OpenMP Target Device Offloading for SX-Aurora TSUBASA

Tim Cramer

IT Center

RWTH AACHEN UNIVERSITY

Motivation

- Motivation
  - User codes of the RWTH Compute Cluster
    - are often memory-bound → might benefit from SX-Aurora TSUBASA capabilities
    - require standard-compliance, e.g., MPI, OpenMP
  - Performance portability: Single application for multiple types off devices
  - RWTH Aachen is member of the OpenMP ARB and Language Committee
  - Real-world applications: Not all code parts might deliver a good performance on a SX-Aurora (e.g., file IO, data initialization)

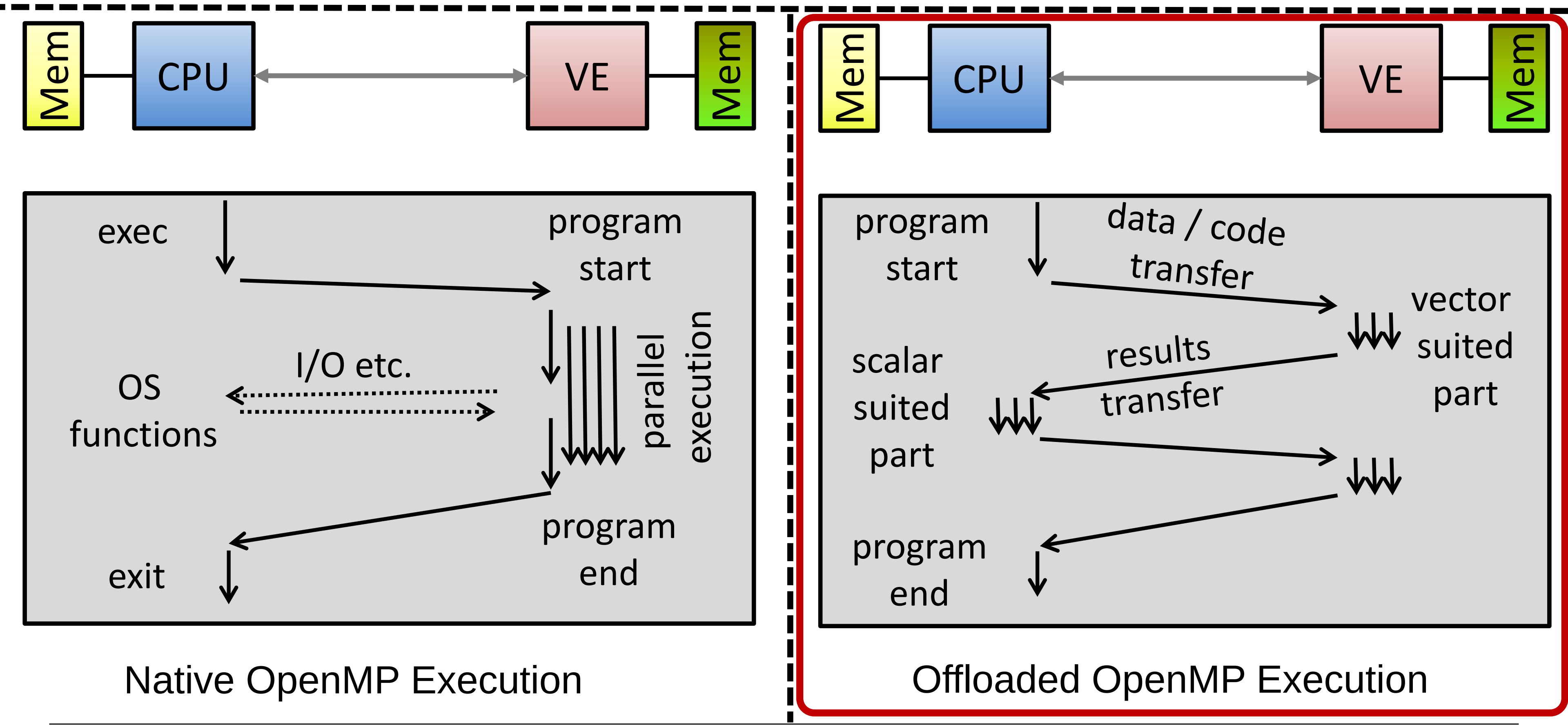
- Project Goal
  - OpenMP-based Offload Programming for the NEC SX-Aurora Architecture

