GeST an Automatic Framework for Generating CPU Stress-Tests

Zacharias Hadjilambrou (University of Cyprus), Shidhartha Das, Paul Whatmough, David Bull (ARM), Yiannakis Sazeides (University of Cyprus)

ISPASS-2019
Madison, March 24-26, 2019

Motivation

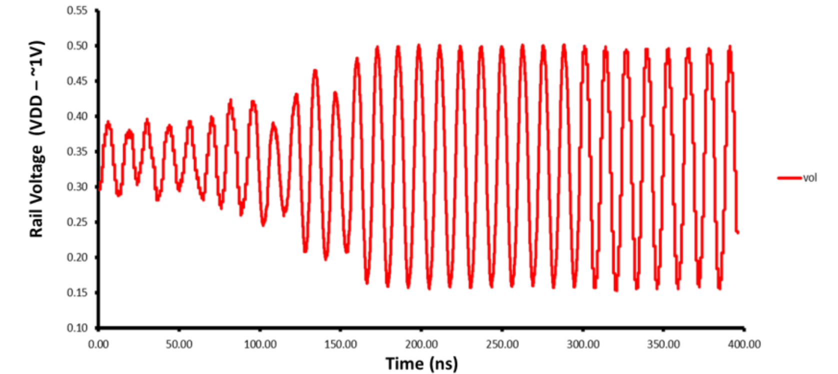
Power viruses

Check TDP, IR drop within specifications

Performance Stress-tests

Performance comparison, max sustainable throughput exploration

dI/dt voltage-noise viruses



Determine chip's operational voltage frequency points

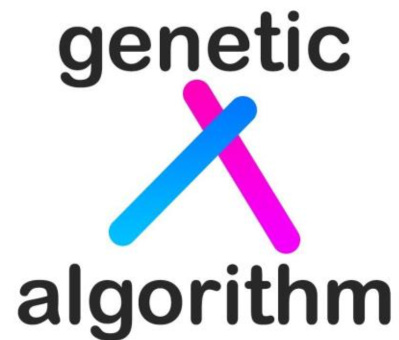
Is **tedious** and **time-consuming** to **manually** craft such stress-tests (viruses)

Need for an **automatic** framework!

This work

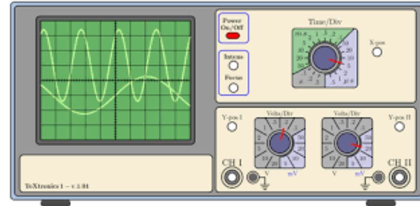
- Proposes **GeST** (Generating Stress Tests): A **Genetic Algorithm (GA)** based framework for automatically generating **stress-tests**

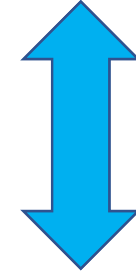
- Written in Python3, inputs defined in XML



GeST overview

```
[root@server ~]# perf list sw
```





ADD
SUB
MUL
DIV
…
r0
r1
r2
…

genetic
algorithm

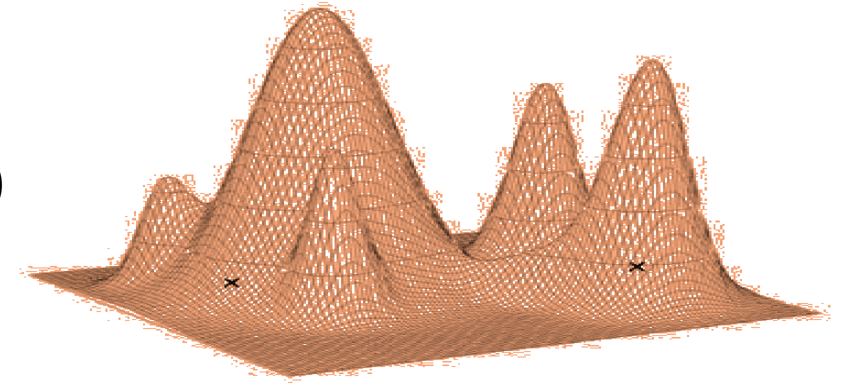
```
1  Loop:
2      STP x6,x7,[x10,#16]
3      FDIV v2.4s,v4.4s,v0.4s
4      LDP x2,x3,[x10,#240]
5      FMUL d8,d25,d26
6      FMUL d6,d5,d13
7      UMADDL x30,w15,w24,x21
8      FMADD d17,d17,d20,d20
9      FRSQRTS v10.4s,v1.4s,v1.4s
10     ASR x22,x30,#31
11     STP x6,x7,[x10,#48]
12     FDIV v11.4s,v9.4s,v4.4s
13     LDP x2,x3,[x10,#48]
14     FMLA v8.4s,v6.4s,v8.4s
15     FMUL d22,d15,d30
16     STP x6,x7,[x10,#248]
17     UMADDL x8,w28,w28,x5
18     FMOV d15,d21
19     SUBS x6,x23,x9
20     FMUL v10.4s,v8.4s,v11.4s
21     FCMP d13,d16
22 b Loop
```

Rest of the Presentation

- Genetic Algorithms
- GeST
- Case-study on Ampere X-Gene2 ARM 64bit Server CPU
- GeST vs Related Work
- Conclusion

Why GA

- GA is well suited to a wide range of problems…
  - Well-suited to non-linear optimisations
  - Can handle large search space (e.g. Travelling salesman problem)
  - Can overcome local optima (c.f. linear optimisations)

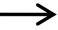


- However…
  - Can not guarantee optimal solution
  - Can be time expensive (depends on measurement time)

- **Overall for our purposes GA offers a good trade-off between time and quality of solution**

GA Flow

Seed Population

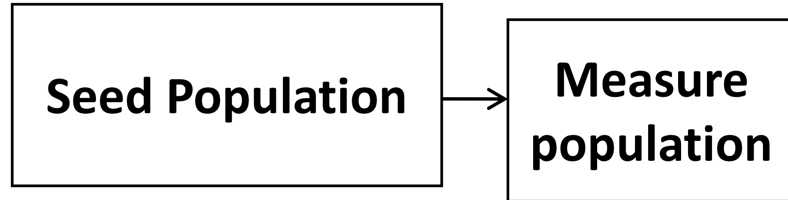


```
1  Loop:
2      STP x6,x7,[x10,#16]
3      FDIV v2.4s,v4.4s,v0.4s
4      LDP x2,x3,[x10,#240]
5      FMUL d8,d25,d26
6      FMUL d6,d5,d13
7      UMADDL x30,w15,w24,x21
8      FMADD d17,d17,d20,d20
9      FRSQRTS v10.4s,v1.4s,v1.4s
10     ASR x22,x30,#31
11     STP x6,x7,[x10,#48]
12     FDIV v11.4s,v9.4s,v4.4s
13     LDP x2,x3,[x10,#48]
14     FMLA v8.4s,v6.4s,v8.4s
15     FMUL d22,d15,d30
16     STP x6,x7,[x10,#248]
17     UMADDL x8,w28,w28,x5
18     FMOV d15,d21
19     SUBS x6,x23,x9
20     FMUL v10.4s,v8.4s,v11.4s
21     FCMP d13,d16
22 b Loop
```

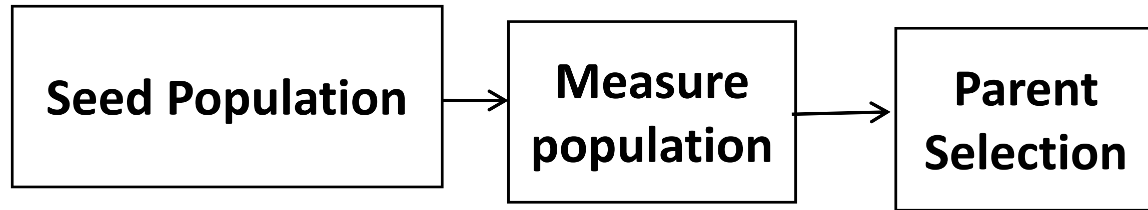
A collection of instruction sequences (**individuals**)

