International Workshop on Arm-based HPC: Practice and Experience (IWAHPCE-2024)

Introducing software pipelining for the A64FX processor into LLVM

Masaki Arai †, Naoto Fukumoto †, Hitohi Murai ‡

† Fujitsu Limited

‡ RIKEN R-CCS

Outline

Background

- Software pipelining is an essential optimization for accelerating High-Performance Computing applications on CPUs
- Although open source compilers such as GCC and LLVM have implemented software pipelining, it is underutilized for the AArch64 architecture
- We have implemented software pipelining for the A64FX processor on LLVM to improve this situation

Background (A64FX)

- The A64FX is an out-of-order superscalar processor designed for HPC, compliant with the ARMv8-A architecture profile
- The A64FX supports the Scalable Vector Extension(SVE) instructions, a vector extension of the ARM instruction set architecture
- The A64FX supports 128, 256, and 512-bit SVE vector lengths
- The Fugaku supercomputer uses the A64FX

Current Status and Issues of LLVM

- LLVM 17 has a MachinePipeliner pass as an optimization pass to perform software pipelining
- The architectures using the MachinePipeliner pass are ARM, Hexagon, and PowerPC, and AArch64 is not supported
- The following issues exist regarding the optimization of HPC applications
  - Swing Modulo Scheduling(SMS) algorithm
  - Phase Ordering Problem
  - No Modulo Variable Expansion(MVE)

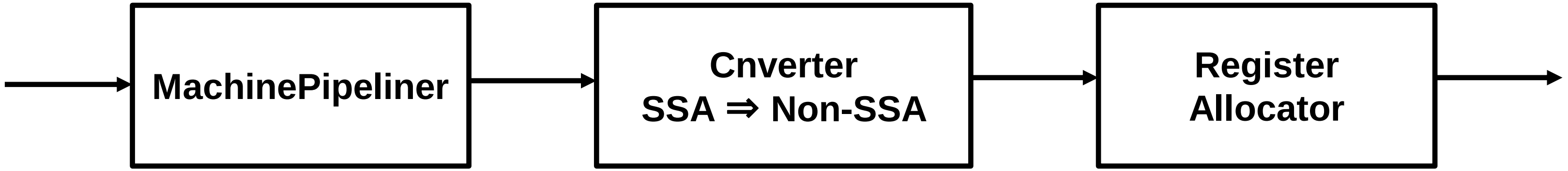
Current Status and Issues of LLVM

FUJITSU

- The algorithm used by MachinePipeliner is the Swing Modulo Scheduling(SMS) algorithm, characterized by short-time optimization and register-constraint-aware kernel generation
- Previous research[*] has shown that the Iterated Modulo Scheduling(IMS) algorithm produces better results than SMS for complex architectures

[*] Codina et al., A Comparative Study of modulo Scheduling Techniques.(ICS ’02)

Current Status and Issues of LLVM



- **MachinePipeliner generates optimization results in SSA form**
- **PHI instructions in SSA form are converted to COPY instructions**
- **These COPYs may remain after register allocation**
- **Register allocator may introduce spill codes**

⇒ These COPYs and spill codes ruin optimization results !

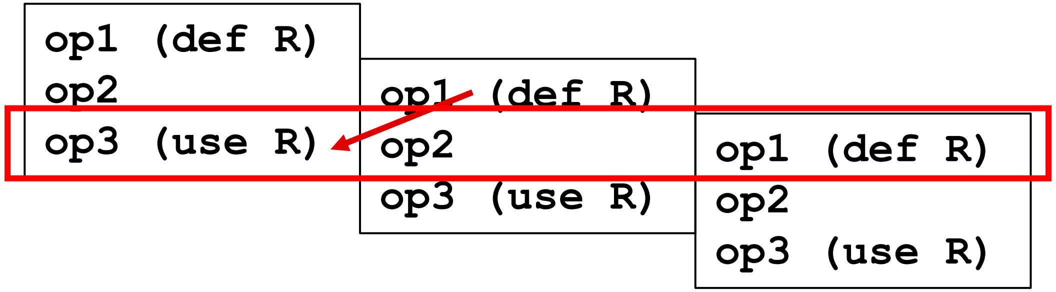
Current Status and Issues of LLVM

FUJITSU

- **Modulo Variable Expansion(MVE)**
  - **Sofware pipelining generates Modulo Reservation Table**

```
op1  (def R)
op2
op3  (use R)
```