Agenda

- Aurora Execution Models
- LLVM Infrastructure
- Solution 1: Source-2-Source Transformation
- Solution 2: Native LLVM-VE path
- Validation: Correctness + Performance
- Conclusion

Aurora Execution Models

- Offloading paradigm has become popular
- Supporting both approaches increases usability

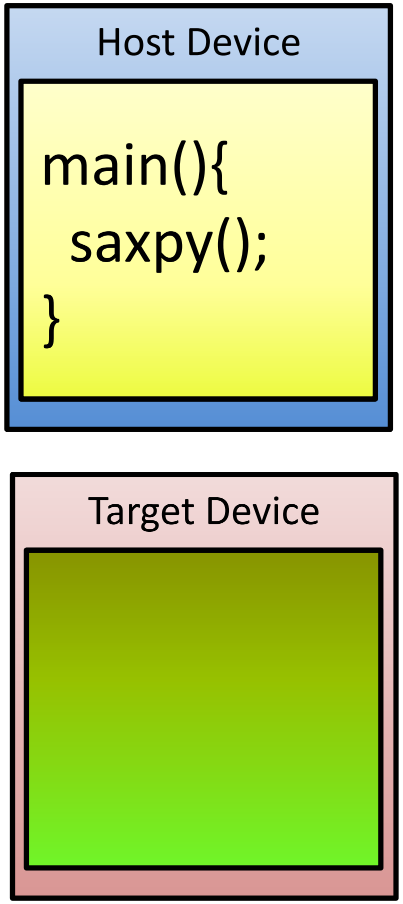


OpenMP Offloading

Target Device Offloading

```
void saxpy(){
  int n = 10240; float a = 42.0f; float b = 23.0f;
  float *x, *y;
  // Allocate and initialize x, y
  // Run SAXPY


  #pragma omp parallel for
  for (int i = 0; i < n; ++i){
    y[i] = a*x[i] + y[i];
  }
}
```

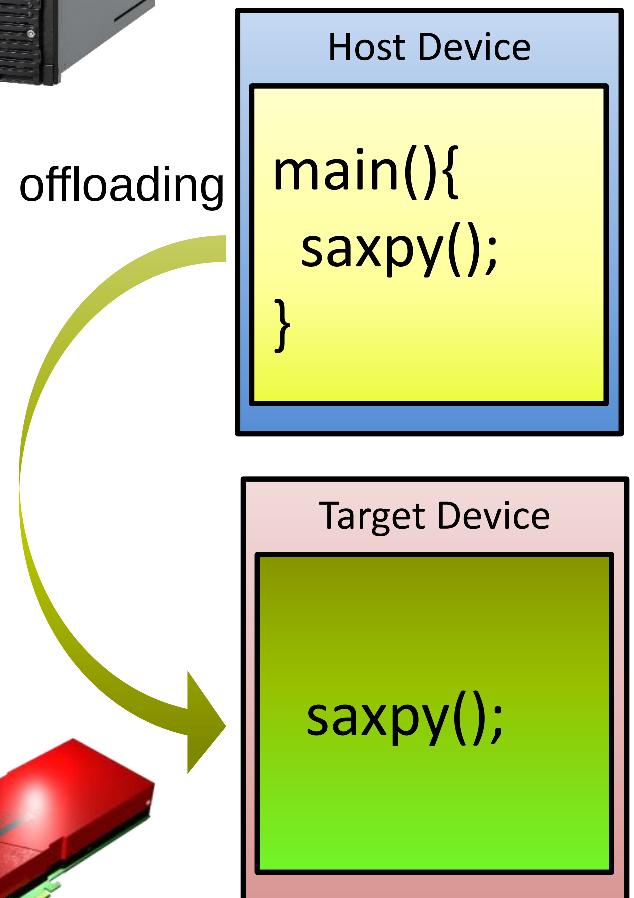


OpenMP Offloading

Target Device Offloading

```
void saxpy(){
  int n = 10240; float a = 42.0f; float b = 23.0f;
  float *x, *y;
  // Allocate and initialize x, y
  // Run SAXPY

  #pragma omp target
  #pragma omp parallel for
  for (int i = 0; i < n; ++i){
    y[i] = a*x[i] + y[i];
  }
}
```