Population size = 50 individuals
Individual size = 15-50 instructions

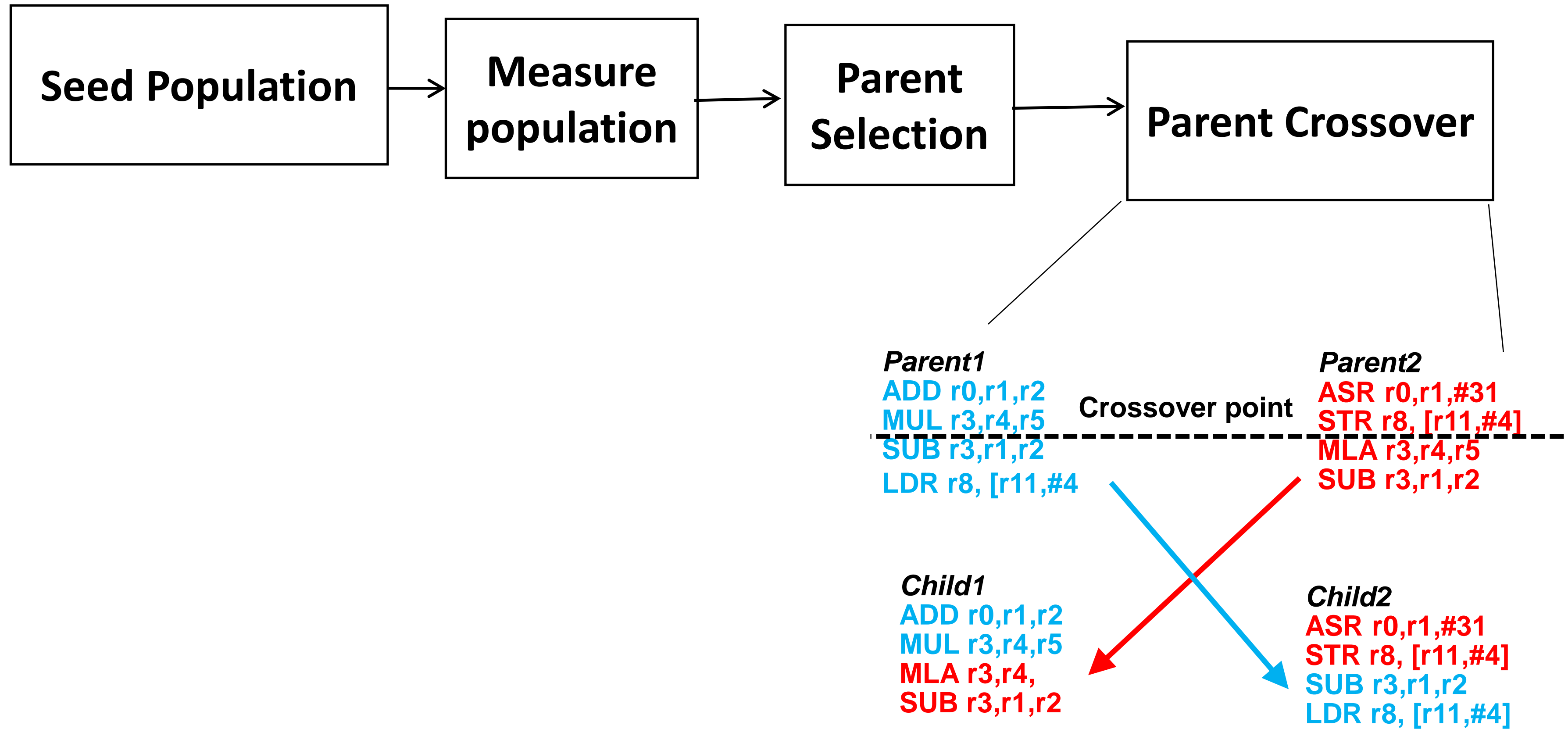
GA Flow



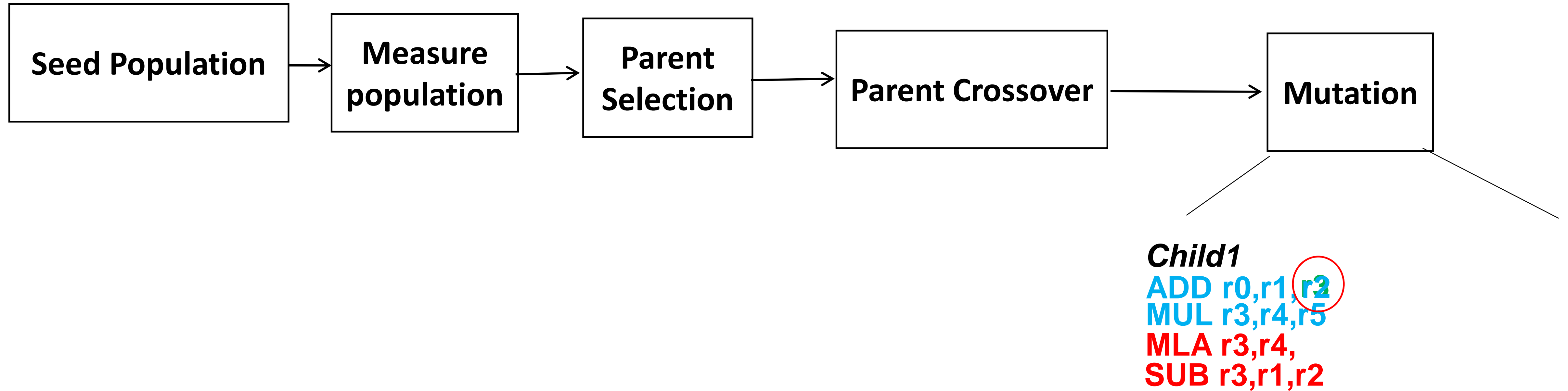
GA Flow



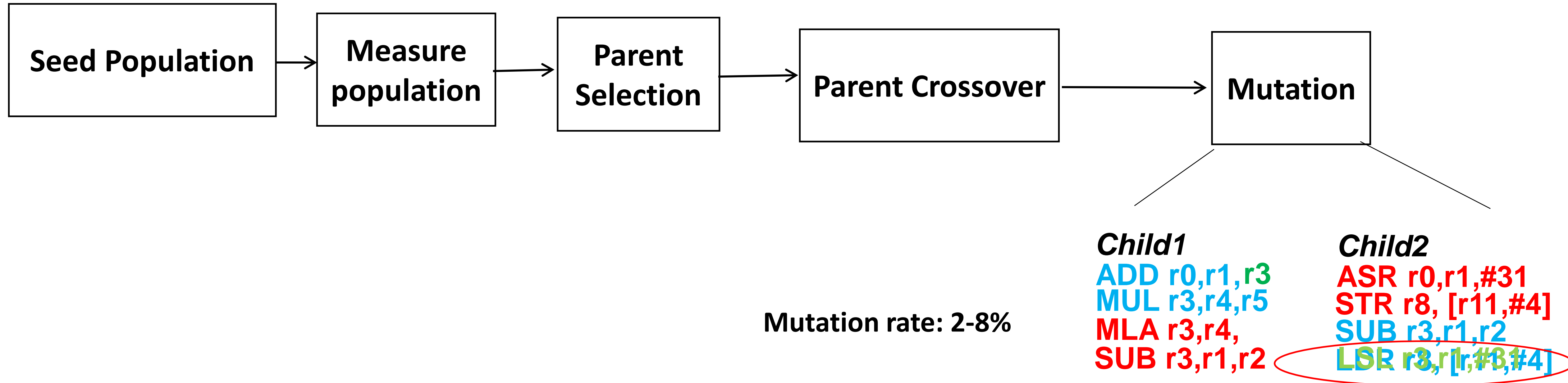
GA Flow



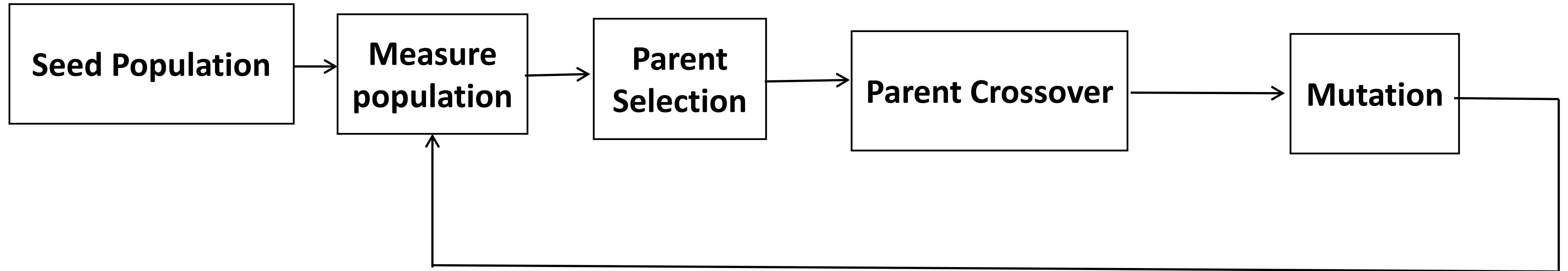
GA Flow



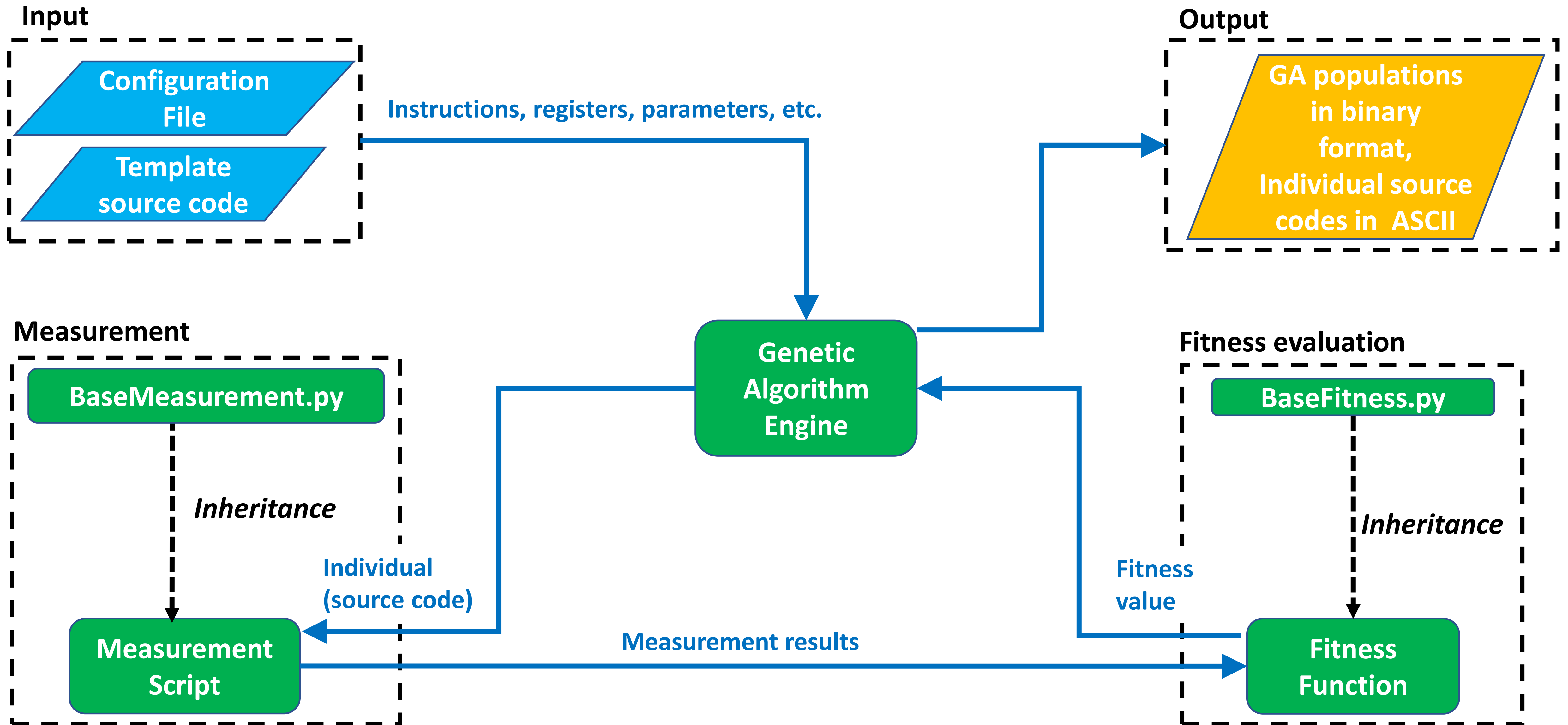
GA Flow



GA Flow



GeST Framework Overview

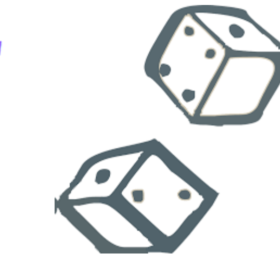


Instruction definition Interface & Code Generation

```
<operand
 id="mem_offset"
 min="4"
 max="128"
 stride="4"
 type="immediate"
 </operand>
```

32 possible values

```
<instruction
name="MOV_1stMem"
format="mov op1(%rsp),op2"
operand1="mem_offset"
operand2="integer_register"
num_of_operands="2"
type="intMem"
</instruction>
```



32*5 = 160 possible instruction forms → **mov 16 (%rsp), %rbx**

```
<operand
 id="integer_register"