  - **Wrong result kernel without MVE**



```
op1  (def R)
op2
op3  (use R)
```

```
op1  (def R)
op2
op3  (use R)
```

```
op1  (def R)
op2
op3  (use R)
```

⇒ One register R cannot hold the required value

Current Status and Issues of LLVM

- **Modulo Variable Expansion(MVE)**
  - **Correct result kernel using MVE(2 unroll)**

`op1 (def R)`			
`op2`	`op1 (def R)`		
`op3 (use R)`	`op2`	`op1 (def R)`	
	`op3 (use R)`	`op2`	`op1 (def R)`
		`op3 (use R)`	`op2`
			`op3 (use R)`

⇒ This kernel does not destroy register values in different iterations

Current Status and Issues of LLVM

- MachinePipeliner does not perform Modulo Variable Expansion(MVE)
- That means additional PHI instructions are required to preserve the meaning of the program
- And that means adding more COPYs

Software pipelining implementation for the A64FX

FUJITSU

- The brief summary:
  - We adopt Iterated Modulo Scheduling as a scheduling algorithm
  - Our implementation extends the LLVM scheduling model for the A64FX
  - We perform instruction scheduling in non-SSA form
  - Our implementation applies Modulo Variable Expansion if necessary
  - We perform register allocation to the kernel part
  - We have introduced countermeasures for register shortages due to register allocation

Software pipelining implementation for the A64FX

- Previous research[*] has shown that the Iterated Modulo Scheduling(IMS) algorithm produces better results than SMS for complex architectures
- The A64FX is a more complex CPU than the complex architectural model used in [*]

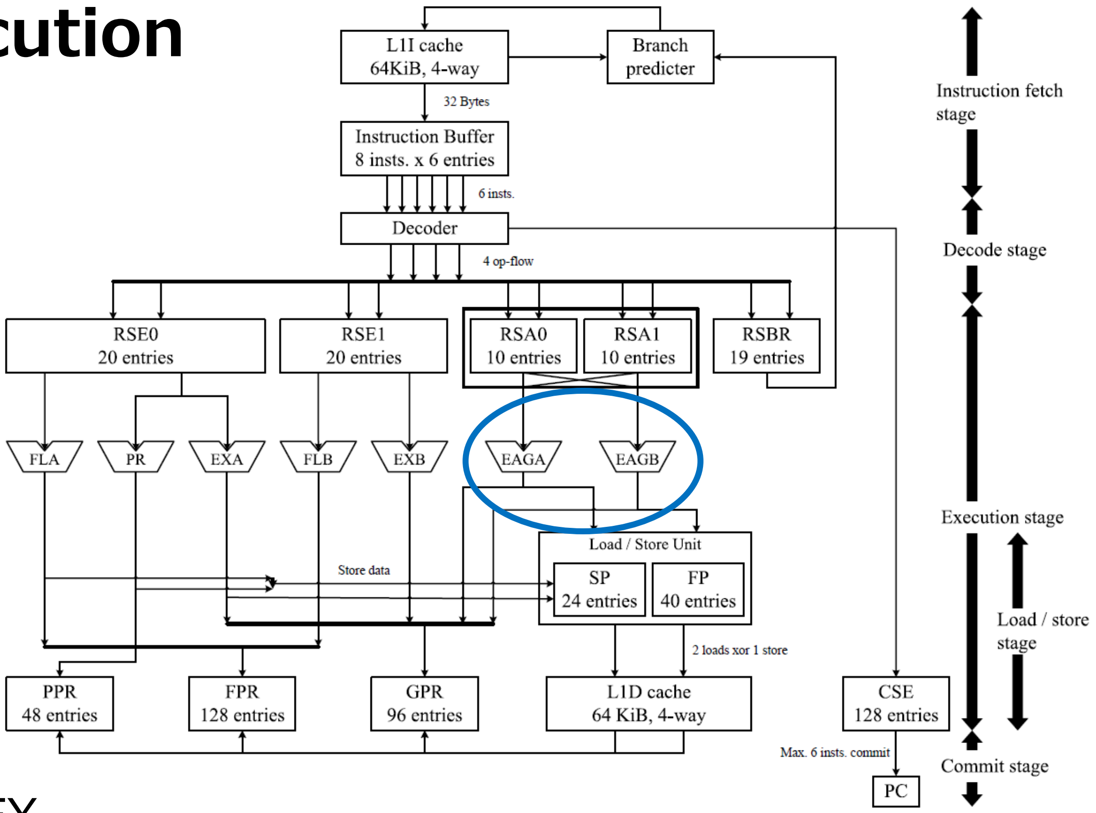
⇒ **We adopt Iterated Modulo Scheduling as a scheduling algorithm**

