code size reduction

Jan 26th 2021

Antitrust Policy Notice

RISC-V International

RISC-V is a free and open ISA enabling a new era of processor innovation through open standard collaboration. Born in academia and research, RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation.

We are a transparent, collaborative community where all are welcomed, and all members are encouraged to participate. We are a continuous improvement organization. If you see something that can be improved, please tell us. help@riscv.org

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone.

https://riscv.org/risc-v-international-community-code-of-conduct/

Agenda

1. Table Jump proposal
2. PUSH + MV proposal
3. Compiler optimisation and code-speed
4. Next meeting: 2nd Feb at 7am PDT

Table Jump

1. Replace (maybe) 256 JAL destinations with a 16-bit encoding to a jump table
2. the table entry is MTBLJALVEC + (XLEN/8) * imm_operand
3. The Jump table contains a pointer to the function, and the LSBs state which link register to use
4. Two other options are
    1. emulation mode – put link address and table index into temporary registers, and jump to MTBLJALVEC
    2. Vector table mode – jump directly to the code, jump to MTBLJALVEC + imm_operand * tablescale
        1. table scale is 8-bytes to 4096-bytes

https://github.com/riscv/riscv-code-size-reduction/blob/master/ISA%20proposals/Huawei/table%20jump.adoc

Dynamically linked libraries

Linux uses DLLs, so it needs to dereference the targets of JALs outside the current ELF e.g. a call to GLIBC:

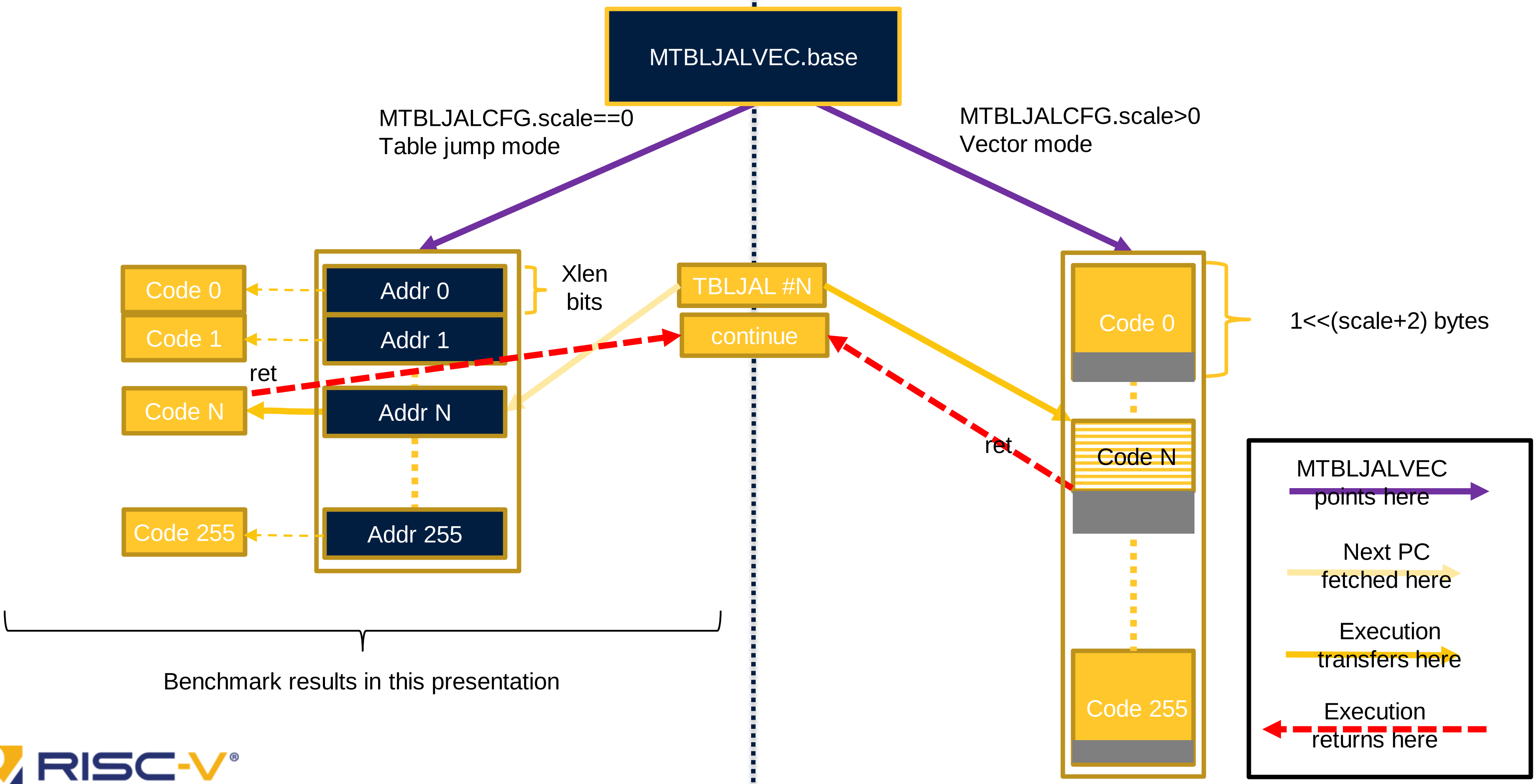
```
16da:       fa6ff0ef            jal         ra,e80 <symlink@plt>

0000000000000e80 <symlink@plt>:
    e80:    00003e17            auipc       t3,0x3
    e84:    198e3e03            ld          t3,408(t3) # 4018 <symlink@GLIBC_2.27>
    e88:    000e0367            jalr        t1,t3
    e8c:    00000013            nop
```