 values="%rax %rbx %rdx %rsi %rdi"
 type="register"
 </operand>
```

5 possible values

Total Search Space vs Search Space Covered

20 instructions
160 possible forms each
50 instructions per individual
Total Search Space = 3200^{50} individuals

100 generations
50 individuals per generation
Search Space covered = 5000 individuals

Still GeST achieves good results!

Paper Case studies

CPU	# of Cores	Board	Environment	Stress-test developed	Measurement Instrument
Ampere X-Gene 2	8	Validation Board	Centos 7.2	thermal-virus and IPC virus	i2c temperature sensor readings, performance counters

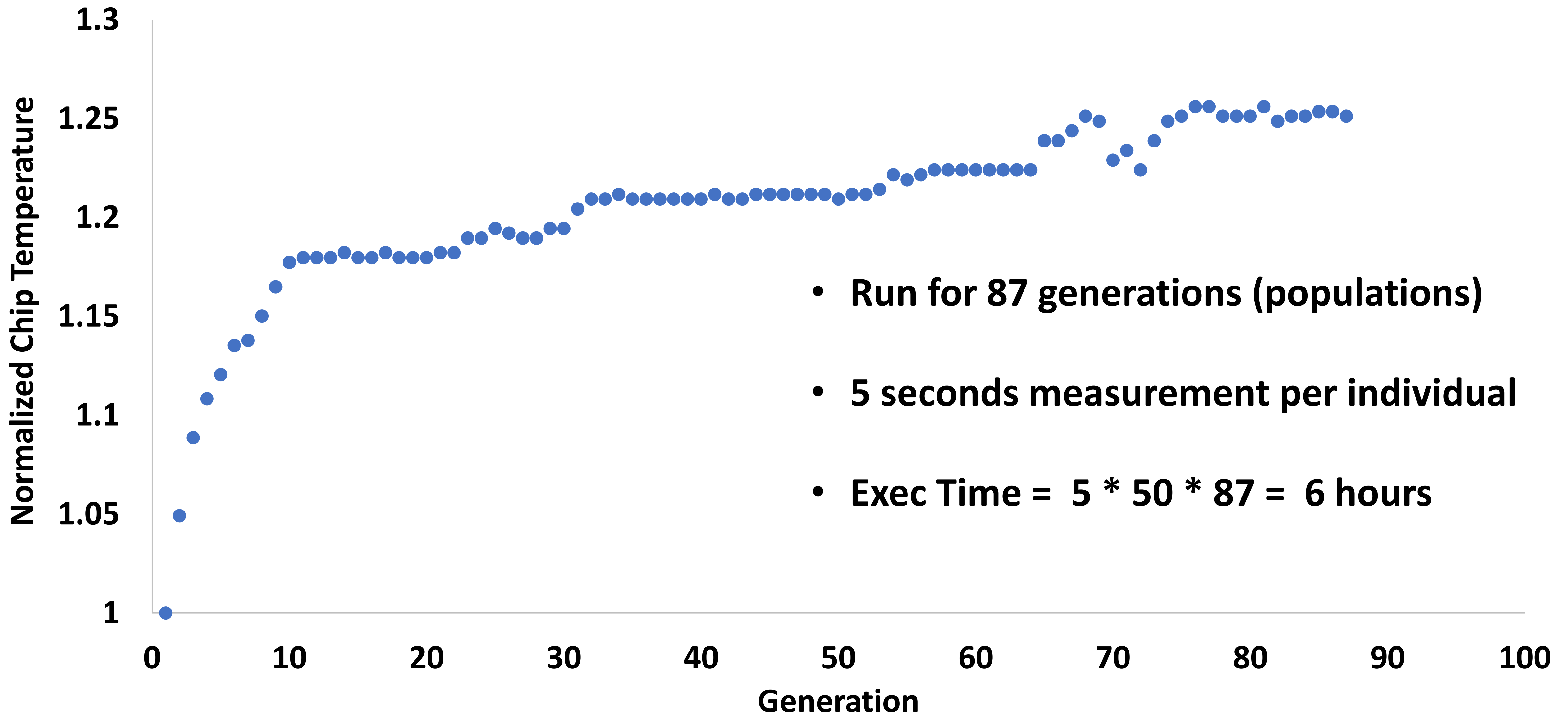
Instructions used

Instruction Type	Instruction
Short Latency Integer	ADD, ADD_LSLS, ADDS, LSL, ASR, ROR, SUB, MOV
Long Latency Integer	MUL
Memory	STP, STR, LDP, LDR
Branch	B
Float/SIMD	FMOV, FMUL, FADD, SCVTF, FCVTAS,FMADD, FSQRT,FABS, FDIV