[*] Codina et al., A Comparative Study of modulo Scheduling Techniques.(ICS '02)

Software pipelining implementation for the A64FX

LD2W instruction execution

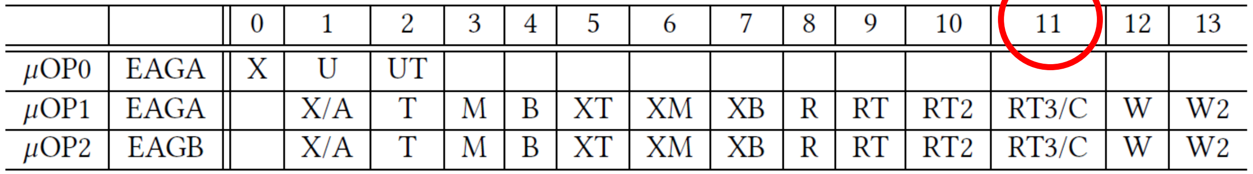
- The A64FX decomposes LD2W instruction into μOP instructions for one address calculation (μOP0) and two data loads (μOP1 and μOP2)
- It submits the operation to either the EAGA or EAGB pipeline



https://github.com/fujitsu/A64FX

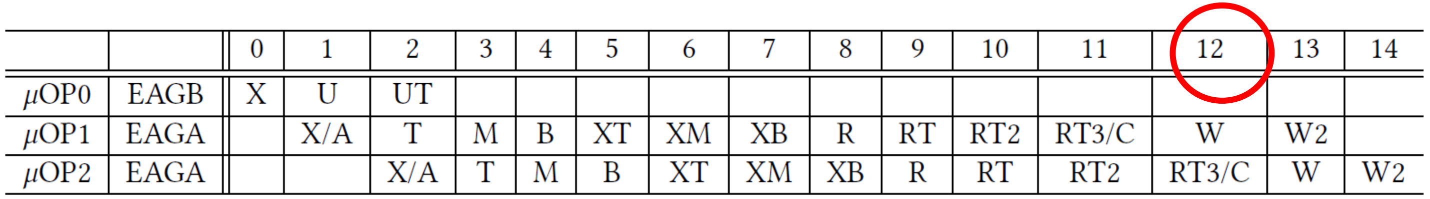
Software pipelining implementation for the A64FX

- **There are two possible instruction latency patterns for LD2W**



		0	1	2	3	4	5	6	7	8	9	10	11	12	13
μOP0	EAGA	X	U	UT											
μOP1	EAGA		X/A	T	M	B	XT	XM	XB	R	RT	RT2	RT3/C	W	W2
μOP2	EAGB		X/A	T	M	B	XT	XM	XB	R	RT	RT2	RT3/C	W	W2