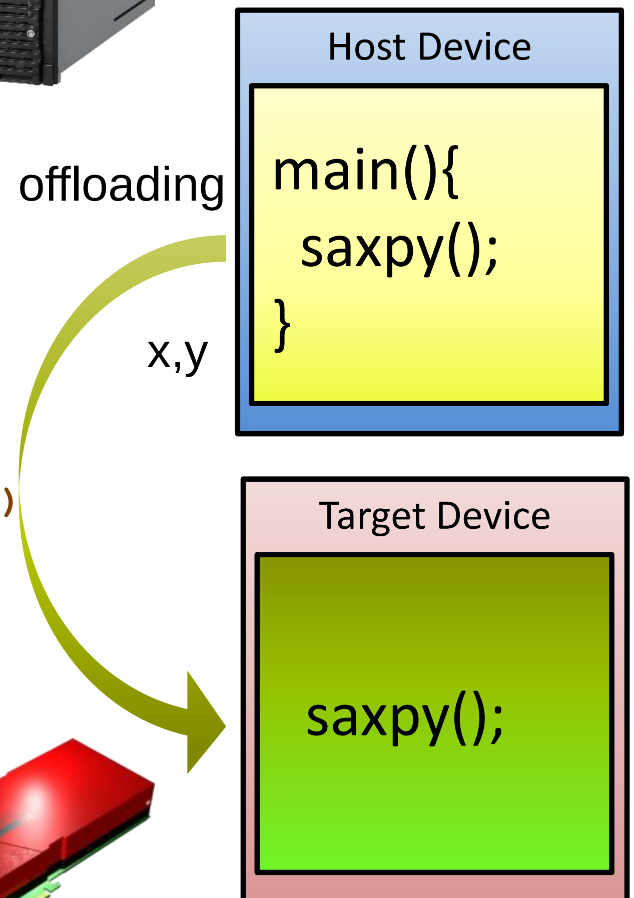


OpenMP Offloading

Target Device Offloading

```
void saxpy(){
  int n = 10240; float a = 42.0f; float b = 23.0f;
  float *x, *y;
  // Allocate and initialize x, y
  // Run SAXPY

  #pragma omp target map(to:x[0:n]) map(tofrom:y[0:n])
  #pragma omp parallel for
  for (int i = 0; i < n; ++i){
    y[i] = a*x[i] + y[i];
  }
}
```



Implementation of the VEO Infrastructure

- Goal: Simple usage of OpenMP Offloading by applying a new target-triple
  - `$ clang -fopenmp -fopenmp-targets=`**`aurora-nec-veort-unknown`**` input.c`
  - Integration in LLVM infrastructure

- Architecture (required components)
  - libomptarget and target OpenMP runtime
  - Clang driver integration
  - Source transformation with `sotoc`
  - Build wrapper

LLVM Offloading Infrastructure

- Central component for LLVM offloading: libomptarget library
  - The offload infrastructure supports multiple target device types at runtime
  - The infrastructure determines the availability of target devices at runtime
  - Target code is stored inside the host binaries as additional ELF sections (Fat Binary)
  - Target code is either target assembly in binary form (ELF, PE, etc.) or a higher-level intermediate representation (IR) such as LLVM IR or any other type of IR

- Development of a SX-Aurora TSUBASA plugin
  - Vector code integrated into the fat binary
  - Plugin use VE Offloading (VEO) framework [1]

