T1 - The Long RISC-V Vector Machine Generator

Jiuyang Liu@HUST&Chips Alliance
Qinjun Li@PLCT/ISCAS
Yunqian Luo@Tsinghua University
And other PLCT-CAAT interns and FTEs

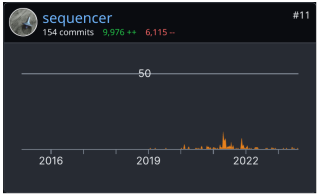
Aug 25, 2023

Bio

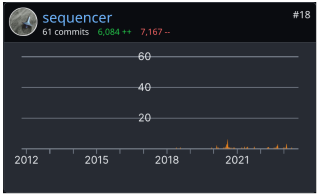
- Jiuyang Liu
- liu@jiuyang.me
- sequencer@GitHub
- PGP Key: 0×8D7B5A

Open Source Chip Developer

- Chisel Developer



- RocketChip Maintainer



- Chips Alliance TSC Member

TSC voting members and Graduated projects

Per the Technical Charter, on top of any members appointed in the initial bootstrap phase, the TSC will be comprised of one representative from each Graduated project.

The current members of the CHIPS Alliance TSC are:

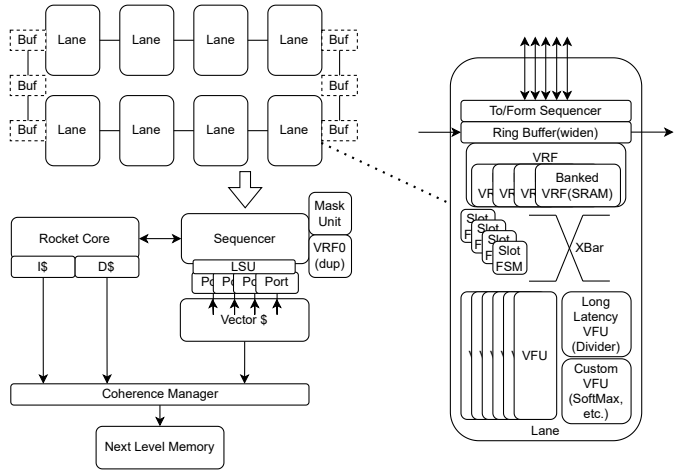
Name	GitHub	Affiliation	Project
Henry Cook (chair)	hcook	SiFive	(chair)
Michael Gielda	mgielda	Antmicro	(mktg)
Henner Zeller	hzeller	Google	Verible
Alain Dargelas	alainmarcel	unaffiliated	Surelog/UHDM
Mehdi Saligane	msaligane	University of Michigan	OpenFASOC
Tom Gorochowik	tgorochowik	Antmicro	SV tests / FPGA tool perf
Maciej Kurc	mkurc-ant	Antmicro	FPGA Interchange
David Kehlet	dkehlet	Intel	AIB
Matt Cockrell	mcockrell-google	Google	RISC-V DV
Tom Michalak	tmichalak	Antmicro	F4PGA
Mikhail Moiseev	mikhailmoiseev	Intel	SystemC compiler
Jiuyang Liu	sequencer	PLCT Lab, ISCAS	Rocket Chip

The Technical Charter describes the composition of the TSC.

Topic Today - T1

- Micro architecture show-off;
- Methodlogy to tune RISC-V Vector performance;
- My 50 cents to the future of RISC-V Vector in HPC;

https://github.com/chipsalliance/t1



The Real Vector Machine

- Large DLEN with multiple VFU
- Large VRF
- Support instruction chaining

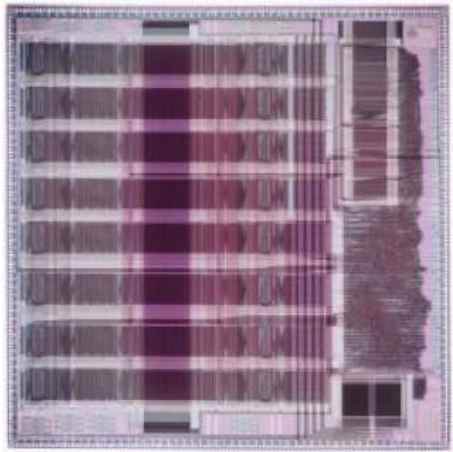


Figure: T0 by Prof. Krste Asanović

Pioneers - Xiangshan

Characteristics:

- Dedicated vector pipeline and rename unit
- No chaining
- Share float registers with vector registers
- Dedicate VFU pipelines
- Share load store unit

Small DLEN, improve code density, IF and BFU friendly.

