Linaro Connect
MADRID 2024 | MAY 14-17 2024

MAD24-415
Software Pipelining Support for AArch64 in LLVM

Ryotaro Kasuga
Fujitsu Limited

Overview

- MachinePipeliner is an optimization pass in LLVM
- We implemented AArch64 support and some improvements for MachinePipeliner
  - Patches were submitted to LLVM upstream
- There are examples where MachinePipeliner improves performance on Neoverse V1
- There are future works to make MachinePipeliner better

https://llvm.org/Logo.html

Background

- Software Pipelining (SWP) is an optimization algorithm that reorders instructions in a loop
- Software Pipelining is valuable for Arm processors because
  - These days Arm processors are adopted for Datacenter and HPC
    - e.g., Arm Neoverse
  - Software Pipelining is an effective optimization for loops with chains of instructions
    - Typically, in HPC applications
    - It has a proven track record with A64FX, which is an Arm processor we developed
    - Performance improvement can be expected for processors without Simultaneous Multithreading (SMT)
      - e.g., Neoverse V1 is not equipped with SMT
- In addition, Software Pipelining is also effective for FUJITSU-MONAKA
  - FUJITSU-MONAKA is an Arm processor we are developing

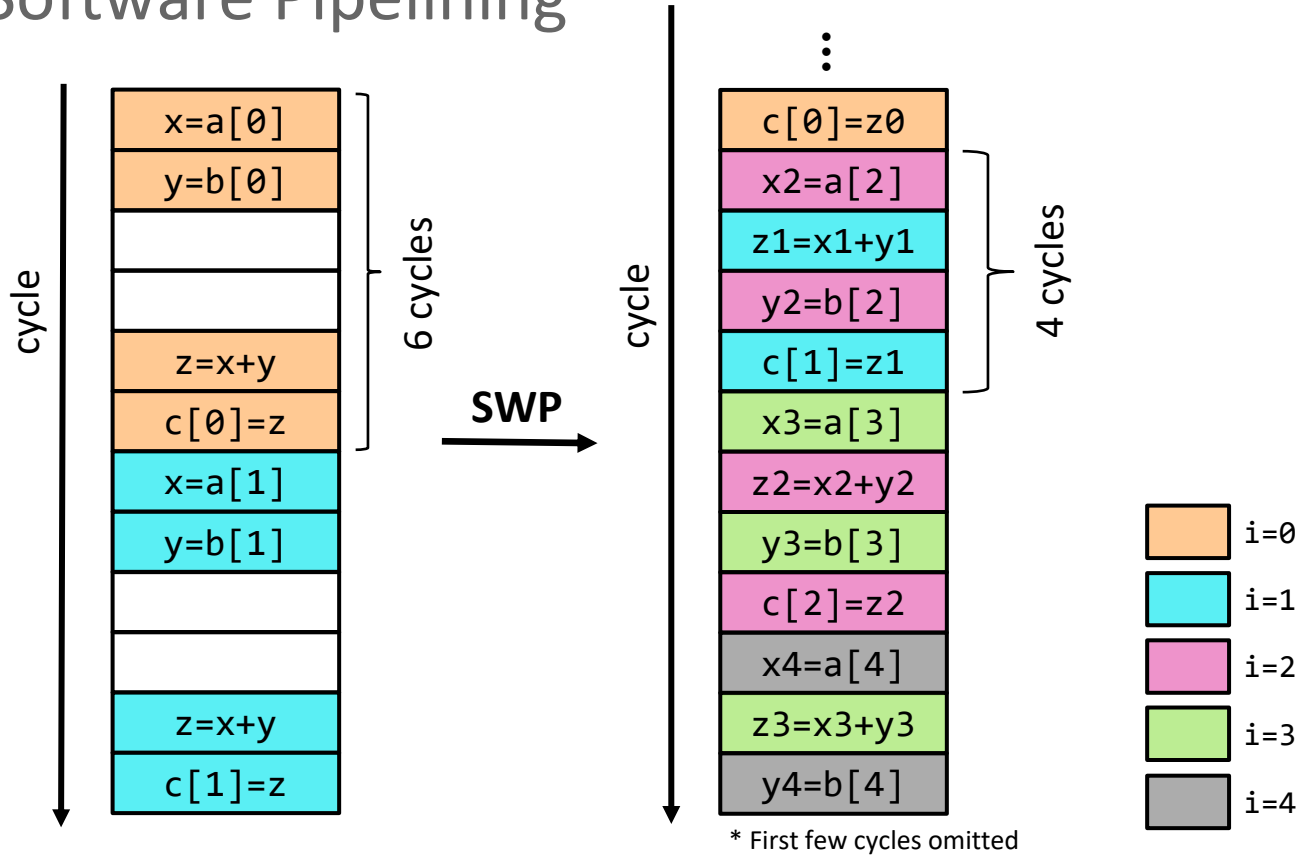
An Example of Software Pipelining

Processor

- In-Order
- One Issue
- Load/Store Latency: 3
- Add Latency: 1

Target Code

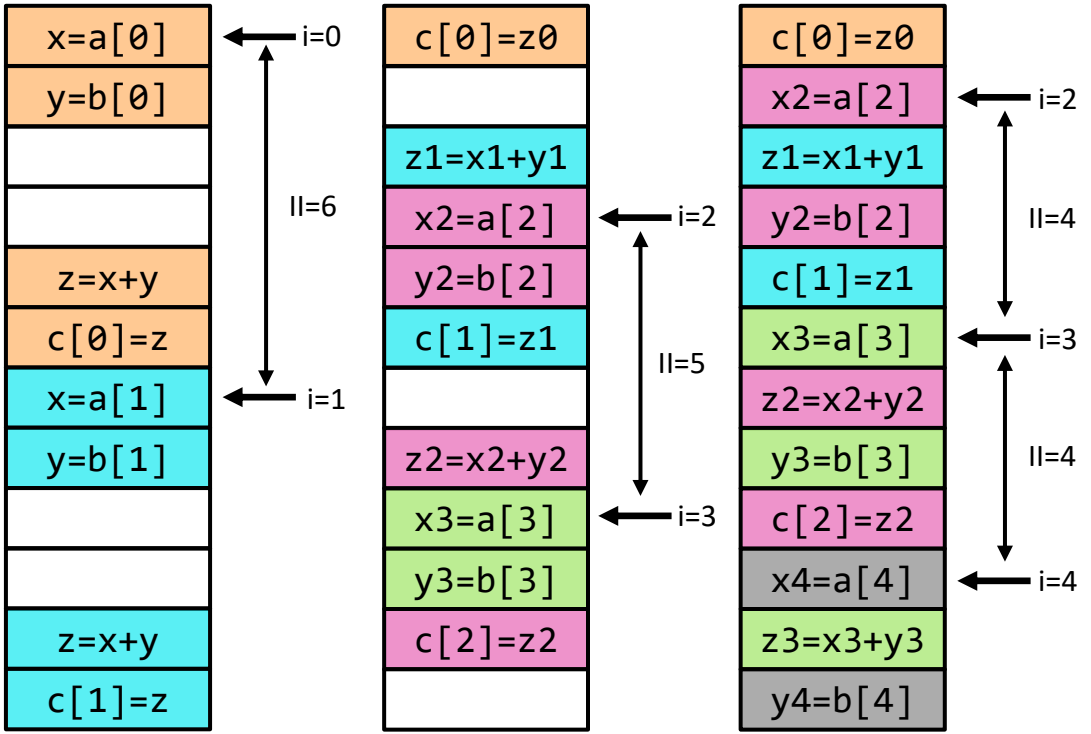
```
for (i=0; i<N; i++) {
  x = a[i];
  y = b[i];
  z = x + y;
  c[i] = z;
}
```



* First few cycles omitted

An Example of Software Pipelining

- **Initiation Interval (II)** is the number of cycles between successive loop iteration starts
- It's equals to the number of cycles to execute one iteration
- Therefore, smaller II is generally shows better performance



An Example of Software Pipelining

- How to realize the previous schedule
- Note that this is at the source code level

