruction implementation.
- **Performance Analysis**: Describe and analyze the performance gain achieved by the custom instruction for evaluating coefficients in the binary expansion series of $(A + X)^n$

Simulation Parameters:

- **CPU Architecture**:. RISC-V
- **Binomial Expansion Formula**:.
 - **simple Binomial Expansion that uses Factorial & Combination, implicitly**
- **Analyzed Performance Gain** for different values of N in the expansion $(A + X)^n$, incase of Normal Binomial expansion code vs Binomial expansion as a custom ins. in the ISA of RISC-V.

For this assignment, we built Gem5 with RISC-V support and added custom instructions. To accomplish this, we utilized the riscv-gnu-toolchain as a cross-compiler to compile C code for the RISC-V architecture. We also built the riscv-gnu-toolchain to run on the x86 architecture.

Steps Involved:

- **Cloning Gem5 & RISC-V toolchain**

```
$ git clone https://github.com/gem5/gem5.git
```

```
$ git clone https://github.com/riscv-collab/riscv-gnu-toolchain.git
```

- **Building RISC-V toolchain :**

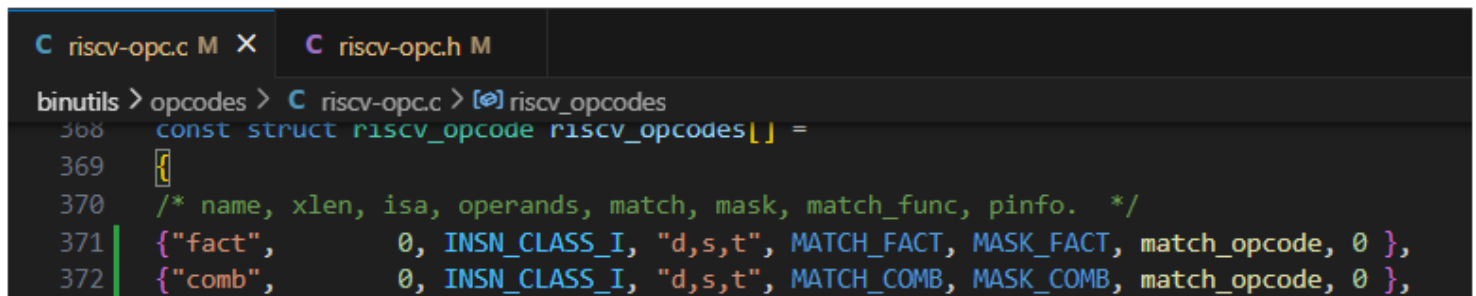
(switch to risc-v toolchain directory & perform the following cmds)

```
$. /configure -- prefix = /opt/riscv
```

```
$ sudo make -j 8
```

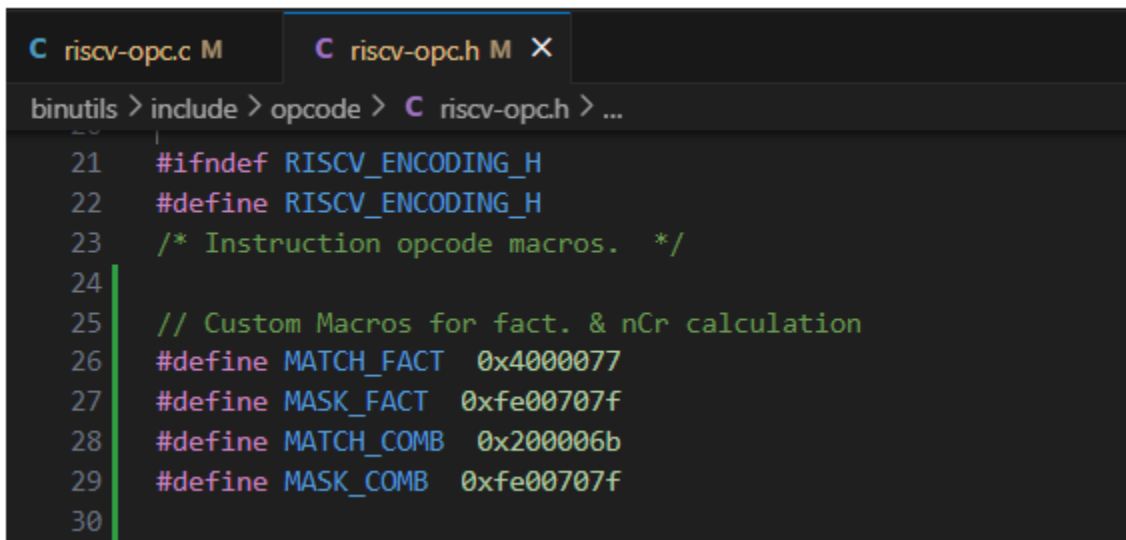
- **Adding Custom Instruction in RISC-V:**