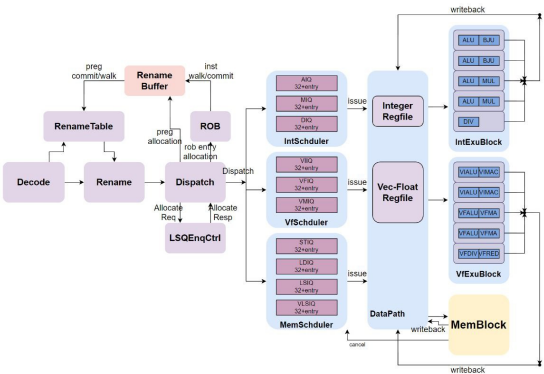


Figure: XiangShan Vector main pipeline

Pioneers - X280

Characteristics:

- Dedicated vector pipeline and flop based registers;
- Configurable to 512bits DLEN;
- Access L1D$ and L2$ simultaneously;

Dual issue scalar core with large DLEN but only one lane.

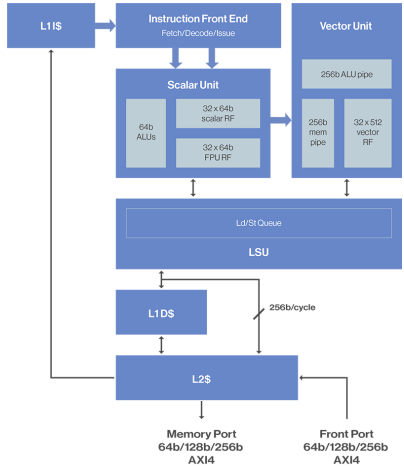


Figure: SiFive X280 Architecture

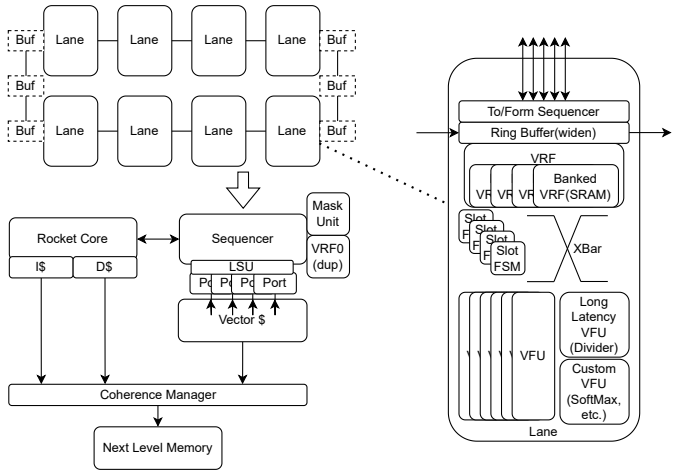
Pioneers - Others

There are other pioneers, not listed because academic/closed-source/no silicon products:

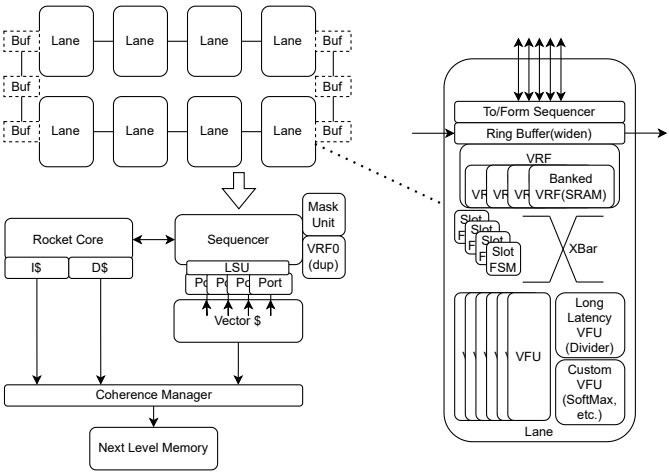
- ETH Ara: multiple lane but no chaining support, interesting toy project.
- UCB Hwacha: Non-standard Vector implementation, before RISC-V Vector 1.0, interesting for decoupled vector architecture.
- Semidynamics Vector: multiple lane with renaming, looking forward to its customer products.
- T-Head: C908 is too weak to say it is the vector processor.
- StarFive/Nuclei/Andes/Rivai: Wait for products, god bless them.