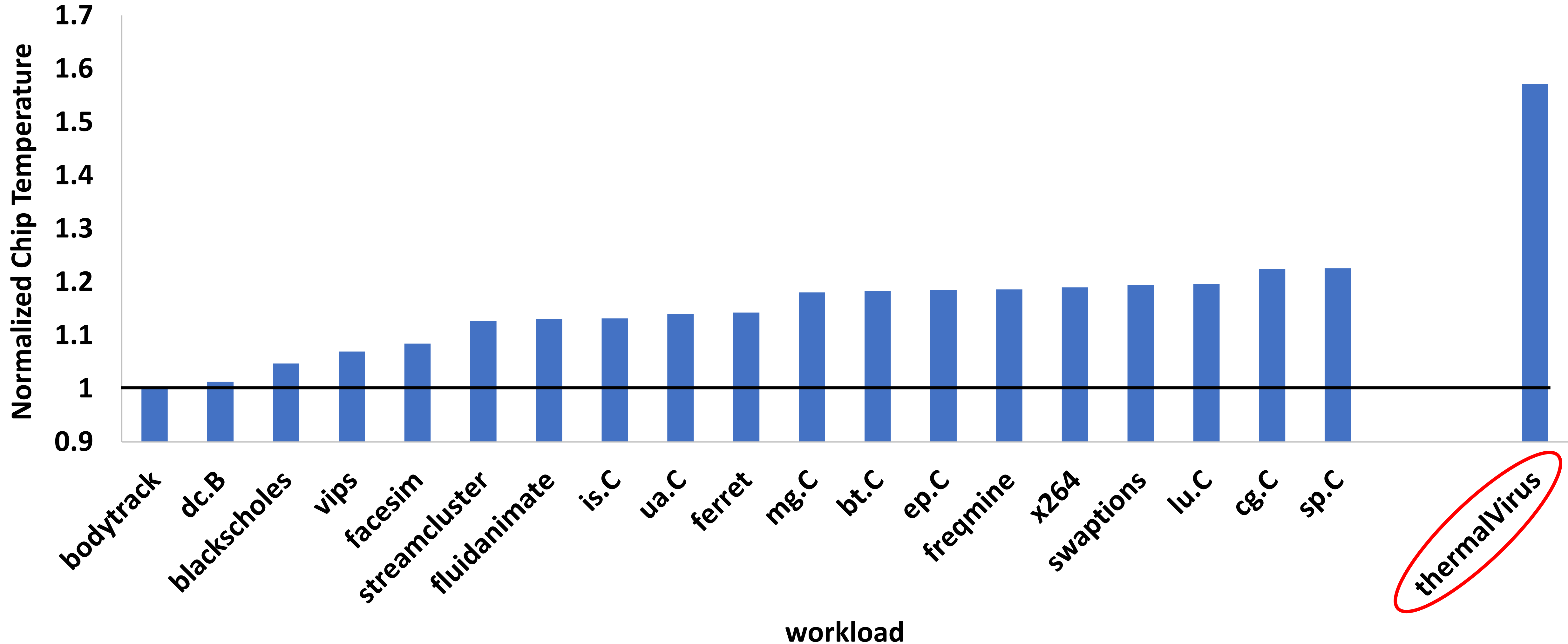
GA parameters used

Parameter	Value
Population size	50
Individual Size (number of loop instructions)	50
Mutation rate	0.02
Crossover operator	one point crossover
Elitism (Best individual promoted to next generation)	TRUE
Parent selection method	Tournament Selection
Tournament size	5