(added the definitions of custom instr. "fact" & "comb" in file:binutils/opcode/riscv-ops.c)



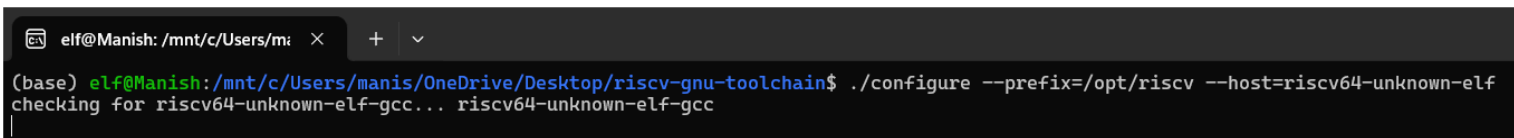
```
C riscv-opc.c M X C riscv-opc.h M
binutils > opcodes > C riscv-opc.c > [⌕] riscv_opcodes
368 const struct riscv_opcode riscv_opcodes[] =
369 {
370     /* name, xlen, isa, operands, match, mask, match_func, pinfo. */
371     {"fact",      0, INSN_CLASS_I, "d,s,t", MATCH_FACT, MASK_FACT, match_opcode, 0 },
372     {"comb",      0, INSN_CLASS_I, "d,s,t", MATCH_COMB, MASK_COMB, match_opcode, 0 },
```

(*&*, updated the macros in file `binutils/include/opcode/riscv-ops.h`)



```
C riscv-opc.c M  C riscv-opc.h M X
binutils > include > opcode > C riscv-ops.h > ...
21  #ifndef RISC_V_ENCODING_H
22  #define RISC_V_ENCODING_H
23  /* Instruction opcode macros. */
24
25  // Custom Macros for fact. & nCr calculation
26  #define MATCH_FACT  0x4000077
27  #define MASK_FACT  0xfe00707f
28  #define MATCH_COMB  0x200006b
29  #define MASK_COMB  0xfe00707f
30
```

- **Rebuilding RISC-V toolchain with addition of Custom instr.:**



```
elf@Manish: /mnt/c/Users/m... X + v
(base) elf@Manish:/mnt/c/Users/manis/OneDrive/Desktop/riscv-gnu-toolchain$ ./configure --prefix=/opt/riscv --host=riscv64-unknown-elf
checking for riscv64-unknown-elf-gcc... riscv64-unknown-elf-gcc
```

`$ sudo make clean`

`$ sudo make -j 8`

- Compiling the file using riscv64-unknown-elf-gcc compiler :

Code:

```
assignment3 > C custom.c > main()
1  #include<stdio.h>
2  #include<stdint.h>
3
4  int main() {
5      uint32_t a = 10, b = 0, value = 0;
6      uint32_t n = 8, r = 5, comb = 0;
7
8      asm volatile("fact %0, %1, %2\n"
9                  : "=r"(value)           // output operand
10                 : "r"(a), "r"(b)        // input operand
11                 :
12                );                      // inline assembly ins. for factorial
13
14      asm volatile("comb %0, %1, %2\n"
15                  : "=r"(comb)           // output operand
16                 : "r"(n), "r"(r)        // input operand
17                 :
18                );                      // inline assembly ins. for combination
19
20      printf("\nValue for expansion of %d! : %u", a, value);
21      printf("\nValue for expansion of %dn%d : %u", n, r, comb);
22      return 0;
23 }
```