Architecture Overview

- Rocket-based RV32 core with OoO write-back, w/o MMU, w/o fast interrupt
- Ring-based interconnection for widen and narrow instruction (configurable buffer for ring)
- Single unit on Sequencer for non data-parallelism instructions, e.g. mask, ffo, reduce.
- Duplicated VRF0 on Sequencer, used for mask unit to reduce the bandwidth usage from Sequencer to lanes
- Single banked-LSU with strong outstanding slots.



Architecture Principle



- High-throughput design with scaleable lane size, RF size, VFU type and connection matrix;
- banked SRAM VRF;
- Load to VFU to VFU to Store chaining ability;
- HPC only, slow external interrupt w/o mmu;

Bandwidth Performance Points

- VRF - Vector Register Field
- VFU - Vector Function Unit
- LSU - Load Store Unit

VRF

- Chip frequency bottleneck is VRF SRAM.
- Flop-based VRF v.s. SRAM based VRF.

T1 chooses the banked SRAM.

VFU

Tradeoff by:

- Cell Area
- Routing Area to and from VRF
- Power

Methodlogy:

- Full custom design for function cells;
- Limit function unit to specific cells;
- Token + Asynchronous function circuit;
- Retime with commercial EDA tools

T1 only use Retime for now :(we need more human resources

LSU

No magic for LSU, In our design, we have already support these feature:

- LSU port + banked next level cache;
- Instruction level memory interleaving;
- Different LSU logic for handling unit stride, stride, indexed load store;
- Merge multiple uop into cacheline size to reduce the memory accessing overhead in the next level cache;
- Intensive outstanding strategy, 3 MSHR for each bank, support hundreds of outstanding;

RISC-V Memory in the future

However, the Memory is the non-trivial part, which may still need a lot of feature to explorer:

- MMU
- Prefetch and Cache for Sparsity
- Configuration for HBM/DDR

The Trade-off

- Balance the bandwidth among VRF, VFU and LSU. Keeping them as busy as possible.
- Hit the memory bandwidth, and keep everything as busy as possible.

Performance

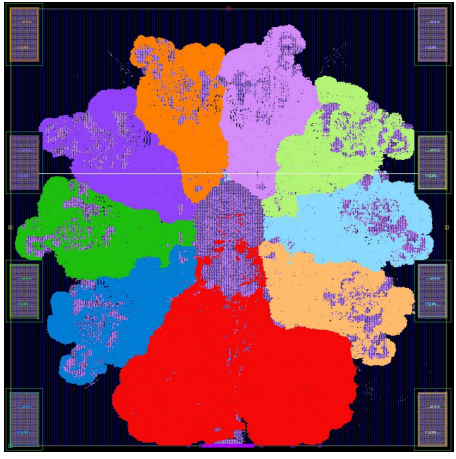
The current result(untuned yet) is a little awkward,

#bank	cycle/elem	memop/cycle	vrf writes/cycle	avg chaining size
2	2.03	1.48	0.25	0.79
4	1.70	1.76	0.31	0.94
8	1.65	1.81	0.30	0.94
16	1.65	1.81	0.30	0.94

Table: matmul 128×128

The Layout

Based on our auto floorplaner, the test layout is:



Tuning Step

Thanks Chisel makes all parameters being configurable!

- Check the bandwidth that can be provided by SoC;
- Check the frequency that can be provided by SRAM;
- Based on the bandwidth and frequency plus architecture level info to calculate the required SRAM ports and VFU counts;
- Based on the workload(specifically the tiling size) and decide the VLEN;
- Based on physical design constraints, decide the lane counts and VFU counts inside each Lane;

How to tune???

For long vector architecture, we are always bitten by memory wall:

In Hardware:

- Sparse Vector Cache.
- Interconnect protocol friendly (~~TileLink~~)
- Memory protocol friendly.

In Compiler:

- Increase the n in Load to VFU $\times$ n to Store
- uArch aware to generate chainable instructions as much as possible

In Software:

- friendly to tiling, no wrap, no SM.

In Architecture:

- Add custom vector instructions to operate on elements to reduce VRF/LSU bandwidth waste.