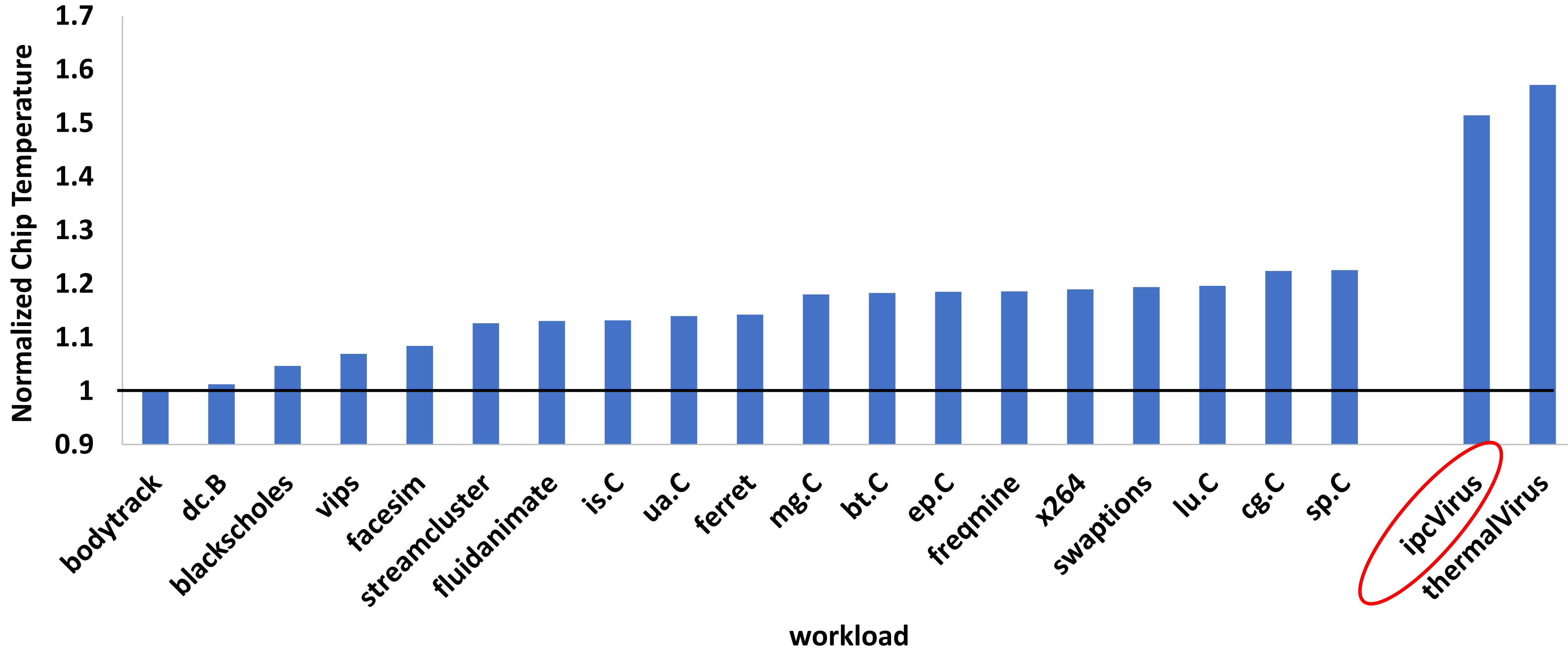
GeST temperature optimization



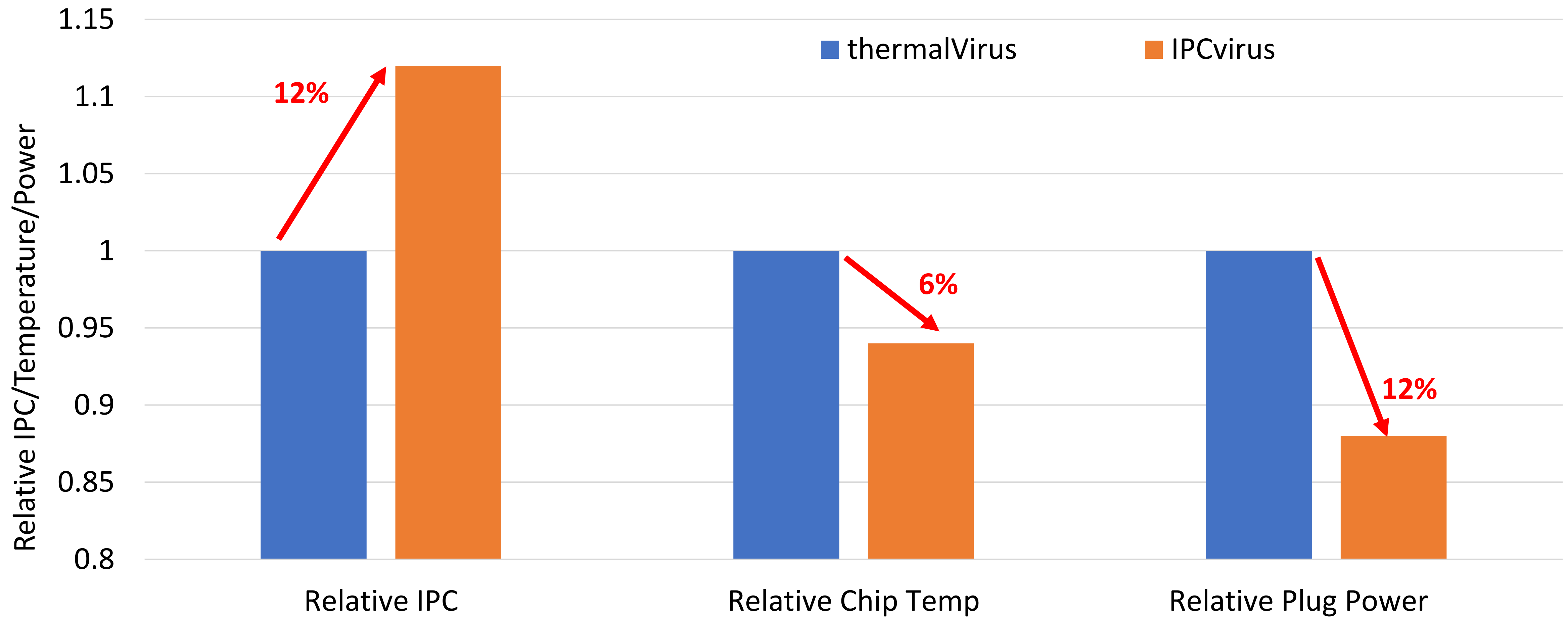
Thermal virus vs conventional workloads



Thermal virus vs conventional workloads

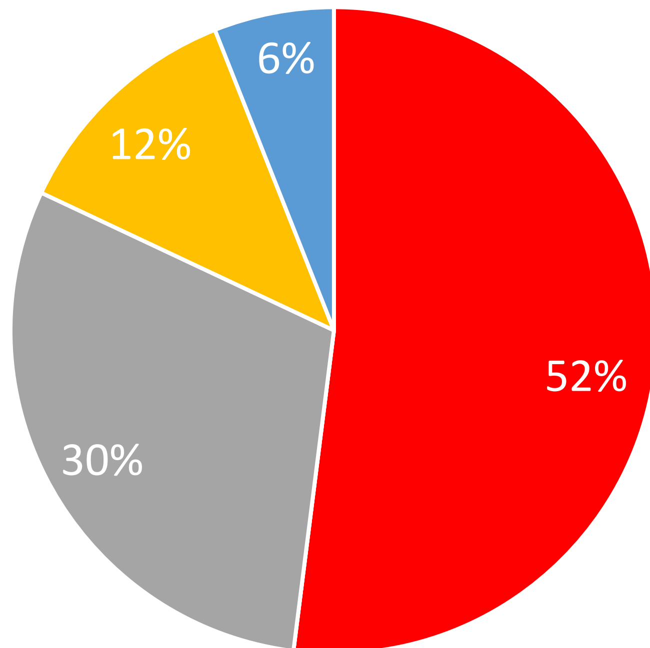


Thermal vs IPC virus



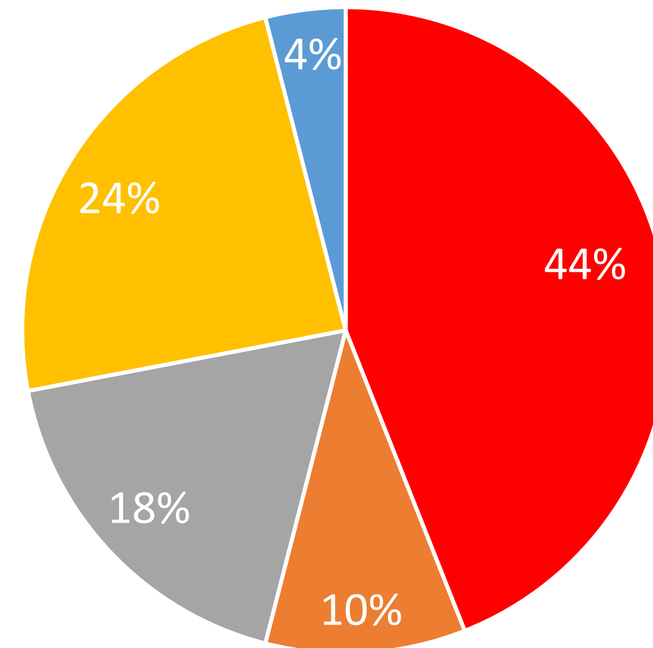
Thermal vs IPC virus instruction breakdown

IPCvirus



■ ShortInt ■ LongInt ■ Float/SIMD ■ Mem ■ Branch

thermalVirus



■ ShortInt ■ LongInt ■ Float/SIMD ■ Mem ■ Branch

GeST in Numbers

	Platform	CPU	OS	ISA	Measurement	Power Virus	dI/dt virus	IPCvirus	Reference
	VersatileExpress	Cortex-A15	BareMetal	ARM	energy probe, oscilloscope	1	1		
	VersatileExpress	Cortex-A7	BareMetal	ARM	energy probe	1			
	JunoR0	Cortex-A57	Debian	ARM	on-chip oscilloscope		1		[1]
	JunoR2	Cortex-A72	Debian	ARM	on-chip oscilloscope	1	1		[2],[3]
	JunoR2	Cortex-A53	Debian	ARM	spectrum analyzer		1		[2],[3]
	Asus M5A78L LE	Athlon II X4 645	Win8.1	x86	external oscilloscope		1		[2]
	Socket LGA1155	Intel i5-2400	Ubuntu	x86	likwid software power meter	1		1	
	Validation Board	X-Gene2	Centos 7.2	ARM	i2c sensors, performance counters	1	1	1	[4]
	Validation Board	X-Gene3	Centos 7.2	ARM	i2c sensors, performance counters	1	1	1	
SUM	**7 platforms**	**9 CPUs**	**5 OS**	**2 ISAs**	**8 instruments**	**6 power virus**	**7 dI/dt viruses**	**3 IPC virus**	**4 papers**

1. Whatmough, Paul N., Shidhartha Das, Z. Hadjilambrou, and David M. Bull. "Power integrity analysis of a 28 nm dual-core arm cortex-a57 cluster using an all-digital power delivery monitor." JSCC 2017
2. Hadjilambrou, Zacharias, Shidhartha Das, Marco A. Antoniades, and Yiannakis Sazeides. "Leveraging CPU Electromagnetic Emanations for Voltage Noise Characterization." MICRO 2018
3. Hadjilambrou, Z., Das, S., Antoniades, M. A., & Sazeides, Y. (2018). Sensing CPU voltage noise through Electromagnetic Emanations. IEEE Computer Architecture Letters, 17(1), 68-71.
4. Tovletoglou K, et al.. Measuring and Exploiting Guardbands of Server-Grade ARMv8 CPU Cores and DRAMs. DSN 2018

GeST vs Related work

Work	Compatible with any ISA or high level language	Extensible	Multi-objective fitness function	ISA	Metric Maximized	Publicly available
AUDIT [MICRO2012]	-	-	-	x86	voltage-noise	No
MAMPO [HPCA 2011]	-	-	-	x86	power	No
Joshi et al [HPCA 2008]	-	-	-	Alpha	power	No
Powermark [MICRO 2011]	-	-	-	x86	power	No
GeST [ISPASS 2019]	Yes	Yes	Yes (power and instruction stream simplicity)	ARM, x86	power, voltage-noise, IPC	Yes

Conclusions

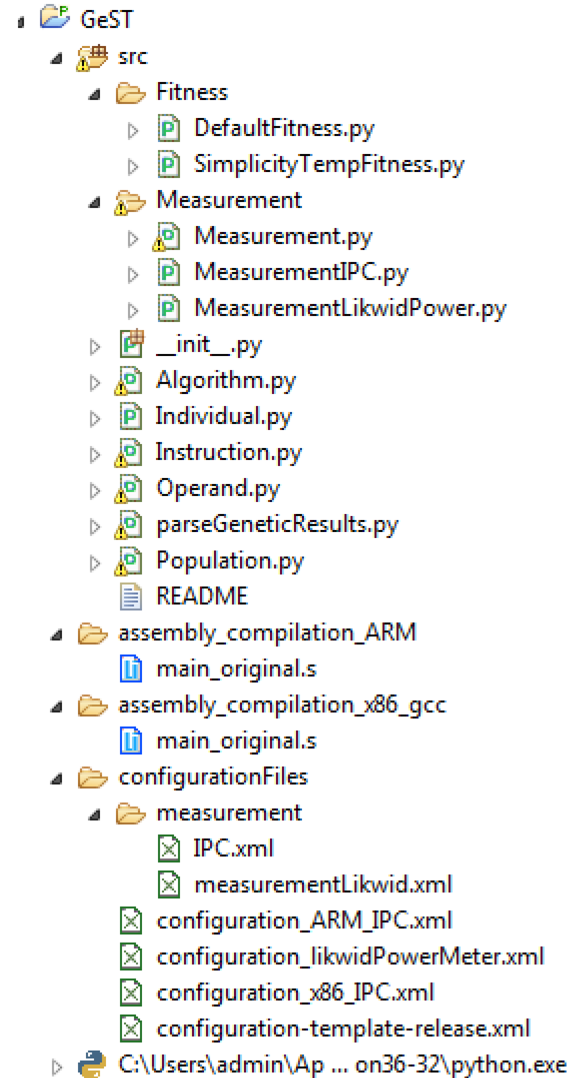
- GeST platform independent automatic stress-test generation framework
- Key strengths flexibility and extensibility – provides easy interface to the experimenter to build upon
- This work shows measurements on real-hardware but framework can be used as well for pre-silicon stress-test generation in conjunction with simulators/models
- GeST will be publicly available on https://github.com/toolsForUarch/GeST **and it's free :)**
- For future work augment GeST with more features e.g. adaptive mutation

THE END

Thank you for listening!