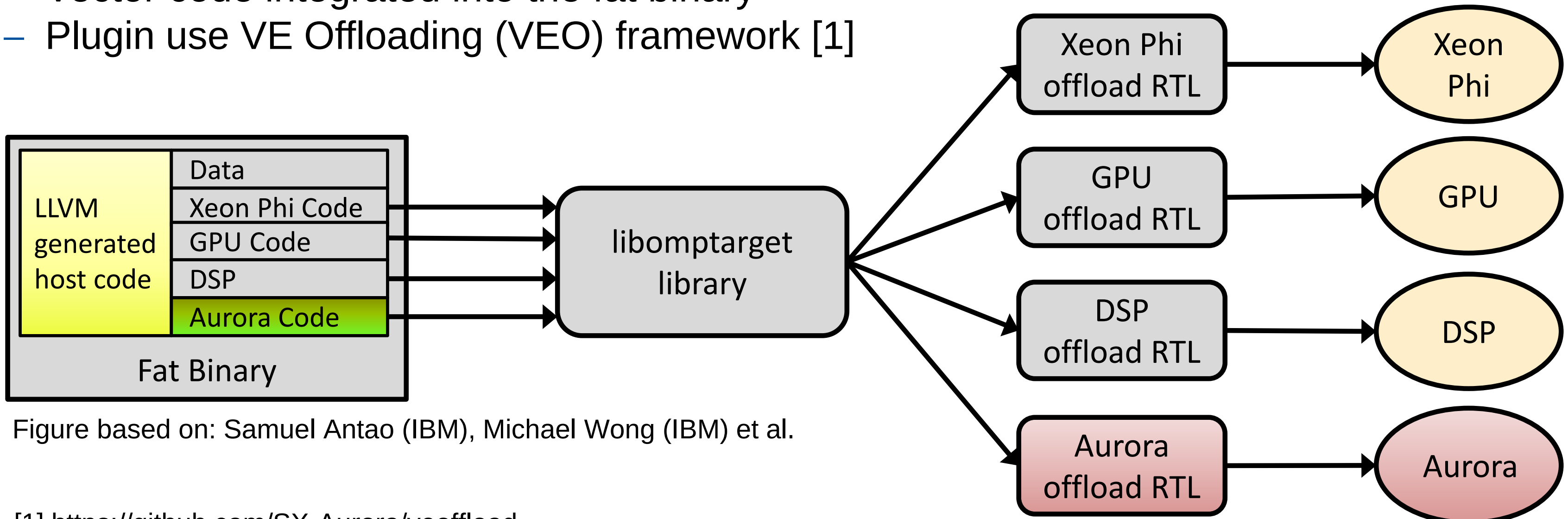
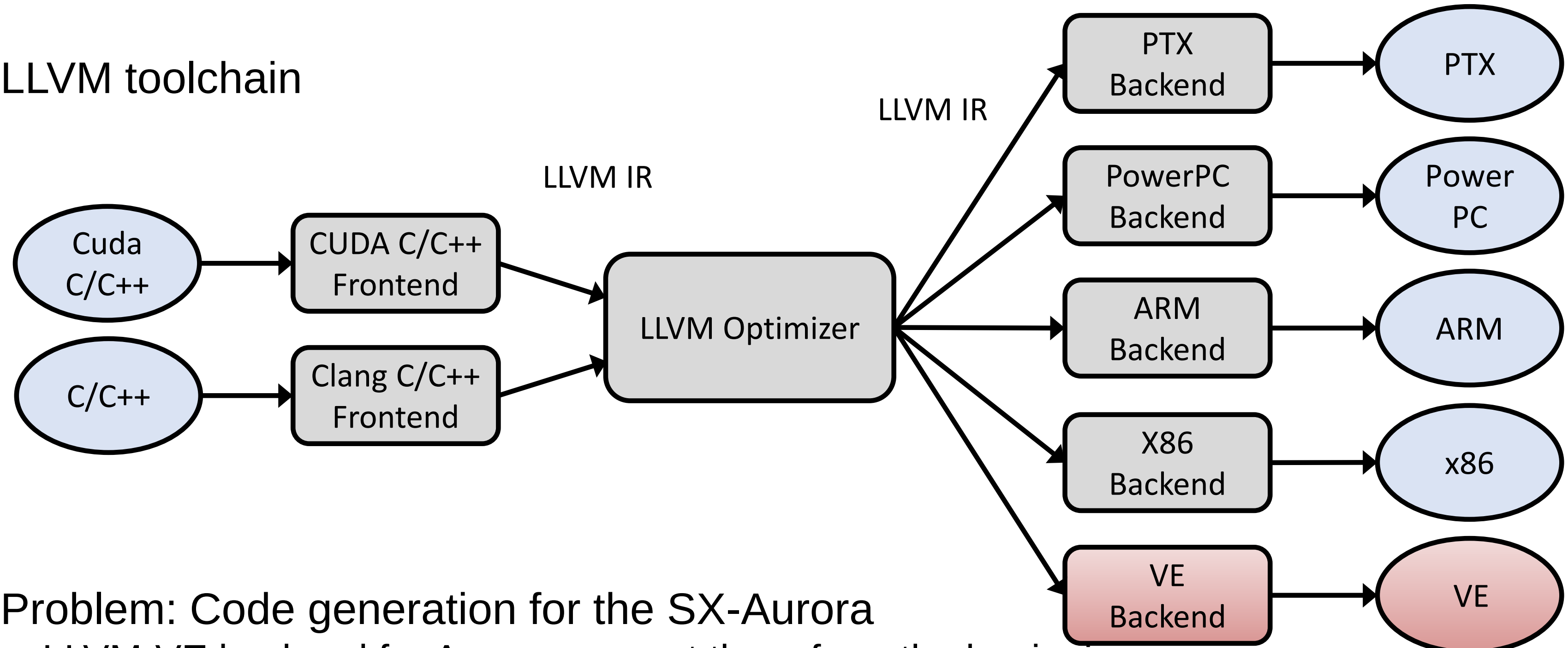