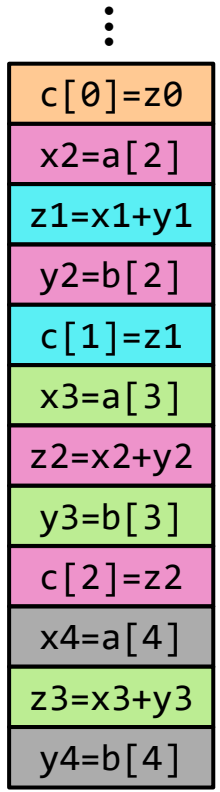
```
for (i=0; i<N; i++) {
  x = a[i];
  y = b[i];
  z = x + y;
  c[i] = z;
}
```

```
x0 = a[0]; y0 = b[0];
x1 = a[1]; y1 = b[1];
z0 = x0 + y0;
for (i=2; i<N-2; i++) {
  c[i-2] = z0;
  x0 = a[i];
  z1 = x1 + y1;
  y0 = b[i];
  z0 = z1;  // mov
  x1 = x0;  // mov
  y1 = y0;  // mov
}
c[N-2] = z0;
z1 = x1 + y1;
c[N-1] = z1;
```

Prolog

Kernel

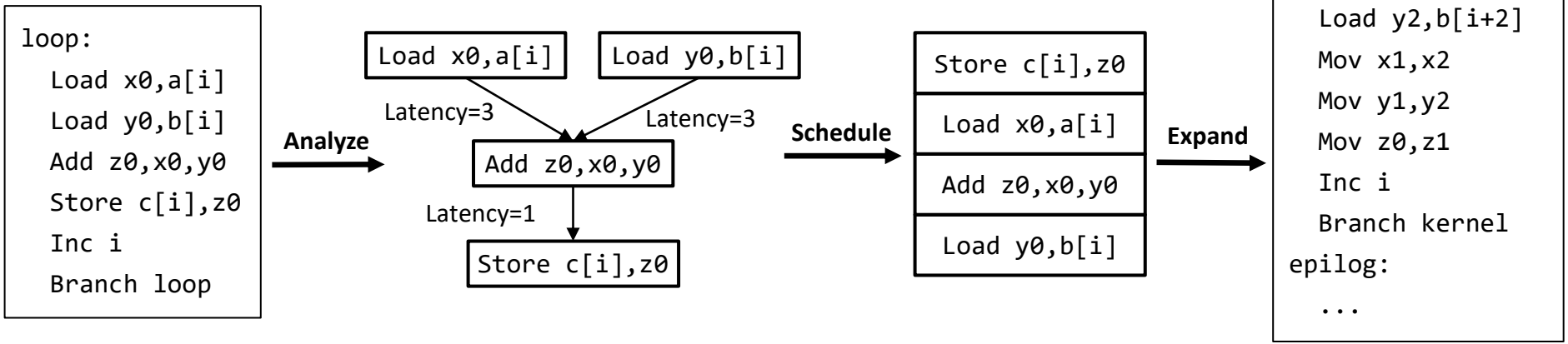
Epilog



* prolog is omitted
* movs are omitted

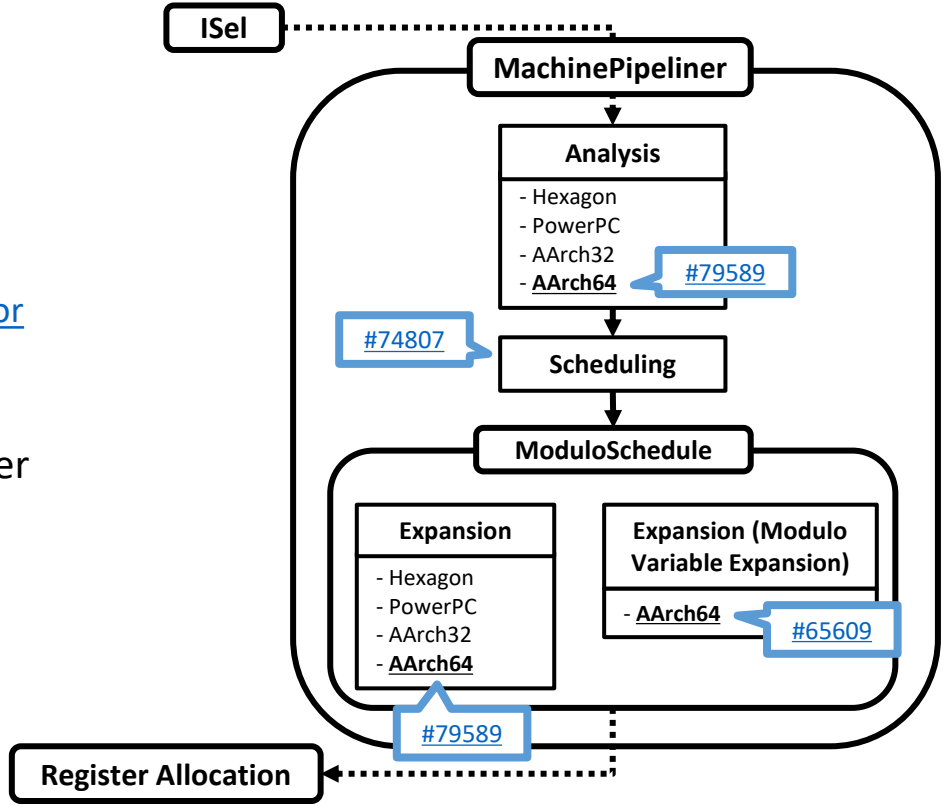
Software Pipelining in LLVM

- In LLVM, MachinePipeliner is an optimization pass for Software Pipelining
- MachinePipeliner consists of three phases
  - Analyze the target loop and build a graph called Data Dependence Graph