Backup

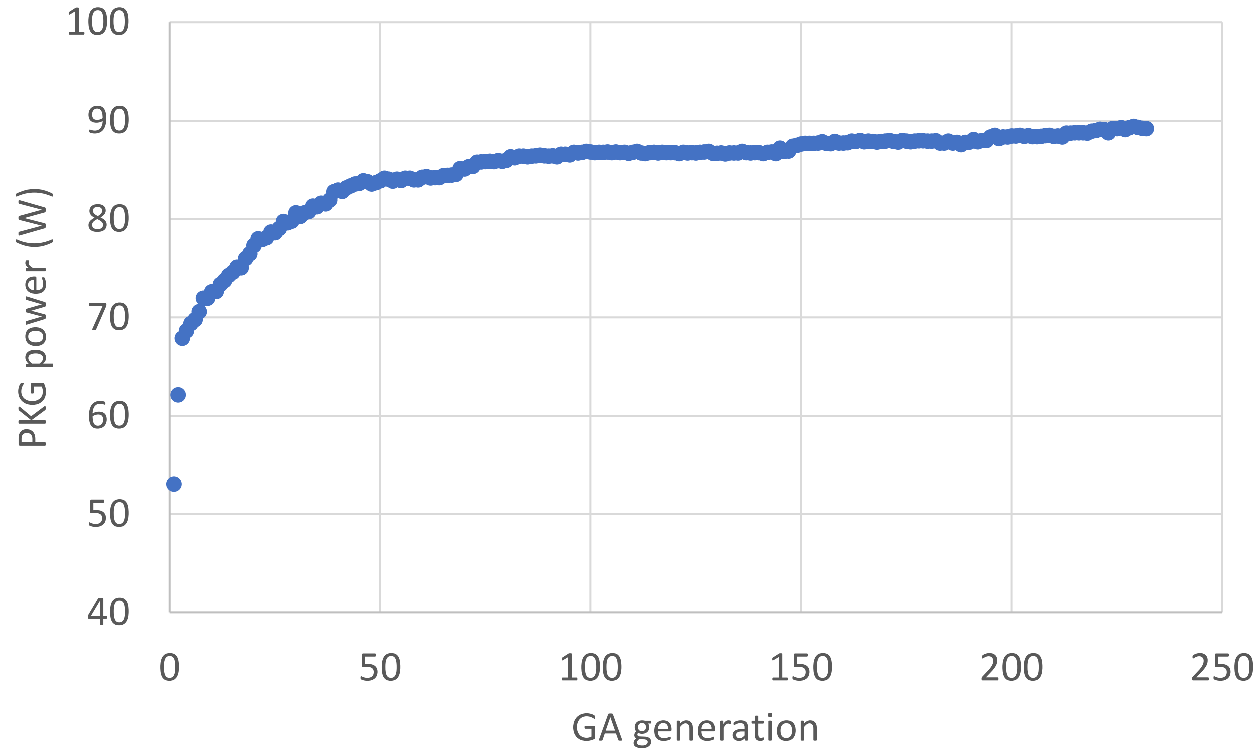
Structure



Conclusions - Main Features Overview

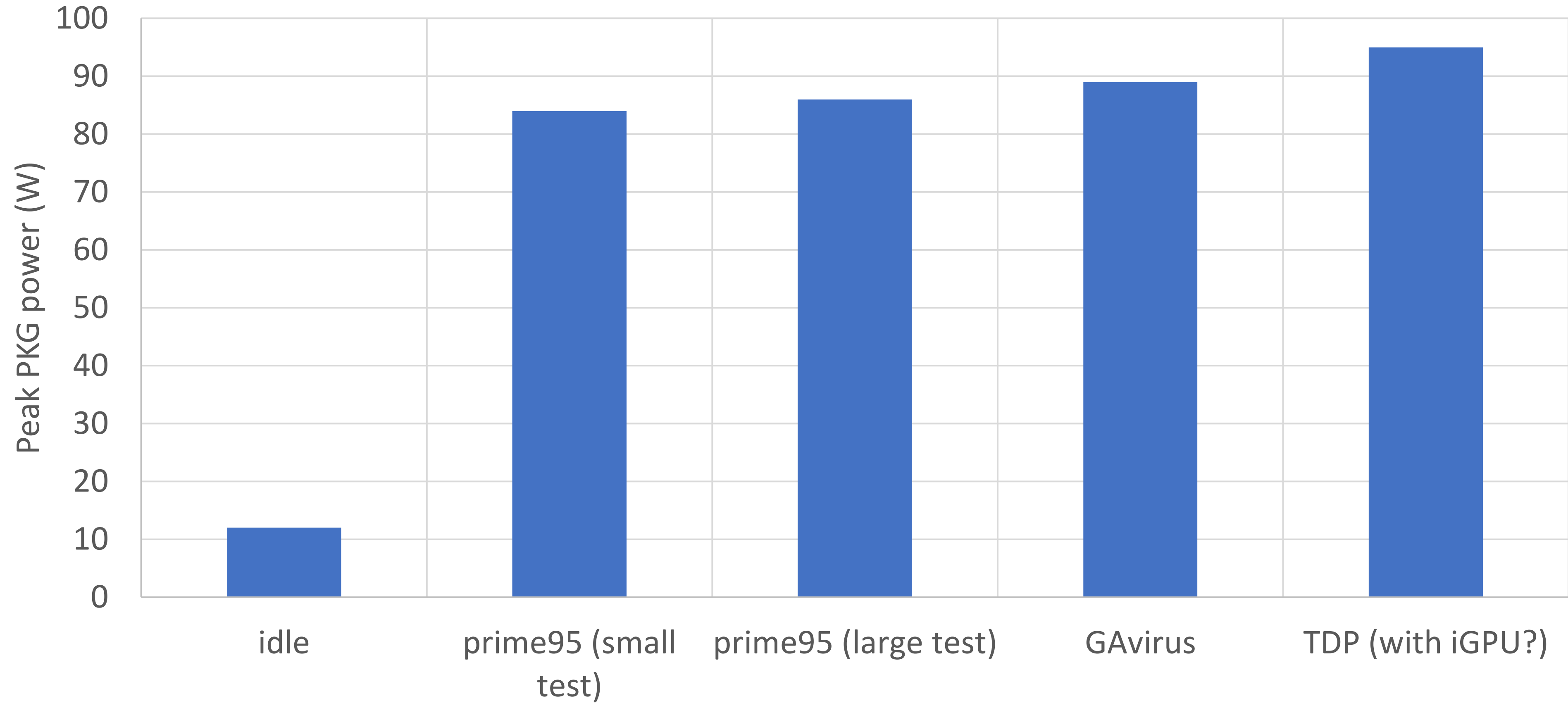
- Written in Python3, inputs defined in XML format
- Allows **setting GA parameters** such as crossover, mutation rate, population size etc.
- Allows specifying which **instructions, registers, memory ranges and even atomic code sequences** are available to the optimization engine
- Use dynamic class loading (class injection) to allow user to add in a **plug-and-play** fashion (by changing the input file) **custom measurement procedures and fitness functions**
- Virtually **compatible with any optimization** (power, voltage-noise etc) and **multi-objective fitness** functions
- Optionally force dependencies between instructions
- Optionally force specific instruction mix
- Stop a run and continue it later