TBLJAL #n will map to something different in each ELF, so they need to be disambiguated.
- emulation mode should handle this, as the behaviour is software defined.
- rv64 has no C.JAL so the benefit of C.TBLJAL will be very high
- jump table / vector mode can work if MTBLJALVEC is set correctly for the current DLL

TBLJAL Benchmark Results

26/01/2021


MTBLJALVEC.base
MTBLJALCFG.scale==0
Table jump mode
MTBLJALCFG.scale>0
Vector mode
Code 0
Code 1
Code N
Code 255
Addr 0
Addr 1
Addr N
Addr 255
Xlen
bits
ret
TBLJAL #N
continue
ret
Code 0
1<<(scale+2) bytes
Code N
Code 255
MTBLJALVEC
points here
Next PC
fetched here
Execution
transfers here
Execution
returns here
Benchmark results in this presentation

Get all function calls and count the number each is used

Static Analysis (Passes through the code, find and count the frequency of each JAL,JALR..).

Go through the entries and eliminate all entries that wont gain from substitution (JAL,J) < 3

```
001f8317 auipc t1,0x1f8 e084c2: 18a302e7 jalr t0,394(t1) # 1000648 <__riscv_save_0>
18a302e7 jalr t0,394(t1) # 1000648 <__riscv_save_0>
```

```
e084be: xxxx tbljal #x ;#<mapped to __riscv_save_0>
```

This would saving 6-bytes, but would require a single table entry of 4 bytes, so even a single entry would saves 2 bytes

```
f61ff0ef jal ra,e08432
```

```
f61ff0ef : xxxx tbljal
```

This would saving 2-bytes, but would require a single table entry of 4 bytes, so we would need at least 3 calls for it to be beneficia

Change the weight of JALR, and JR entries to be 3*Count

When selecting the top 128 or 256 entries, we need to give priority to JALR, JR because their replacement save more space, thus their weight is multiplied with 3