  - Schedule instructions based on the graph
  - Expand the scheduled instructions sequence



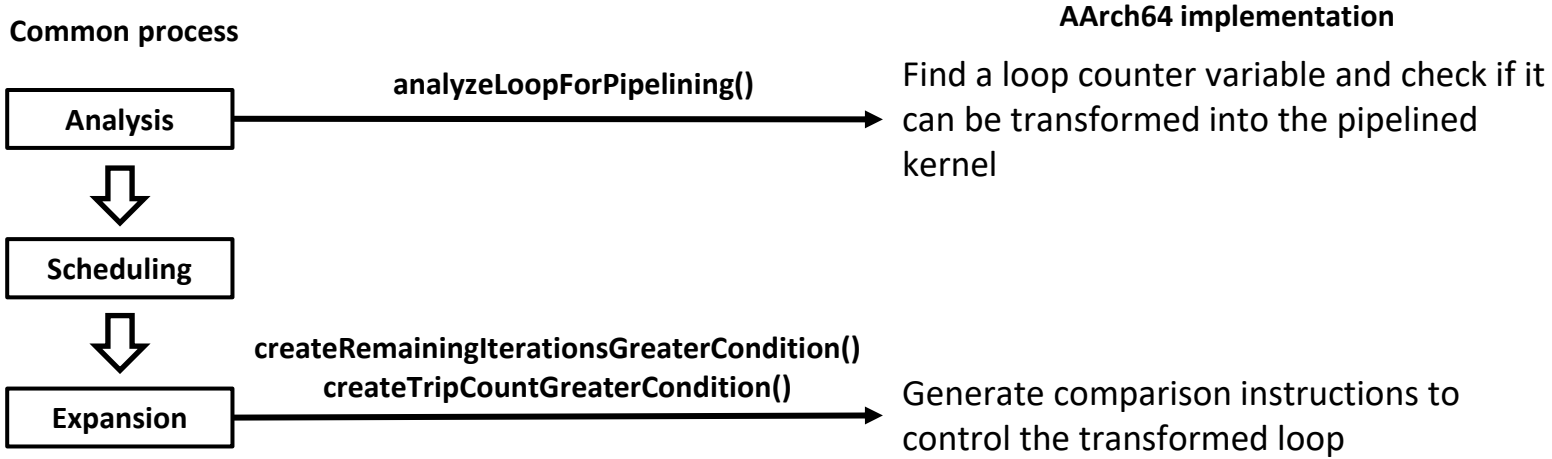
Software Pipelining in LLVM

- Our contributions
  - Add pipeliner support for AArch64 (#79589)
  - Limit register pressure when scheduling (#74807)
  - Implement modulo variable expansion for pipelining (#65609) (under review)
  - Some minor bug fixes
- The schedule model for MachineScheduler can be reused
  - MachineScheduler is local scheduler of Basic-Block units



Details of Implementation: AArch64 Support

- Add pipeliner support for AArch64 (#79589)
  - Also revised in Implement modulo variable expansion for pipelining (#65609) (under review)
- Implement some architecture-dependent functions