Figure based on: Samuel Antao (IBM), Michael Wong (IBM) et al.

[1] https://github.com/SX-Aurora/veoffload

Source-To-Source Transformation with SOTOC

- LLVM toolchain



- Problem: Code generation for the SX-Aurora
  - LLVM-VE backend for Aurora was not there from the beginning
  - NEC compiler does not understand LLVM IR
- Solution: Source-to-source transformation tool
  - Powerful interface with full control of the AST
  - Outlining of target regions (including parameters/dependencies)
  - NEC Compiler generates target device code
  - Integrated into the driver

Source-To-Source Transformation with SOTOC (example)

```
void saxpy(){
  int n = 10240; float a = 42.0f; float b = 23.0f;
  float *x, *y;
  // Allocate and initialize x, y
  #pragma omp target map(to:x[0:n]) map(tofrom:y[0:n])
  #pragma omp parallel for
  for (int i = 0; i < n; ++i){
    y[i] = a*x[i] + y[i];
  }
}
```

```
$ sotoc saxpy.c -- -fopenmp
```

```
void __omp_offloading_28_395672b_saxpy_l8(int *__sotoc_var_n, float * y,
                                          float *__sotoc_var_a, float * x) {
  int n = *__sotoc_var_n;
  float a = *__sotoc_var_a;
  #pragma omp parallel for
  for (int i = 0; i < n; ++i){
    y[i] = a*x[i] + y[i];
  }
  *__sotoc_var_n = n;
  *__sotoc_var_a = a;
}
```

Combined Constructs

- There is more then "`#pragma omp target`"
- For convenience OpenMP defines a big set combined constructs, e.g.:
  - `#pragma omp target parallel`
  - `#pragma omp target parallel for`
  - `#pragma omp target parallel for simd`
  - `#pragma omp target parallel loop`
  - `#pragma omp target simd`
  - `#pragma omp target teams`
  - `#pragma omp target teams distribute`
  - `#pragma omp target teams distribute simd`
  - `#pragma omp target teams loop`
  - `#pragma omp target teams distribute parallel for`
  - `#pragma omp target teams distribute parallel for simd` (really! ☺)

- Directives can have different clauses (e.g., `private, first-private, map, reduction`, etc.)
  - Some directives are only applicable to one, others to more constructs
  - Handling slightly differs in OpenMP 4.5 and 5.0
  - We implemented all of them, but some might have some limitation

Build Wrapper

- Clang driver calls wrapper infrastructure instead calling the tool (compiler, linker, assembler) directly

- Benefits
  - Independent from underlying device
  - Testing without NEC compiler possible
  - For testing: Integration of GCC code into the fat binary build by Clang