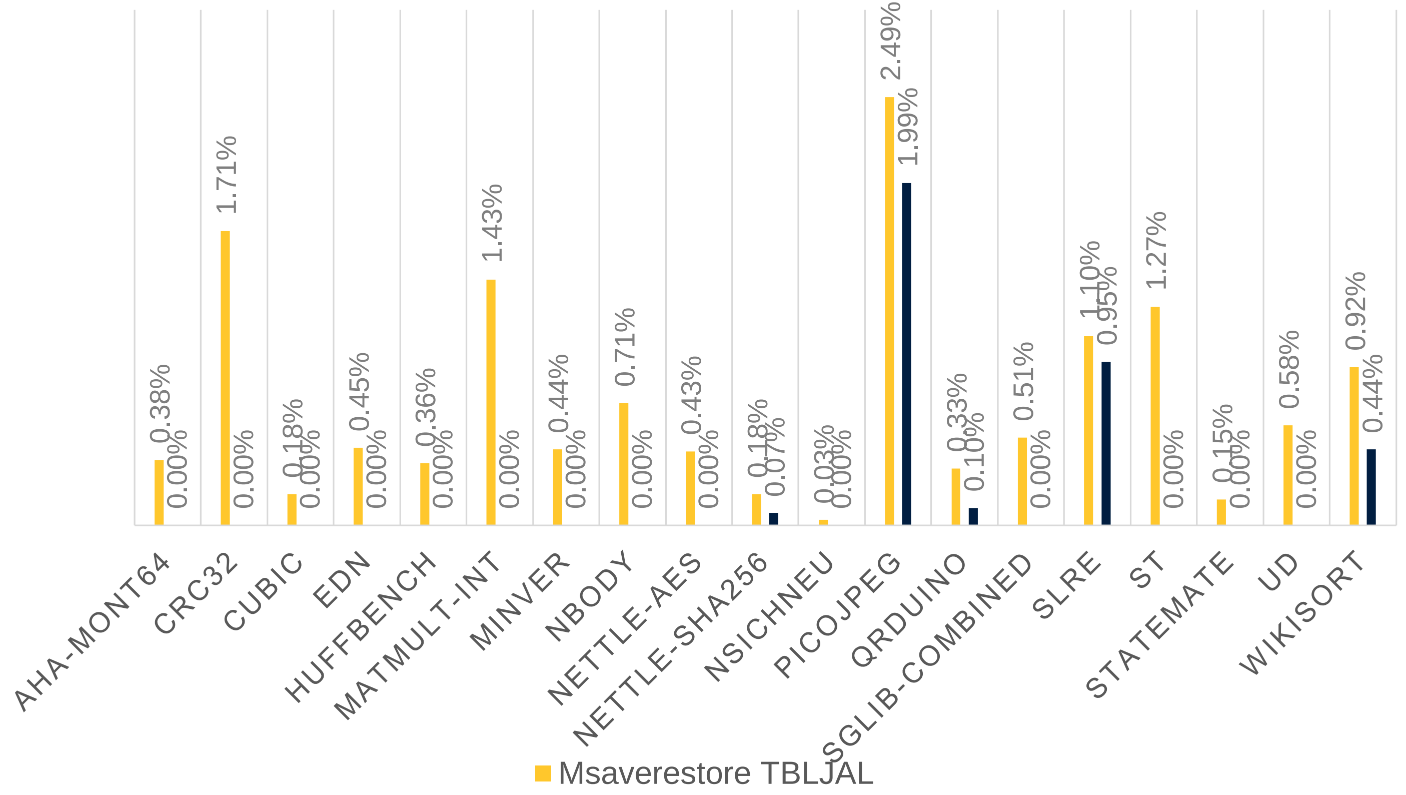
Get the most common (128,256 entries)

Get the top X entries

Calculate the gains

Calculate the gains, and compensate for the table size

EMBENCH SAVINGS



	Save Restore	Push Pop
aha-mont64	1	0
crc32	1	0
cubic	1	0
edn	1	0
huffbench	1	0
matmult-int	1	0
minver	1	0
nbody	1	0
nettle-aes	1	0
nettle-sha256	2	1
nsichneu	1	0
picojpeg	13	8
qrduino	4	2
sglib-combined	2	0
slre	2	1
st	1	0
statemate	1	0
ud	1	0
wikisort	8	4

Allocated Table Size

Bigger Benchmarks !