For the above C code, we compiled it using the riscv-toolchain with added custom instructions fact & comb. Following are the results obtained in the search query for Object dump of the custom.o file :

```
$/opt/riscv/bin/riscv64 - unknown - elf - objdump - D custom.c - o custom.o
```

```
(base) elf@Manish:/mnt/c/Users/manis/OneDrive/Desktop/HPC_sem3/assignment3$ /opt/riscv/bin/riscv64-unknown-elf-objdump -D custom.o | grep fact
10206: 04e787f7      fact    a5,a5,a4
(base) elf@Manish:/mnt/c/Users/manis/OneDrive/Desktop/HPC_sem3/assignment3$ /opt/riscv/bin/riscv64-unknown-elf-objdump -D custom.o | grep comb
10216: 02e787eb      comb    a5,a5,a4
```

The "fact" & "comb" Instructions were parsed successfully!

- **Building RISC-V & executing the simulation on the custom.o object file:**

```
$ scons build/RISC - V/gem5.opt - j 12
```

(switched to risc-v build directory & perform the following cmd)

```
$. /gem5.opt simple - riscv.py
```

(simple-riscv.py was updated to include the computed binary object file)

```
(base) elf@Manish:/mnt/c/Users/manis/OneDrive/Desktop/gem5/build/RISC$ ./gem5.opt simple-riscv.py
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 24.0.0.0
gem5 compiled Nov 10 2024 08:14:59
gem5 started Nov 11 2024 09:36:26
gem5 executing on Manish, pid 654821
command line: ./gem5.opt simple-riscv.py

Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/arch/riscv/isa.cc:276: info: RVV enabled, VLEN = 256 bits, ELEN = 64 bits
src/arch/riscv/linux/se_workload.cc:60: warn: Unknown operating system; assuming Linux.
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation!
src/sim/simulate.cc:199: info: Entering event queue @ 0. Starting simulation...
src/arch/riscv/faults.cc:204: panic: Unknown instruction 0x4000010004e787f7 at pc (0x10206->0x1020a).(0->1)
Memory Usage: 640176 KBytes
```

Since we didn't add support for those instructions in Gem5, it encountered an error because it did not know how to decode & execute them.

- **Adding Custom instructions in RISC-V ISA :**

```
simple-riscv.py 9+  decoder.isa M X
src > arch > riscv > isa > decoder.isa
43  decode QUADRANT default Unknown::unknown() {
529  0x3: decode OPCODE5 {
4883 > 0x1c: decode FUNCT3 { ...
5026  }
5027
5028  0x1d: decode FUNCT3 {
5029      format ROp {
5030          0x0: decode = FUNCT7 {
5031              0x02: fact({{
5032                  uint32_t n = Rs1;
5033                  uint32_t r = Rs2;
5034                  uint64_t res = 1;
5035
5036                  while(n > 1) {
5037                      res *= n;
5038                      n -= 1;
5039                  }
5040
5041                  Rd = res;
5042              }}};
5043      }
5044  }
5045  }
5046
5047  0x1e: M5Op::M5Op();
```

```
simple-riscv.py 9+  decoder.isa M X
src > arch > riscv > isa > decoder.isa
43  decode QUADRANT default Unknown::unknown() {
529  0x3: decode OPCODE5 {
4877
4878  0x1a: decode FUNCT3 {
4879      format ROp {
4880          0x0: decode = FUNCT7 {
4881              0x01: comb({{
4882                  uint32_t n = Rs1;
4883                  uint32_t r = Rs2;
4884                  uint64_t res = 1, i = 1;
4885                  r = (r > n - r) ? n - r : r;
4886                  while(i != r + 1) {
4887                      res *= n;
4888                      res /= i;
4889                      n -= 1;
4890                      i += 1;
4891                  }
4892
4893                  Rd = res;
4894              }}};
4895      }
4896  }
4897  }
```

To define the custom RISC-V instructions, we need to modify the `decoder.isa` file located at `src/arch/riscv/isa/`. This file has a nested data structure, with everything organized under a "Quadrant" construct specific to Gem5, which is not part of the original RISC V instruction types. We referred to this medium article [link](#) to identify the instruction types.