Details of Implementation: Scheduling Improvement

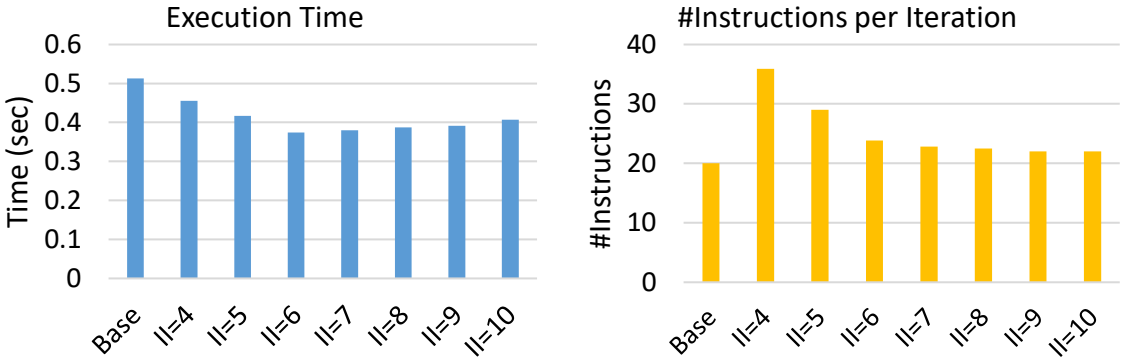
- Limit register pressure when scheduling (#74807)
- The minimum schedulable Initiation Interval (**II**) is not necessarily the best
  - Too small II can generate additional register spills/fills
- This patch introduces a mechanism to reject schedules with high register pressure

Search for Schedule (pseudo code)

```
for (II=MinII; II<=MaxII; II++) {
  Schedule,Found = TryToSchedule(II);
  if (Found && EnoughRegs(Schedule))
    return;
}
// Could not find schedule
```

Added in this patch

SingleSource/Benchmarks/Misc/flops-6.c **(llvm-test-suite)**



Compile options (base)

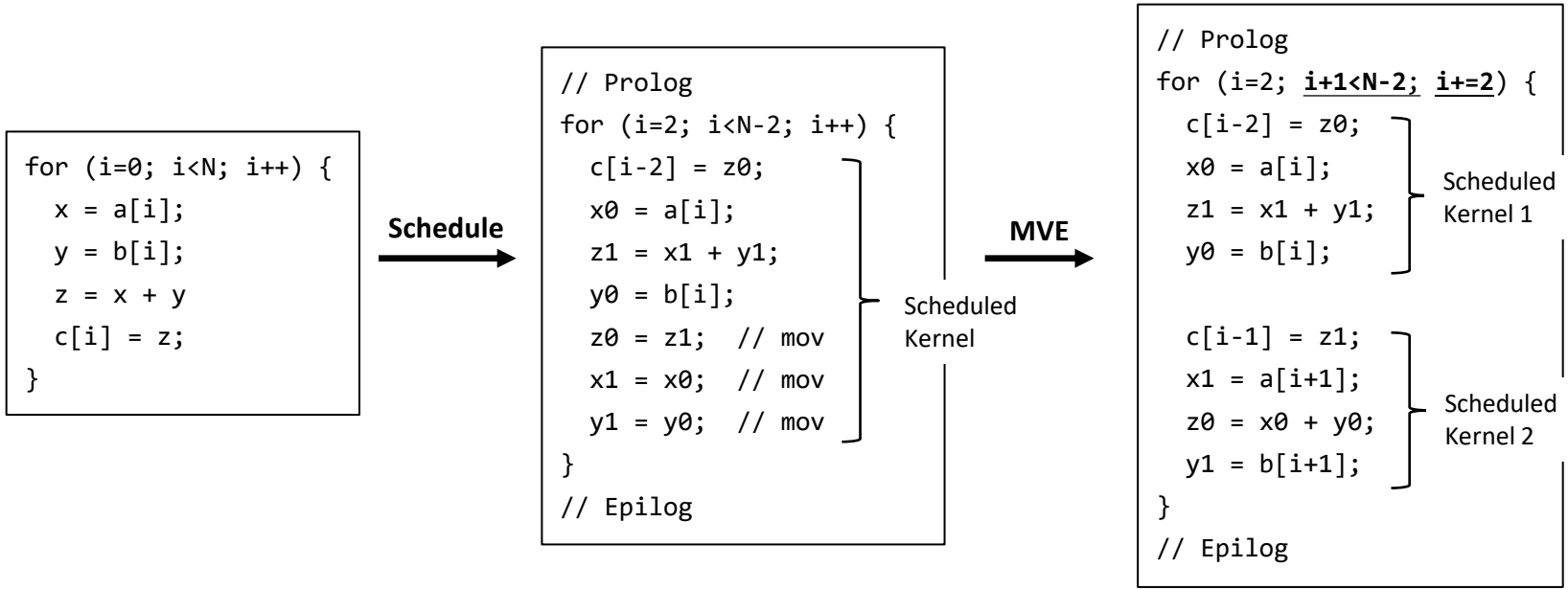
-Ofast -mrecip -mcpu=neoverse-v1
-fno-vectorize -fno-unroll-loops