Save Restore	128 Upper Limit		256 Upper Limit		No Upper Limit	
	Table Size	Saving #1	Table Size	Saving #1	Table Size	Saving #1
huawei_iot_application	128	9.43%	256	9.90%	608	10.24%
huawei_iot_protocol	128	6.74%	256	7.37%	2629	9.11%
zephyr_central	128	6.76%	220	7.23%	142	6.83%
zephyr_peripheral	128	6.90%	142	6.83%	220	7.23%

Push Pop	128 Upper Limit			256 Upper Limit			No Upper Limit		
	Table Size	Saving #1	Saving #2	Table Size	Saving #1	Saving #2	Table Size	Saving #1	Saving #2
huawei_iot_application.elf	128	5.52%	10.73%	256	5.96%	11.15%	604	6.28%	11.45%
huawei_iot_protocol.elf	128	3.66%	7.64%	256	4.24%	8.20%	2632	5.98%	9.87%
zephyr_central.elf	117	3.13%	8.29%	117	3.13%	8.29%	117	3.13%	8.29%
zephyr_peripheral.elf	128	3.77%	8.28%	209	4.06%	8.57%	209	4.06%	8.57%

** Saving #1 measures the incremental saving of adding tbljal to the push/pop compiler output.

** Saving #2 is the cumulative effect of push/pop and tbljal.

Table Size Vs Saving (IOT_Application)