Upon adding the decode information for our instructions, **we rebuilt the RISC-V & ran the `simple-risc.py` file again!**

```
(base) elf@Manish:/mnt/c/Users/manis/OneDrive/Desktop/gem5/build/RISCV$ ./gem5.opt simple-riscv.py
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 24.0.0.0
gem5 compiled Nov 11 2024 23:11:00
gem5 started Nov 11 2024 23:48:23
gem5 executing on Manish, pid 192218
command line: ./gem5.opt simple-riscv.py

Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/arch/riscv/isa.cc:276: info: RVV enabled, VLEN = 256 bits, ELEN = 64 bits
src/arch/riscv/linux/se_workload.cc:60: warn: Unknown operating system; assuming Linux.
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation!
src/sim/simulate.cc:199: info: Entering event queue @ 0. Starting simulation...

Value for expansion of 10! : 3628800
Value for expansion of 8n5 : 56Exiting @ tick 387613000 because exiting with last active thread context
```

Our custom defined instructions are now being processed by the risc-v toolchain gcc compiler!

- **Performance Gain:**

- We implemented two versions of code for binomial expansion: one that uses a custom instruction to compute nCr values, and another that calculates nCr values through a C-code function without the custom instruction. We compiled and executed both versions, adjusting the values of n, and analyzed the m5out/stats.txt file to obtain the following graphs and results. The performance gain was calculated based on execution time using the formula below:

$$Gain = (Simseconds(without custom_{ins}) - Simseconds(custom_{ins})) / Simseconds(without custom_{ins})$$

We've varied value for N from 50 to 120 with a step size of 10 & obtained the following data:

Folder	Mode	N	Execution Time	CPI	Number of Cycles
customcombination50_stats	custom	50	0	96.71	3966148
normalcombination50_stats	normal	50	0.01	79.64	8919696
normalcombination60_stats	normal	60	0.01	79.77	12544915
customcombination60_stats	custom	60	0.01	97.41	5487727
customcombination70_stats	custom	70	0.01	98.02	7231106
normalcombination70_stats	normal	70	0.02	79.9	16764602
normalcombination80_stats	normal	80	0.02	79.94	21638745
customcombination80_stats	custom	80	0.01	98.39	9278651
customcombination90_stats	custom	90	0.01	98.59	11577673
normalcombination90_stats	normal	90	0.03	79.98	27140750
normalcombination100_stats	normal	100	0.03	80.02	33268969
customcombination100_stats	custom	100	0.01	98.77	14140216
normalcombination110_stats	normal	110	0.04	80.04	40023568
customcombination110_stats	custom	110	0.02	98.9	16960692
normalcombination120_stats	normal	120	0.05	80.05	47398986
customcombination120_stats	custom	120	0.02	99	20039249