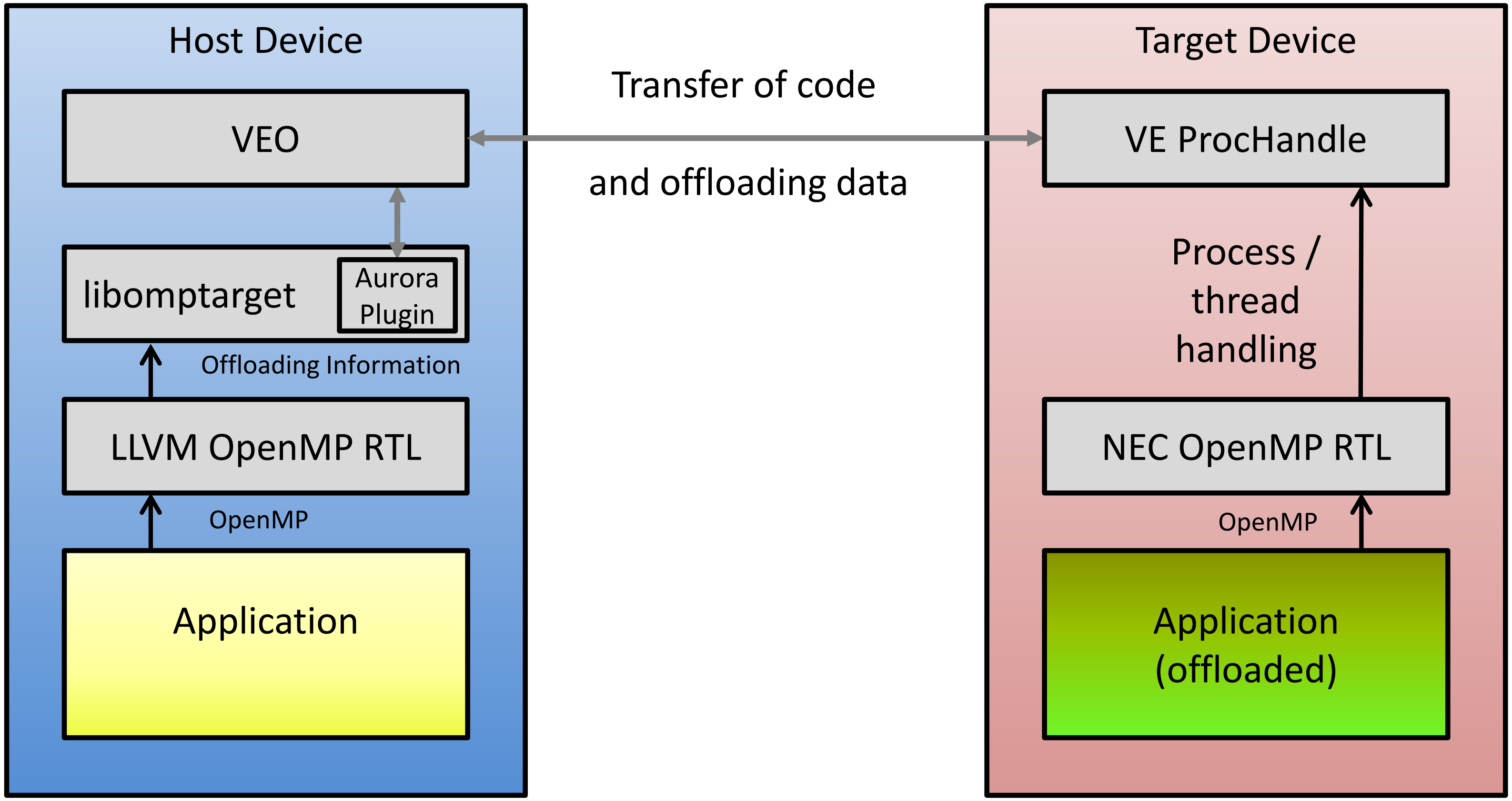
- Source-To-Source transformation not common compile step
  - SOTOC is called by the compiler wrapper

- Flexible configuration possible, e. g.
  - static linking target image:
    `-Xopenmp-target "-Xlinker -fopenmp-static"`
  - Use a different compiler for target device code (could also be a gcc for other devices):
    `-fopenmp-nec-compiler=path:/opt/nec/ve/ncc/3.0.8/bin/ncc`