Figure 2: Pipeline execution pattern 4 of LD2W (scalar plus scalar) instruction



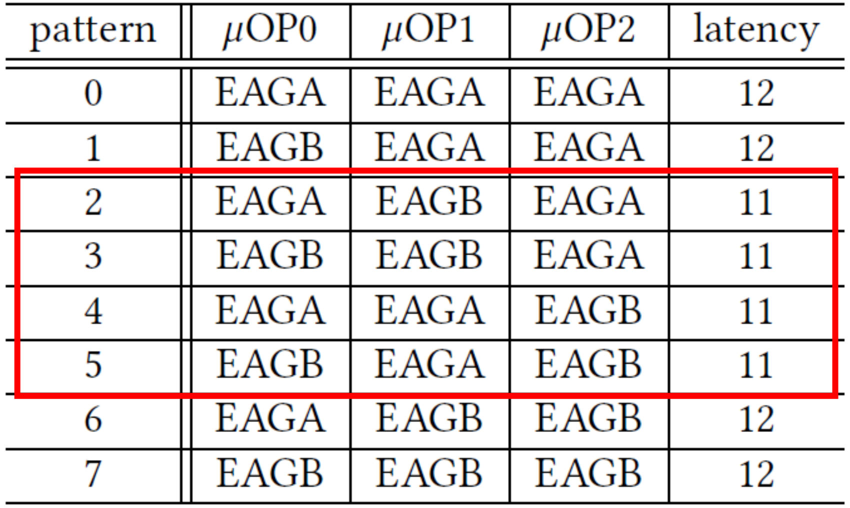
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
μOP0	EAGB	X	U	UT												
μOP1	EAGA		X/A	T	M	B	XT	XM	XB	R	RT	RT2	RT3/C	W	W2	
μOP2	EAGA			X/A	T	M	B	XT	XM	XB	R	RT	RT2	RT3/C	W	W2

Figure 3: Pipeline execution pattern 1 of LD2W (scalar plus scalar) instruction

Software pipelining implementation for the A64FX

- There are eight execution patterns for LD2W
- Since our implementation does not use pipeline execution patterns that cause delays as a scheduling model

Table 1: Pipeline execution patterns of LD2W (scalar plus scalar) instruction



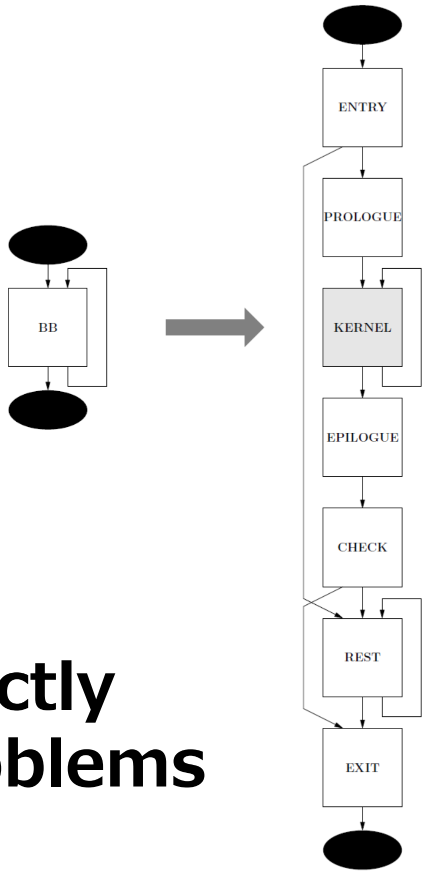
pattern	μOP0	μOP1	μOP2	latency
0	EAGA	EAGA	EAGA	12
1	EAGB	EAGA	EAGA	12
2	EAGA	EAGB	EAGA	11
3	EAGB	EAGB	EAGA	11
4	EAGA	EAGA	EAGB	11
5	EAGB	EAGA	EAGB	11
6	EAGA	EAGB	EAGB	12
7	EAGB	EAGB	EAGB	12

Software pipelining implementation for the A64FX

FUJITSU

- In our implementation, the kernel part is in non-SSA form
- Register allocation is performed in the kernel part
- Parts other than the kernel maintain the SSA form and leave processing to the subsequent LLVM pass

⇒ Our optimization pass can directly solve the COPY and spill code problems



Software pipelining implementation for the A64FX

- Our implementation suppresses the instruction level parallelism and reschedules the kernel if there is a shortage of registers due to register allocation to the kernel part
- If that doesn't work, introduce spill code and reschedule the kernel