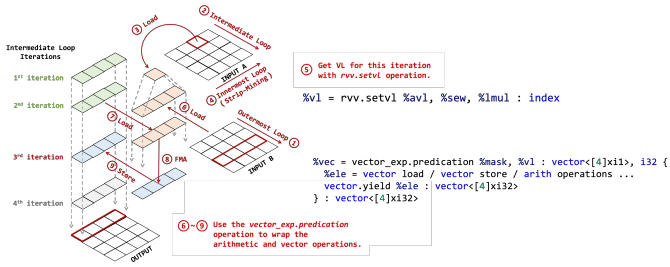
BuddyCompiler + MLIR to speed up compiler exploration

Burn old world down:

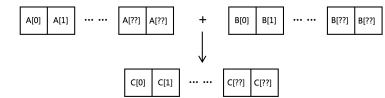
- ~~limited programming model~~
- ~~limited loop nest optimization~~
- ~~limited on CUDA~~

To make it friendly:

- rewrite high-level operations in MLIR.



Vectorization Approaches



Get the application vector length (d) at runtime

Mask-Based Approach

```
Tail = getTail(d)
Loop:
  if (not Tail)
    vector load
    vector add
    vector store
  else
    calculate mask
    masked load
    masked add
    masked store
  end if
End loop
```

Strip-Mining Approach

```
AVL = d
While(AVL > 0):
  do:
     vl = setvl AVL , LMUL , SEW
     vector load vl
     vector add vl
     vector store vl
     AVL = AVL - vl
End
```

Future Work

- MMU support;
- FGMT scalar core for handling interrupt during vector SIMD being running;
- MLIR Sparse Compiler for RVV;
- Custom Instruction Support;

The Future of Vector is unlimited.

Processor (year)	Vector clock rate (MHz)	Vector registers	Elements per register	Vector arithmetic units	Vector load-store units	Lanes
Cray-1 (1976)	80	8	64	6: FP add, FP multiply, FP reciprocal, integer add, logical, shift	1	1
Cray X-MP (1983)	118	8	64	6: FP add, FP multiply, FP reciprocal, integer add, logical, shift, population count/parity	2 loads 1 store	1
Cray Y-MP (1988)	166	8	64	6: FP add, FP multiply, FP reciprocal, integer add, logical, shift, population count/parity	2 loads 1 store	1
Cray-2 (1985)	244	8	64	5: FP add, FP multiply, FP reciprocal/sqrt, integer add/shift/population count, logical	1	1
Fujitsu VP100 (1982)	133	8-256	32-1024	3: FP or integer add/logical, multiply, divide	2	1
Fujitsu VP200 (1982)	133	8-256	32-1024	3: FP or integer add/logical, multiply, divide	2	2
Hitachi S810 (1983)	71	32	256	4: FP multiply-add, FP multiply/divide-add unit, 2 integer add/logical	3 loads 1 store	1
Hitachi S820 (1983)	71	32	256	4: FP multiply-add, FP multiply/divide-add unit, 2 integer add/logical	3 loads 1 store	2
Convex C-1 (1985)	10	8	128	2: FP or integer multiply/divide, add/logical	1	1 (64 bit), 2 (32 bit)
NEC SX/2 (1985)	167	8+32	256	4: FP multiply/divide, FP add, integer add/logical, shift	1	4
Cray C90 (1991)	240	8	128	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	2
Cray T90 (1995)	460	8	128	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	2
NEC SX/5 (1998)	312	8+64	512	4: FP or integer add/shift, multiply, divide, logical	1	16
Fujitsu VPP5000 (1999)	300	8-256	128-4096	3: FP or integer multiply, add/logical, divide	1 load 1 store	16
Cray SV1 (1998)	300	8	64 (MSP)	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	1 load-store	2
SV1ex (2001)	500	8	64 (MSP)	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	1 load-store	8 (MSP)
VMIPS (2001)	500	8	64	5: FP multiply, FP divide, FP add, integer add/shift, logical	1 load-store	1
NEC SX/6 (2001)	500	8+64	256	4: FP or integer add/shift, multiply, divide, logical	1	8
NEC SX/8 (2004)	2000	8+64	256	4: FP or integer add/shift, multiply, divide, logical	1	4
Cray X1 (2002)	800	32	64	3: FP or integer, add/logical, multiply/shift, divide/square root/logical	1 load 1 store	2 8 (MSP)
Cray XIE (2005)	1130	32	64	3: FP or integer, add/logical, multiply/shift, divide/square root/logical	1 load 1 store	2 8 (MSP)

Thanks

- Chisel
- RocketChip community for rocket core generator.
- SiFive for inclusive cache generator.
- Buddy Compiler and MLIR community for software generator.
- Krste's PhD thesis for inspring us.
- YSYX tapeout shuttle.
- PLCT Lab for funding.