→ Very generic approach

Execution Model / Target OpenMP Runtime

- Two different OpenMP runtimes
  - Host: LLVM
  - Device: NEC



Limitations source-to-source approach

- C++ support
  - Needs to differentiate in Clang driver
  - Needs some work on the build wrapper tools
- Fortran support
  - Not planned (might work with LLVM Flang in future)
- Bugs / Known issues
  - Anonymous enums and structs not supported → Hard to fix with source-2-source transformation
  - Limited support for multiple parallel target regions
- Maintenance
  - We relying on internal AST (non-stable interface) → might break with LLVM internal updates

Native LLVM-VE path

- Now a native LLVM-VE path exists in LLVM
- Using the same runtime plugin (libomptarget / VEO)
- Uses native LLVM-VE backend for VE code generation
  → Talk from Simon Moll (NEC)
- As easy as before:
  - ~~`$ clang -fopenmp -fopenmp-targets=`**`aurora-nec-veort-unknown`**` input.c`~~
  - `$ clang -fopenmp -fopenmp-targets=`**`ve-linux`**` input.c`

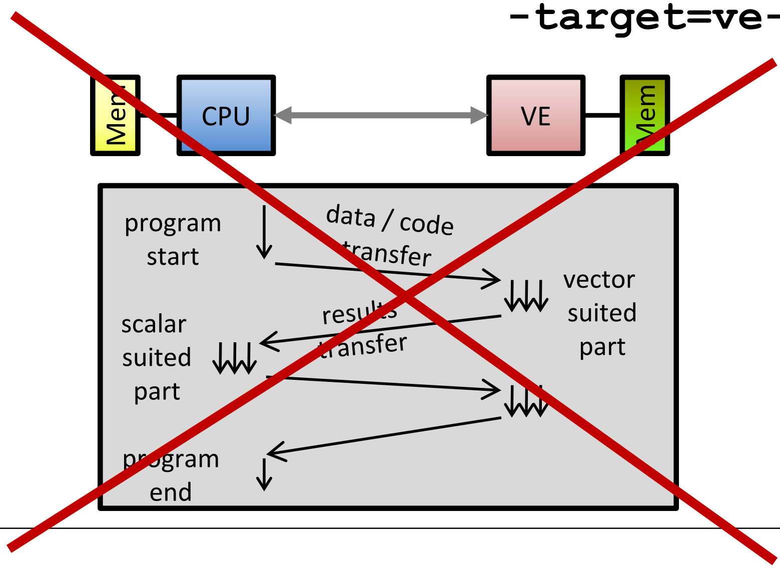
Tim Cramer

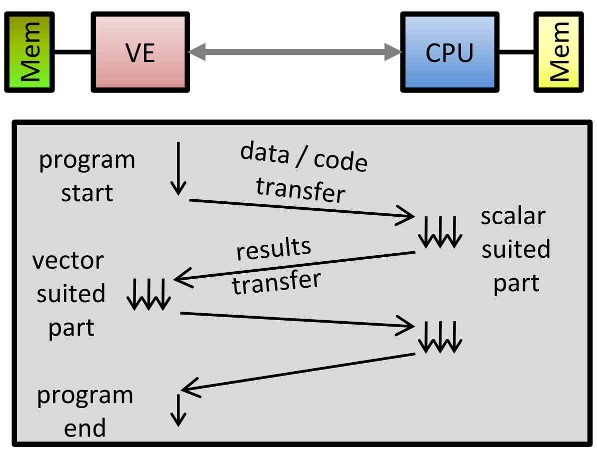
Reverse Offloading

- Using the different runtime plugin (libomptarget / **VHCall**)

- Uses native LLVM-VE backend for VE code generation
  → Talk from Simon Moll (NEC)

- As easy as before:
  - ~~`$ clang -fopenmp -fopenmp-targets=`**`aurora-nec-veort-unknown`**` input.c`~~
  - ~~`$ clang -fopenmp -fopenmp-targets=`**`ve-linux`**` input.c`~~
  - `$ clang -fopenmp -fopenmp-targets=`**`x86_64-pc-linux-gnu`**` \`
    `-target=ve-linux`` input.c`