Compile options (additional for SWP)

-mllvm -aarch64-enable-pipeliner
-mllvm -pipeliner-max-stages=20
-mllvm -pipeliner-mve-cg
-mllvm -pipeliner-enable-copytophi=0
-mllvm -pipeliner-force-ii=$ii

Details of Implementation: Modulo Variable Expansion

- Implement modulo variable expansion for pipelining (#65609) (under review)
- If we don't have register window, we must insert movs to hold values across iterations
- Modulo Variable Expansion (MVE) eliminates them by unrolling the loop



Performance Improvements

- We observed performance improvement with SWP in some cases
- All cases are measured under following environment
  - CPU: Graviton3
  - RAM: 64 GB (c7g.8xlarge)
  - Use 1 core, 1 thread
  - LLVM Version: bfd19445c38a2ad6a1def7ee9a1f8ff26a159caf, applying #65609

Case 1. Long Dependence Chain

- Comparison of OoO and OoO + SWP
- A loop with long fadd chain
  - On Neoverse V1 (refer to optimization guide)
    - Latency: 2
    - Throughput: 2 (maximum)
- Throughput goes down without SWP
- It is stable when applying SWP

Compile options (base)

```
-O3 -mcpu=neoverse-v1
-fno-unroll-loops
```

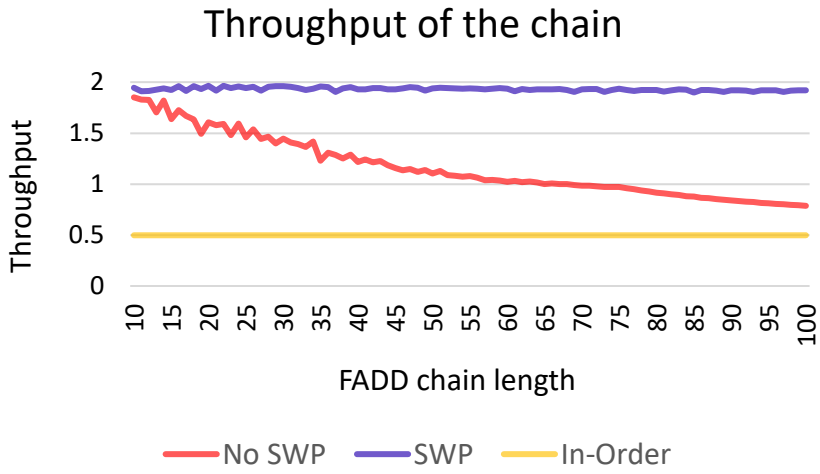
Compile options (SWP)

```
-mllvm -aarch64-enable-pipeliner
-mllvm -pipeliner-max-stages=20
-mllvm -pipeliner-max-mii=100
-mllvm -pipeliner-enable-copytophi=0
-mllvm -pipeliner-mve-cg
-mllvm -enable-misched=0
-mllvm -enable-post-misched=0
```

Target Code

```
for (int i=0; i<n; i++) {
  double z = b[i];
  z = z + b[i];
  ...
  z = z + b[i];
  a[i] = z
}
```

Change the length of this chain



Case 2. Benchmark Application

- ExaMiniMD
  - One of the ECP Proxy Applications
  - Molecular Dynamics
- A hotspot loop calculates Lennard-Jones potential
  - Accounts for about 50% of total time