GA optimization

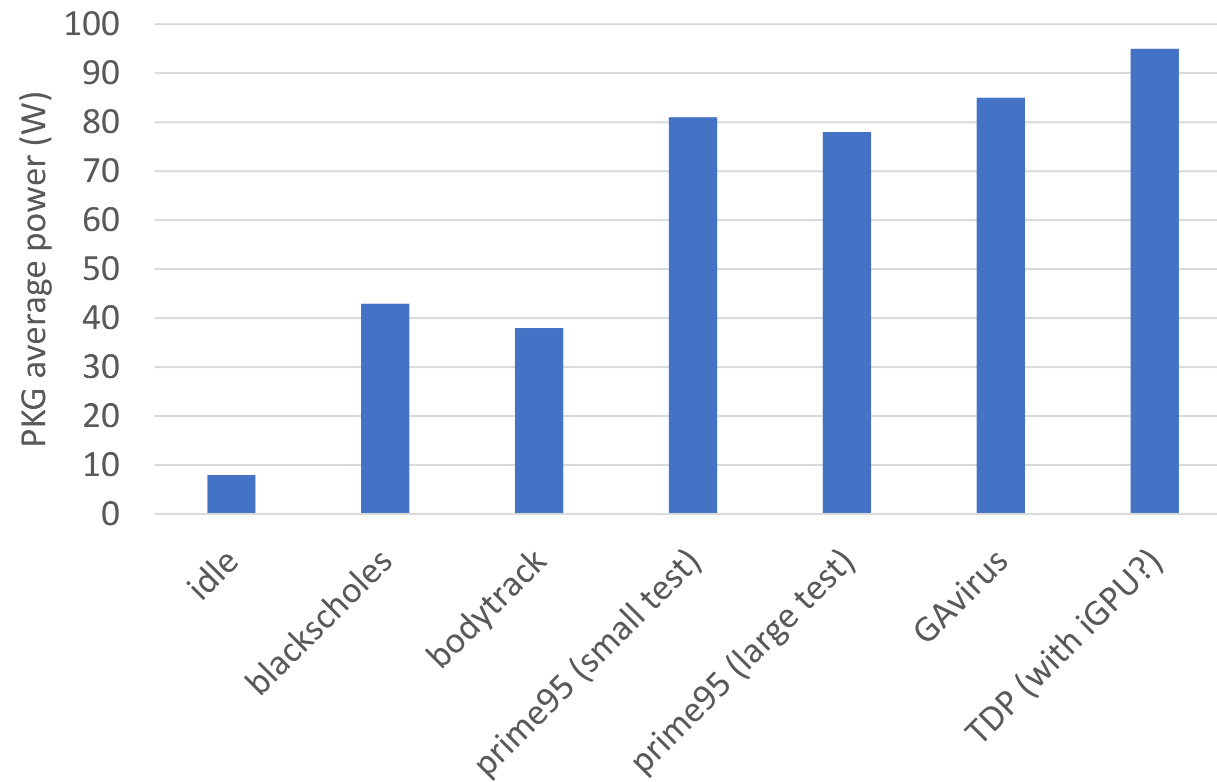


Best source code per generation is shown
Best of random population (1st) generates 53W
Latest populations generate 89-90W
Whole optimization process took ~16hours

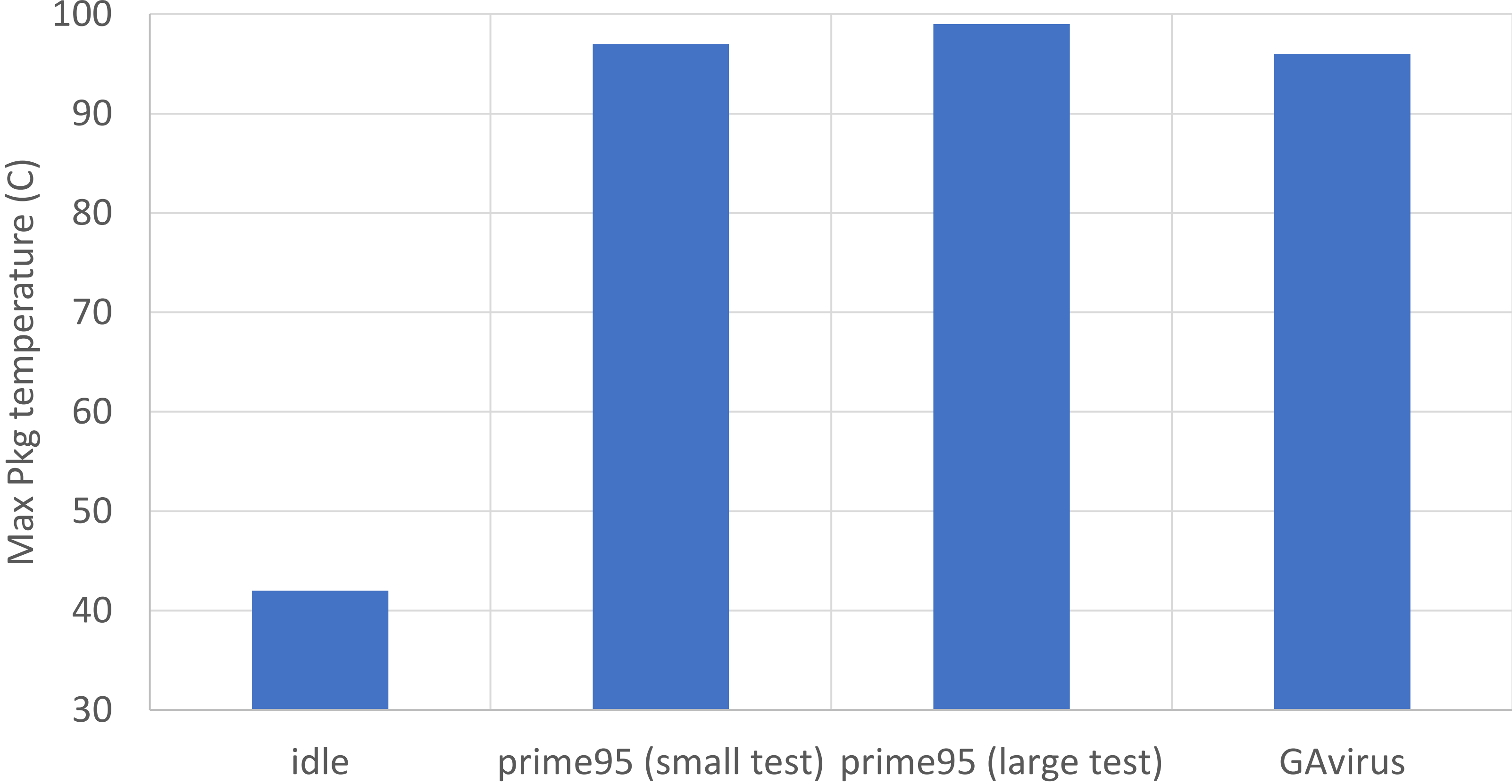
Peak Power Measurements



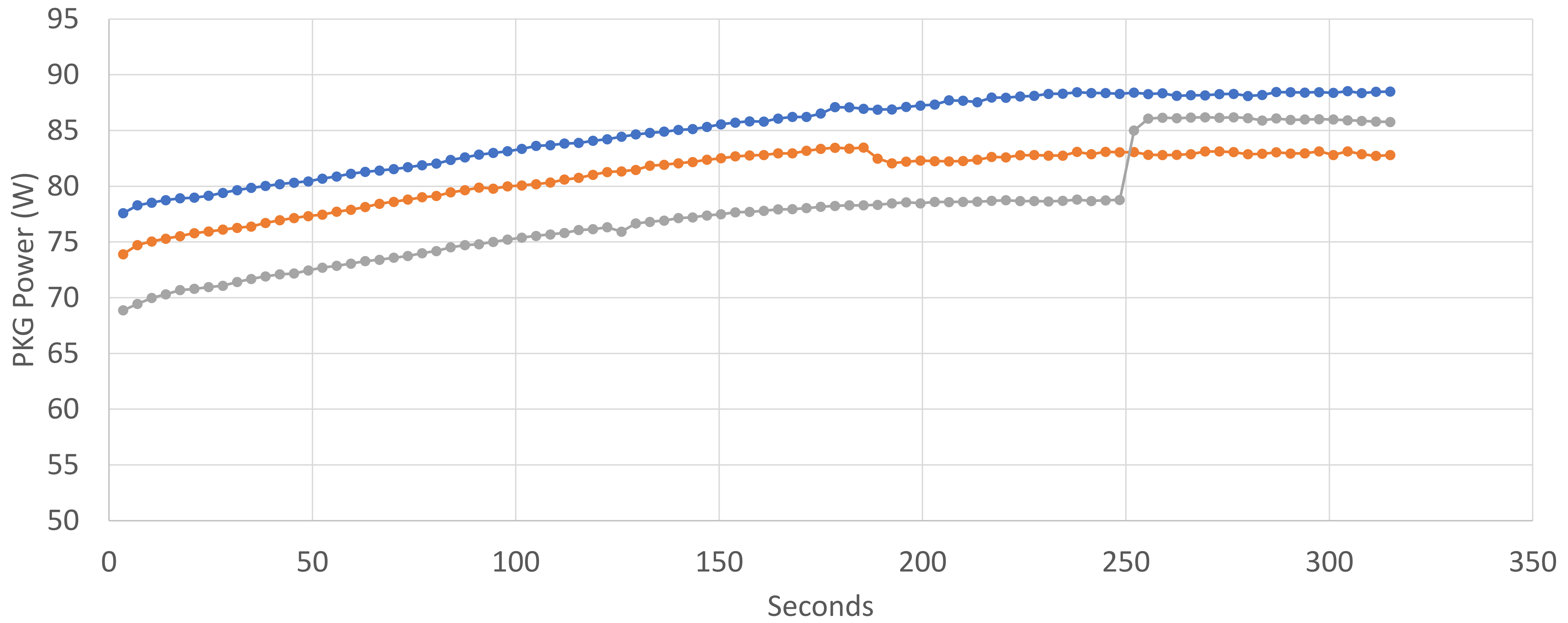
Average power measurements (including some parsec)