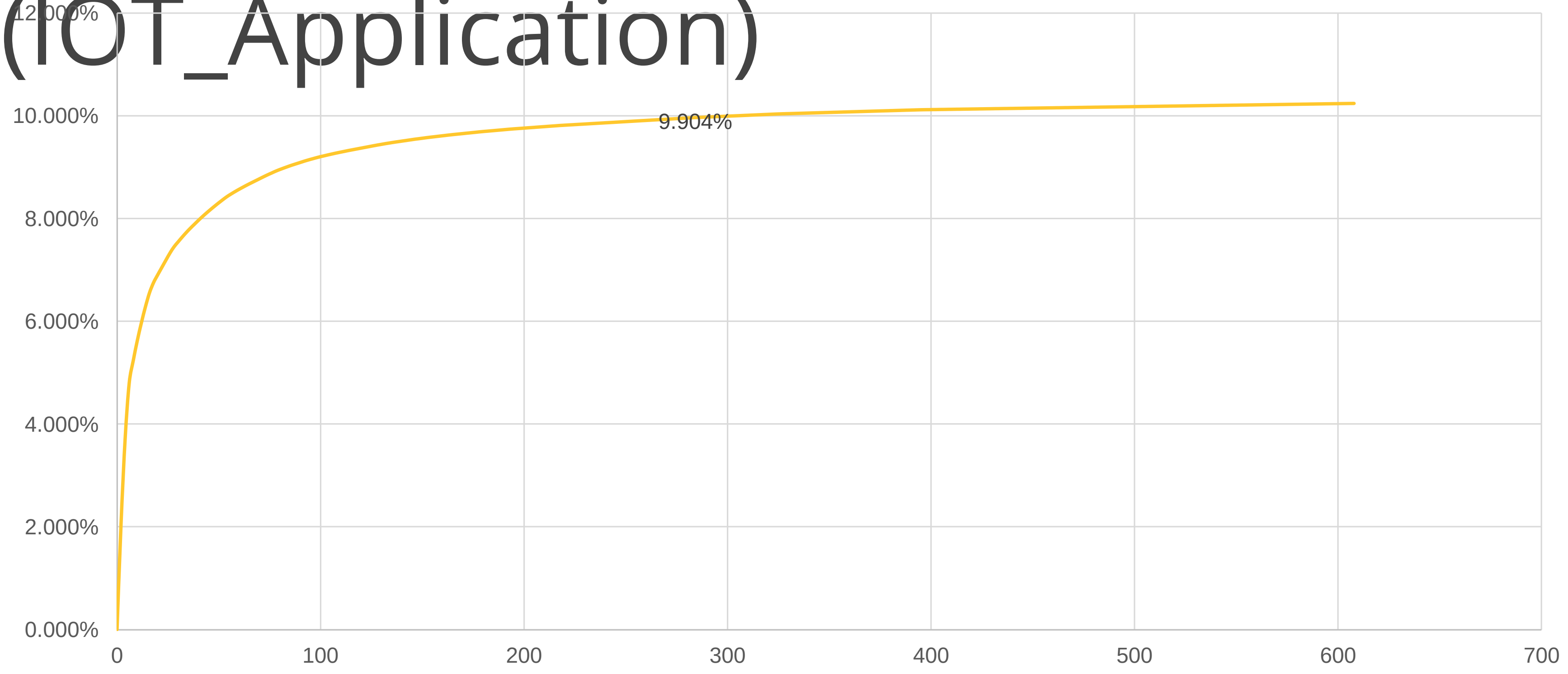
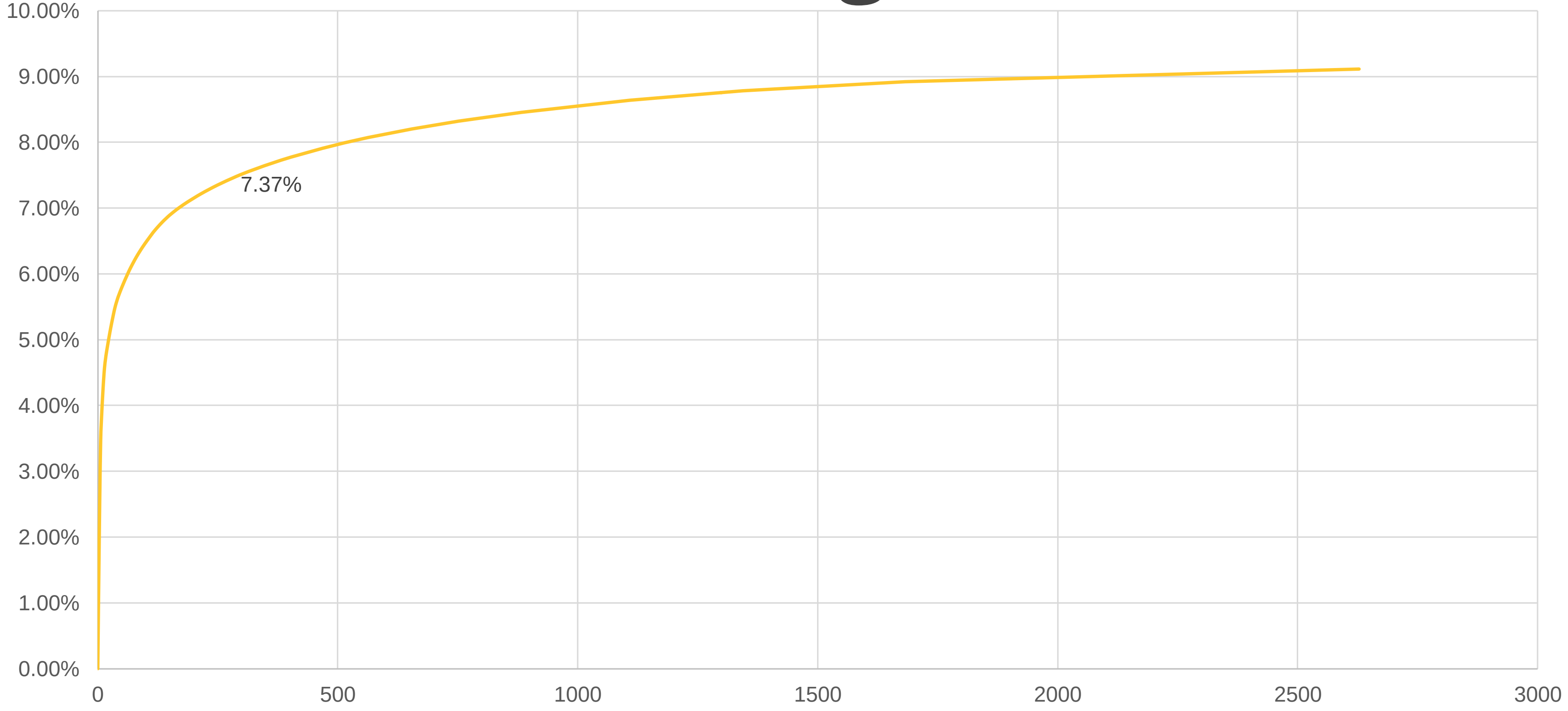


Table Size Vs Saving (IOT_Protocol)



Thank You