Target Loop in src/force_types/force_lj_neigh_impl.h

```
 for(int jj = 0; jj < num_neighs; jj++) {
    T_INT j = neighs_i(jj);
    const T_F_FLOAT dx = x_i - x(j,0);
    const T_F_FLOAT dy = y_i - x(j,1);
    const T_F_FLOAT dz = z_i - x(j,2);

    const int type_j = type(j);
    const T_F_FLOAT rsq = dx*dx + dy*dy + dz*dz;

    const T_F_FLOAT cutsq_ij =
STACKPARAMS?stack_cutsq[type_i][type_j]:rnd_cutsq(type_i,type_j);

    if( rsq < cutsq_ij ) {
      const T_F_FLOAT lj1_ij =
STACKPARAMS?stack_lj1[type_i][type_j]:rnd_lj1(type_i,type_j);
      const T_F_FLOAT lj2_ij =
STACKPARAMS?stack_lj2[type_i][type_j]:rnd_lj2(type_i,type_j);

      T_F_FLOAT r2inv = 1.0/rsq;
      T_F_FLOAT r6inv = r2inv*r2inv*r2inv;
      T_F_FLOAT fpair = (r6inv * (lj1_ij*r6inv - lj2_ij)) * r2inv;
      fxi += dx*fpair;
      fyi += dy*fpair;
      fzi += dz*fpair;
    }
  }
```

Thompson, Aidan, Cangi, Attila, Trott, Christian, Junghans, Christoph, Moore, Stan, Germann, Tim, and USDOE. ECP-copa/ExaMiniMD. Computer software. https://www.osti.gov//servlets/purl/1696908. USDOE. 13 Feb. 2018. Web. doi:10.11578/dc.20201102.5.

Case 2. Benchmark Application

- SWP reduces the execution time of the target loop by 6.7%
  - 3.6% overall

Compile options (base)

```
-Ofast -mcpu=neoverse-v1 -mrecip
-mllvm -sve-gather-overhead=1 -mllvm -sve-scatter-overhead=1
```

Compile options (additional for force_types/force_lj_neigh.cpp)

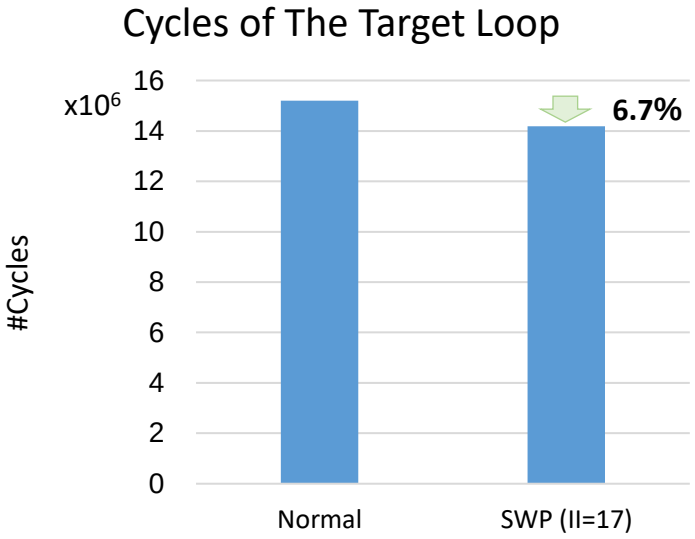
```
-mllvm -aarch64-enable-pipeliner -mllvm -pipeliner-max-stages=20
-mllvm -pipeliner-enable-copytophi=0 -mllvm -pipeliner-mve-cg
-mllvm -pipeliner-force-ii=17 -fno-unroll-loops
```

Other Build Arguments

```
- MPI=0
- KOKKOS_DEVICES=Serial
```

Run

```
$ ./ExaMiniMD -il ../input/in.lj --kokkos-num-threads=1
```



Future Works

- There are still improvements that can be made
  - Provide better user interface like pragma
    - There is a pragma to disable SWP, but no pragma to enable
  - Improve data dependence analysis of target loops, especially for memory dependencies
    - Some are lacking, some are excessive
    - We are discussing about this topic in community
  - Eliminate factors inhibiting MachinePipeliner due to other optimizations
    - e.g., Around VL register of SVE
  - Enhance algorithms to determine Initiation Interval
- We'll keep contributing to LLVM to enhance MachinePipeliner
  - Aiming to target LLVM 20
- We are happy if you are interested in developing MachinePipeliner with us

Acknowledge

- This presentation is based on results obtained from a project, JPNP21029, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).


LinaroConnect
MADRID 2024 | MAY 12-17 2024
Thank you