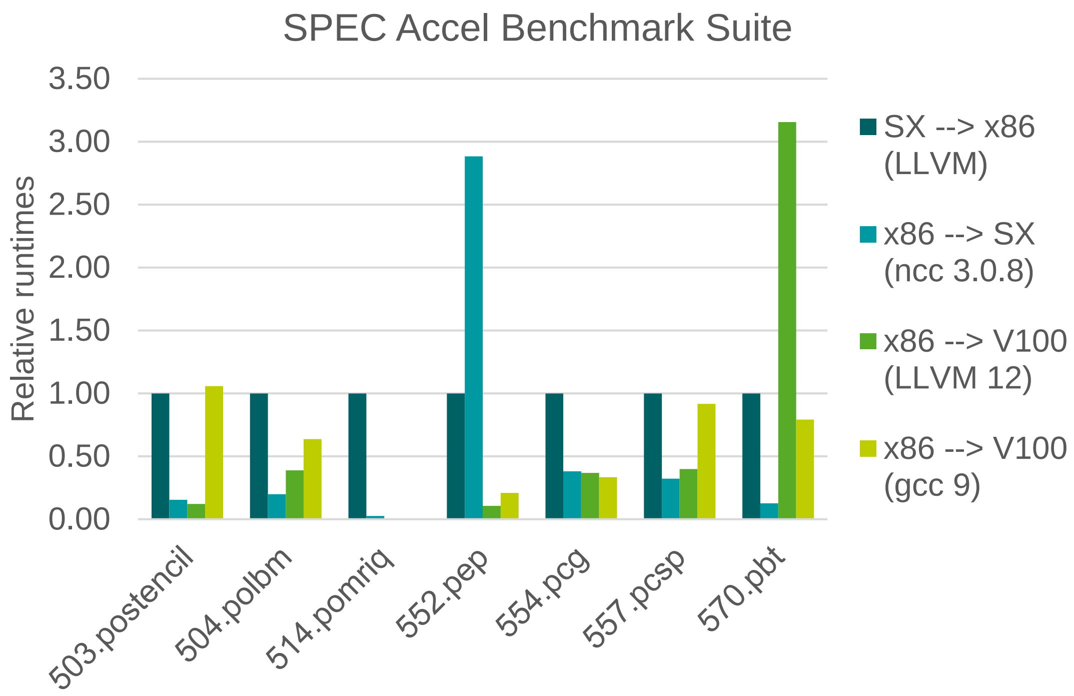
Verification with SOLLVE

- OpenMP Validation and Verification Suite (SOLLVE) [1,2]
- Designed to test OpenMP offloading implementations
- 109 tests written in C
  - 85% - 93% test compile + run successfully for all approaches
  - Most others are known limitations (or under investigation)
- 14 in C++
  - Only native LLVM-VE path can compile + run C++ test

[1] Diaz, J.M., Pophale, S., Friedline, K., Hernandez, O., Bernholdt, D.E., Chandrasekaran, S.: Evaluating Support for OpenMP Offload Features. In: Proceedings of the 47th International Conference on Parallel Processing Companion. pp. 31:1–31:10. ICPP ’18, ACM, New York, NY, USA (2018)
[2] Diaz, J.M., Pophale, S., Hernandez, O., Bernholdt, D.E., Chandrasekaran, S.: OpenMP 4.5 Validation and Verification Suite for Device Offload. In: Evolving OpenMP for Evolving Architectures. pp. 82–95. Springer Int. Publishing (2018)

Performance

- SPEC Accel Benchmark Suite
  - All benchmarks run on VE with source-to-source approach
  - Most benchmarks are competitive compared to NVidia V100 or 2x Intel Xeon Silver CPU
  - Relative results (lower is better)



- x86: 2x Xeon Silver 4108 CPUs
- SX: SX-Aurora TSUBASA Vector Engine Type 10B
- V100: Nvidia V100-SXM2 GPU

Conclusion

- This project benefits from LLVM infrastructure
- Easy to use
- Good performance
- Very generic source-to-source approach -> suitable for other target devices
- High flexible (source-to-source, native LLVM-VE, reverse offloading)

Links

- Sources
  - “sotoc path” only: https://github.com/RWTH-HPC/llvm-project/tree/aurora-offloading-prototype
  - All paths: https://github.com/sx-aurora-dev/llvm-project/tree/hpce/develop
- Packages
  - https://sx-aurora.com/repos/veos/ef_extra/x86_64
- Documentation
  - “sotoc” path: https://rwth-hpc.github.io/sx-aurora-offloading/

Thank you for your attention.