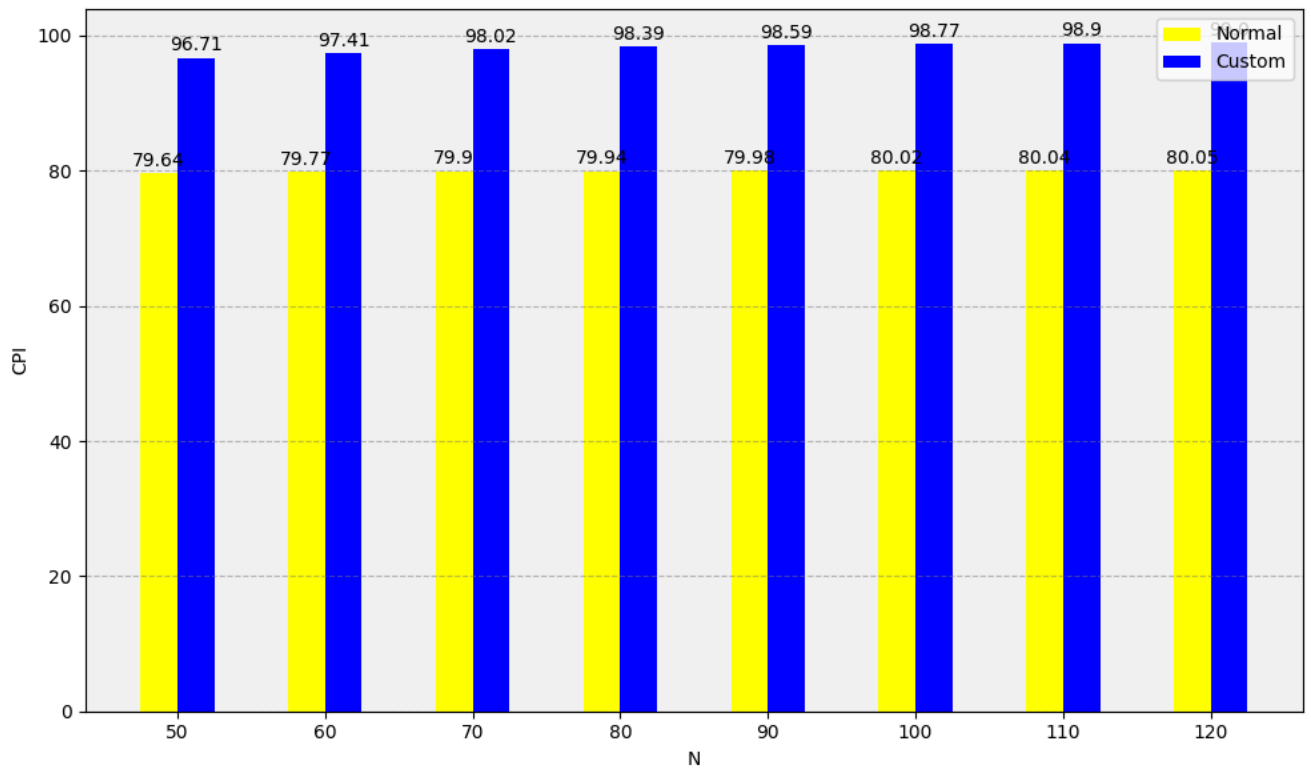


Fig: Variation of CPI based on value of N in the implementation

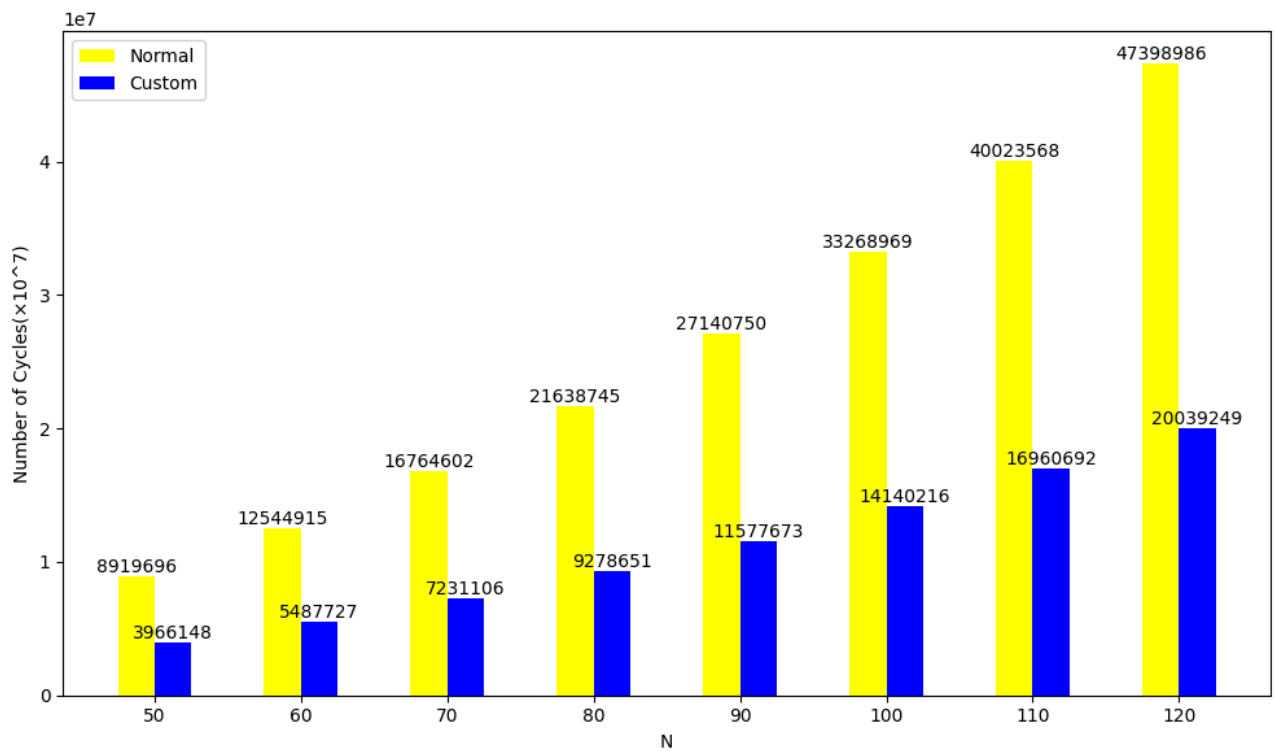


Fig: Variation of num_cycles based on value of N

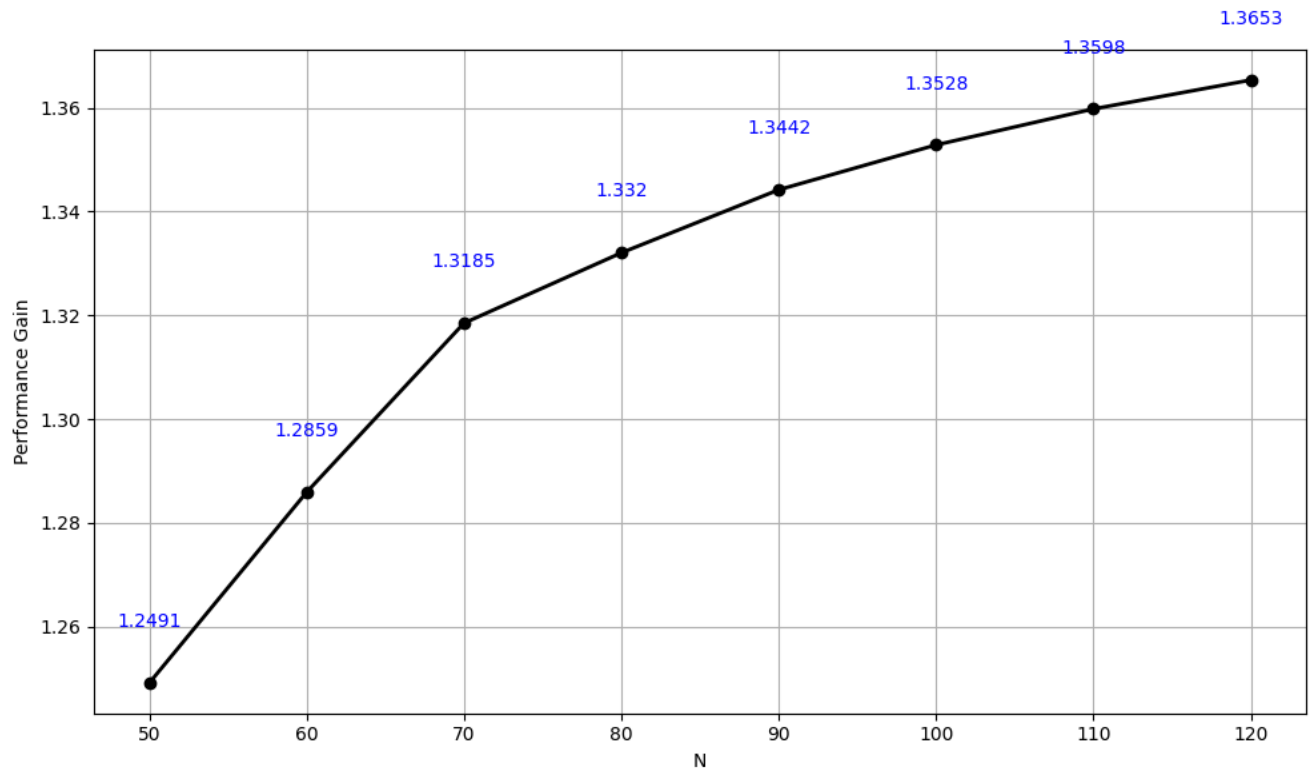


Fig.: Overall Performance Gain

- From the above trend, it can be noticed that there was a huge performance gain when we had implemented the Factorial & Combination as a part of ISA (as custom instructions) rather than computing them in the system.