NEC and RWTH Aachen University Collaboration: OpenMP Offload Programming Model for SX-Aurora TSUBASA

Tim Cramer
SX-Aurora TSUBASA Execution Models

**Aurora Native (OpenMP) Execution**
- Execute entire (OpenMP) program on Vector Engine
- Good for highly vectorizable applications

**Aurora Offload (OpenMP) Execution**
- Execute scalar suited part of the program on the host processor
- Offload highly parallel parts on the Vector Engine
- RWTH Aachen University is working on a prototype in collaboration with NEC

→ Supporting both approaches increases usability
OpenMP Offloading

Target Device Offloading

```c
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];
    }
}

main() {
    saxpy();
}
```
OpenMP Offloading

Target Device Offloading

```c
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];
    }
}
```
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];
    }
}
Integration into the LLVM OpenMP Runtime

- Intel gave their runtime to the LLVM project (open source)
  - Supported by different compilers (Clang, Intel, GNU)
  - Target offloading support

- Goal: Simple usage of OpenMP Offloading by applying a new target-triple
  - `$ clang -fopenmp -fopenmp-targets=aurora-nec-veort-unknown input.c`
  - Clang driver calls NEC compiler for vector code generation

- Development of a SX-Aurora TSUBASA plugin for libomptarget
  - The offload infrastructure supports multiple target device types at runtime
  - Target code is stored inside the host binaries as additional ELF sections (Fat Binary)

Based on: Samuel Antao (IBM), Michael Wong (IBM) et al.
Execution Model / Target OpenMP Runtime

- Two different OpenMP runtimes
  - Host: LLVM
  - Device: NEC
Source-To-Source Transformation with SOTOC

- OpenMP Target Regions are outlined by using SOTOC
  - Transformation of target regions (including parameters/dependencies)
  - Integration into the clang driver
  - Use LLVM libtooling (full control of the abstract syntax tree (AST))

```c
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];
    }
}

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;
}
```

$ sotoc saxpy.c -- -fopenmp
Limitations

• 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)

• Limited macros support at the moment
  – We need to call the preprocessor twice (clang + ncc)

• Variables in #pragma omp declare target constructs
  – This is partly a Clang bug/problem
Conclusion + Next Steps

• Conclusion
  – This project benefits from LLVM infrastructure
  – Very generic approach -> suitable for other target devices
  – First prototype implementation is working

• Next Steps
  – Increase the stability of the source-2-source transformation
  – Validation with SPEC Accel benchmarks
  – Performance evaluation