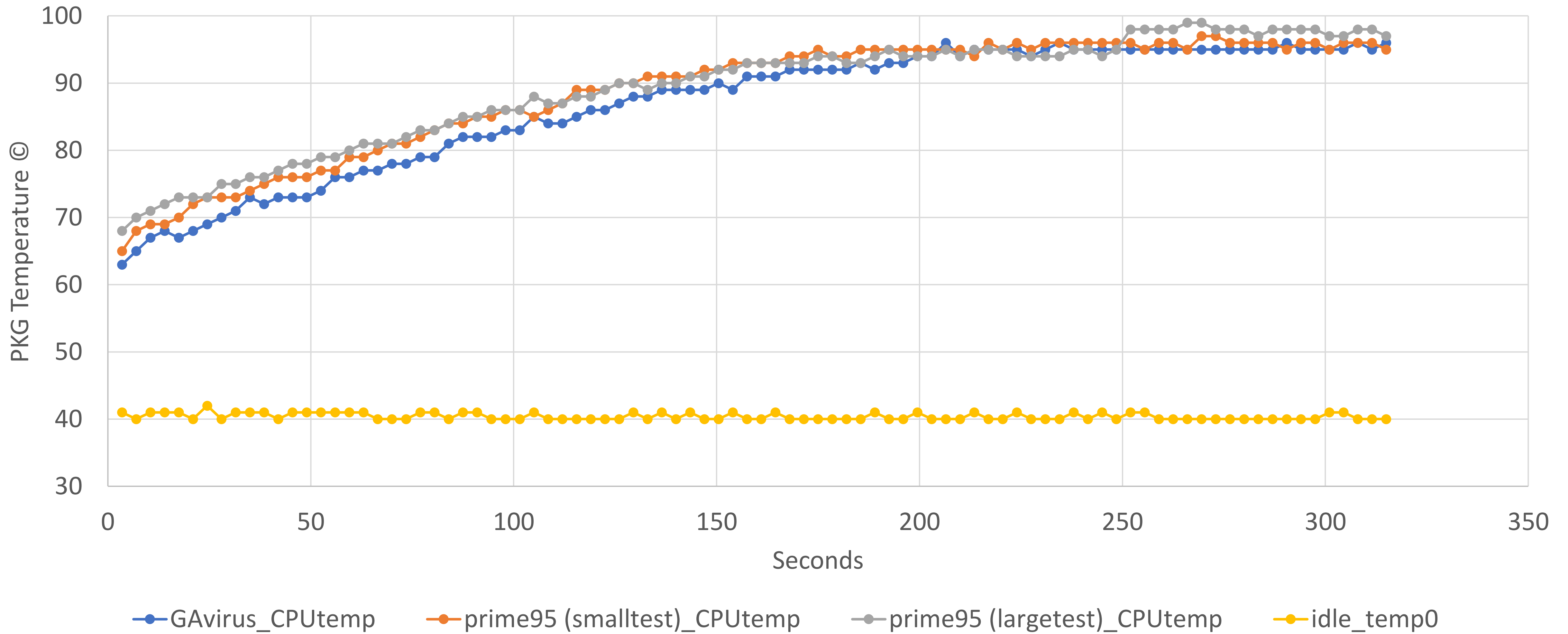
Peak Temperature Measurements



Power over time (GAvirus and Prime95)

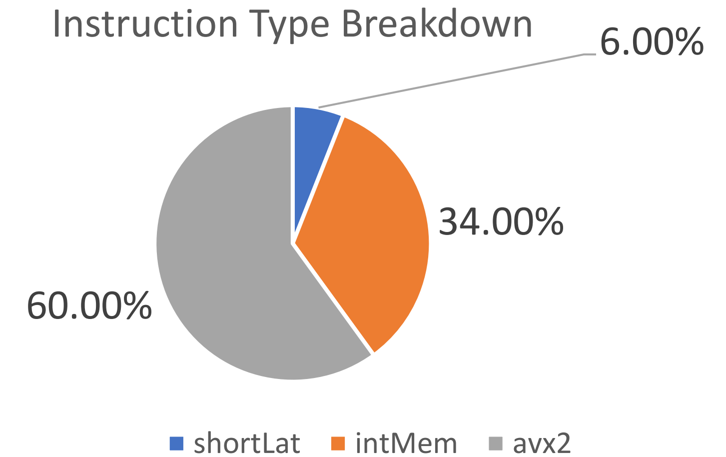


Temperature over time (GAvirus and Prime95)



GA virus characteristics

- Loop of 50 instructions
- Doesn't touch L2
- 60% of instructions are avx2
- 34% instructions are integer instructions with memory accesses
- 6% register only short latency integer instructions
- Long latency register-only instructions such as MUL and old floating-point instructions such as faddp were included in the optimization mix. But these instructions are not found in the GA virus. Probably GA figured out that these instructions do not help in raising the power consumption
- Virus IPC is 3.58
  - mprime (small) has 2.21, mprime (large) has 1.54 initially and ~after 5 minutes 2.14



GA virus source code and instruction breakdown

- Essentially the virus code is a sequence of various avx2 instructions, and some ads, moves, and one shift!
- Half of the user defined instructions are missing from the virus
- GA ended up using only 12 out of 24 user defined instructions
- VPSHUF that stresses the shuffling unit is not preferred by the GA
- Integer vector instructions are also not preferred by the GA
- As expected, long latency int instructions such as mul are also discarded by GA
- Fmulp and Faddp are also discarded

Instruction name	*Count*
MUL	0
SAR	1
ROR	0
VSUBPD	3
MOV_2ndMem	2
VMULPD	14
VPADDW	0
MOV	0
ADD_2ndMem	4
MUL_1stMem	0
VXORPD	2
MOV_1stMem	10
ADD	0
VPSHUFB	0
VPMULUDQ	0
FMUL	0
ADD_IM	1
CMP	0
MUL_IM	0
ADD_1stMem	1
SHL	1
VADDPD	6
VMAXPD	5
FADD	0

```
vmulpd %ymm9,%ymm13,%ymm11
vaddpd %ymm5,%ymm12,%ymm10
add %rsi,68(%rsp)
add %rax,116(%rsp)
mov 28(%rsp),%rbx
vsubpd %ymm15,%ymm14,%ymm6
vmulpd %ymm6,%ymm12,%ymm11
vmulpd %ymm2,%ymm8,%ymm2
vmaxpd %ymm4,%ymm1,%ymm7
add %rsi,96(%rsp)
mov 116(%rsp),%rdi
mov 20(%rsp),%rbx
vmaxpd %ymm12,%ymm1,%ymm9
vmulpd %ymm10,%ymm12,%ymm10
shl $31,%rsi
add 88(%rsp),%rbx
vmulpd %ymm4,%ymm0,%ymm10
vmulpd %ymm2,%ymm15,%ymm15
vmulpd %ymm15,%ymm1,%ymm0
mov 96(%rsp),%rsi
vaddpd %ymm5,%ymm5,%ymm15
mov 124(%rsp),%rbx
vsubpd %ymm11,%ymm11,%ymm4
vxorpd %ymm14,%ymm9,%ymm0
vaddpd %ymm2,%ymm5,%ymm5
vmulpd %ymm3,%ymm10,%ymm0
vmulpd %ymm4,%ymm12,%ymm10
vmulpd %ymm1,%ymm12,%ymm12
mov %rdx,28(%rsp)
vmulpd %ymm6,%ymm6,%ymm0
mov 68(%rsp),%rbx
vmulpd %ymm2,%ymm9,%ymm12
vaddpd %ymm7,%ymm15,%ymm11
vmaxpd %ymm4,%ymm0,%ymm11
mov 60(%rsp),%rax
vxorpd %ymm8,%ymm8,%ymm3
add %rbx,108(%rsp)
mov 80(%rsp),%rbx
vsubpd %ymm5,%ymm15,%ymm15
vmaxpd %ymm13,%ymm1,%ymm11
vaddpd %ymm8,%ymm12,%ymm6
vmulpd %ymm4,%ymm13,%ymm15
add $1646404055,%rdx
mov 52(%rsp),%rbx
vmulpd %ymm15,%ymm14,%ymm3
vmaxpd %ymm5,%ymm2,%ymm11
vaddpd %ymm1,%ymm7,%ymm15
mov 52(%rsp),%rax
mov %rbx,36(%rsp)
mov %rdx,88(%rsp)
```