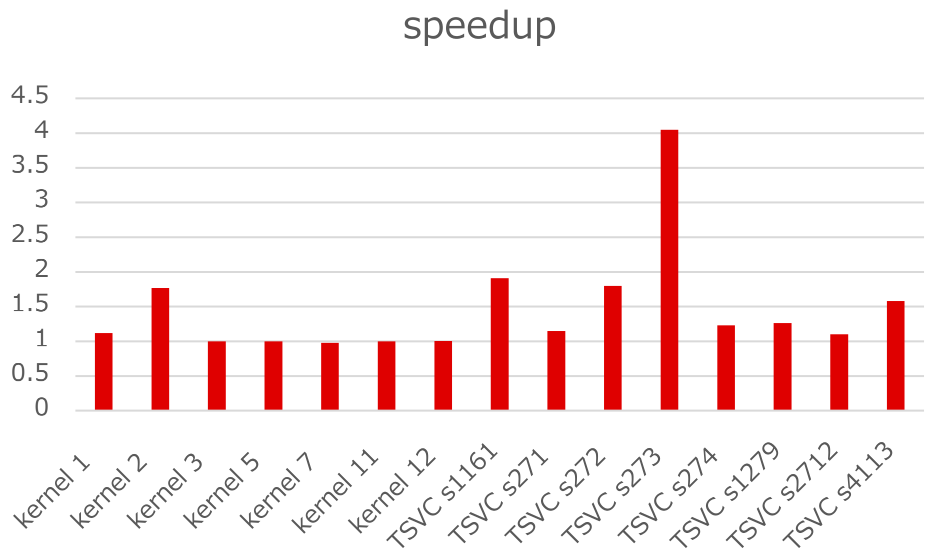
⇒ This method guarantees that unscheduled spill code will not occur

Performance Evaluation

- We used the Livermorec benchmark and the TSVC benchmark
- We rewrite loops that can be executed in parallel into SIMD executable code using the ACLE descriptions
- Loop unrolling was not applied to avoid register shortage situations
- Accurate data dependence distance information is provided at compile time via command line options if necessary

Performance Evaluation

Evaluation Environment	
System	PRIMEHPC FX700
CPU	A64FX, 2.0 GHz, 48 cores
Memory	32GiB(HBM2)
OS	CentOS 8.3
Compiler	LLVM 17.0.4
Compile option	-O2 -fno-unroll-loops -msve-vector-bits=512



- Kernel 7 has slightly lower performance
- There are kernels that have a large performance improvement rate even on a single core

Performance Evaluation

- **Unroll : the number of times MVE unrolled the loop body**

⇒ It is necessary to use MVE to generate high-performance kernels

benchmark	unroll
kernel 1	8
kernel 2	6
kernel 3	1
kernel 5	3
kernel 7	6
kernel 11	3
kernel 112	8
TSVC s1161	5
TSVC s271	7
TSVC s272	7
TSVC s273	8
TSVC s274	7
TSVC s1279	7
TSVC s2712	7
TSVC s4113	4

Performance Evaluation

- II : the value of the initiation interval resulting from software pipelining
- II∞ : the initiation interval value, assuming an infinite number of registers exist

⇒ This table indicates that the lack of registers suppresses instruction-level parallelism in most benchmarks

benchmark	II∞	II
kernel 1	4	6
kernel 2	4	7
kernel 3	109	109
kernel 5	20	20
kernel 7	6	12
kernel 11	10	10
kernel 112	2	3
TSVC s1161	6	11
TSVC s271	3	7
TSVC s272	5	7
TSVC s273	7	8
TSVC s274	6	10
TSVC s1279	4	10
TSVC s2712	3	7
TSVC s4113	23	23

Future Work

- Adjusting the loop size by loop distribution/fusion and loop unrolling is necessary for register shortage problem
- Introducing hierarchical reduction and reverse-if-conversion for loops of scalar instructions with conditional statements
- Suppressing application of software pipelining to loops with a small number of executions

Conclusions

- Software pipelining is an essential optimization for accelerating HPC applications on CPUs
- We implement software pipelining for the A64FX processor on LLVM and evaluate its performance
- We confirmed that our implementation improves the performance of several benchmark programs
- We are also considering proposing our implementation to LLVM upstream

FUJITSU

Thank you