LLVMs of SX-Aurora

- **LLVM-VE**
  - Open Source
  - *Inofficial* compiler for SX-Aurora

- **llvm-vec** NEC LLVM-IR Vectorizer
  - Official compiler
LLVM-VE ecosystem

- Clang
- libOpenMP
- OpenMPTarget
- libcxx(abi)
- x86-64
- Region Vectorizer

Flang (Fortran) Under development
Vectorization
Vectorization in LLVM-VE

- **Clang --target=x86_64-unknown-linux-gnu**
  - Uses LLVM upstream vectorizers (LV)
    - Inner loops only
  - Automatic vectorization

- **Clang --target=ve-linux**
  - Uses the *Region Vectorizer*
    - Outer loop vectorization
  - Automatic vectorization
  - Best controlled with pragmas
How to vectorize

◆ #pragma omp simd  [ simdlen(256) | simdlen(512) ]
  - Vectorize this loop
  - [Optional] hint for normal (256 wide) or packed mode (512 wide) vectorization

◆ #pragma omp parallel
  - May trigger vectorization (details next slides)

◆ Some unannotated loops - Automatic vectorization
  - Automatic parallel loop detection and vectorization
Region Vectorizer - Outer-loop vectorization

```c
#pragma omp simd
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        if (A[i] > 42.0)
            <do stuff>
        if (C[j])
            <do that other thing>
```

Vanilla LLVM cannot vectorize this:

- Outer loop
- Control flow (if statements) inside

Region Vectorizer can

- Will retain uniform branch in C[j]
Controlling Vectorization

```c
#pragma omp parallel for
for (int i = 0; i < n; ++i)
[..]
#pragma omp simd
for (int j = 0; j < m; ++j)
[..]
```

Parallel execution

Vectorized
Controlling Vectorization

```c
#pragma omp parallel for
for (int i = 0; i < n; ++i)
  [..]
```

```
for (int j = 0; j < n; ++j)
  [..]
```

Parallel execution

May still vectorize, if

- Loop parallelism detected
- Better score than pragma parallel loop
Diagnostics

clang -Rpass=rv -target=ve-linux -O3 clenshaw.c [...]

clenshaw.c:6:3: remark: Loop vectorized (width 256) with dynamic VL [-Rpass=rv-loopvec]
   #pragma omp simd
   ^
Adaptive math vectorization

```c
void foo (double x, ..) {
    #pragma omp simd
    for (int i = 0; i < n; ++i)
        A[i] = pow(B[i], x);
}

double pow(double, double) → double pow_vu(double256, double)
```
energy_ia Compute Kernel

- LLVM-VE + Packed Mode

<table>
<thead>
<tr>
<th>energy_ia</th>
<th>LLVM-VE-RV (not packed)</th>
<th>LLVM-VE-RV (packed)</th>
<th>NCC 3.0.6*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime [ms]</td>
<td>0.284</td>
<td>0.188</td>
<td>0.437</td>
</tr>
<tr>
<td>Speedup [rel to ncc]</td>
<td>35%</td>
<td>56%</td>
<td>-</td>
</tr>
</tbody>
</table>

clang --target=ve-linux -fopenmp-simd -O3 -ffast-math

ncc -ffast-math (version 3.0.6)

sx-at-test version of energy_ia

NEC SX-Aurora VE10B – one thread
OpenMP [Target]
LLVM OpenMP

• libOpenMP  
  #pragma omp parallel
  - Implements parallel loops, barriers and reductions
  - Linked against the device code
    • Either as part of VE-native application
    • Or VE-native kernels

• libOpenMPTarget  
  #pragma omp target
  - Performs the kernel dispatch, buffer transfers
  - Linked against the host application
  - Plugin mechanism for actual offload (see Tim Cramer’s talk)
LLVM OpenMP

• Generic LLVM OpenMP library compiled for VE
  
  ```
  #pragma omp parallel
  ```

• Pro: Mature OpenMP implementation
  
  Standard OpenMP runtime for x86 for Clang/LLVM

• Con: Not tuned for vector architectures
  
  Based on pthreads, standard synchronization primitives (futex), not hw features
## OpenMP – EPCC Syncbench

<table>
<thead>
<tr>
<th></th>
<th>OpenMP (ncc)</th>
<th>LLVM OpenMP (VE)</th>
<th>LLVM OpenMP (x86)</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel for</td>
<td>6.77</td>
<td>724.4</td>
<td>7.27</td>
</tr>
<tr>
<td>barrier</td>
<td>3.74</td>
<td>309.8</td>
<td>1.87</td>
</tr>
<tr>
<td>reduction</td>
<td>7.01</td>
<td>608.5</td>
<td>7.5</td>
</tr>
</tbody>
</table>

NEC OpenMP is fast    LLVM OpenMP needs tuning
LLVM OpenMPTarget

```c
#pragma omp target
```

- AVEO plugin
  - VH → VE Offloading
- VHCall plugin
  - VE → VH Offloading

- SOLLVE OpenMP Target Verification suite
OpenMP Target - SOLLVE C

<table>
<thead>
<tr>
<th>VH $\rightarrow$ VE</th>
<th>VE $\rightarrow$ VH</th>
<th>VH $\rightarrow$ VE (sotoc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile Error</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Runtime Error</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Passed</td>
<td>98</td>
<td>101</td>
</tr>
</tbody>
</table>

Conformant LLVM code path

Custom source-to-source
## OpenMP Target - SOLLVE C++

<table>
<thead>
<tr>
<th></th>
<th>VH → VE</th>
<th>VE → VH</th>
<th>VH → VE (sotoc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile Error</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>Runtime Error</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Passed</td>
<td>13</td>
<td>12</td>
<td>0</td>
</tr>
</tbody>
</table>

- Conformant LLVM code path
- Source-to-source for C only
LLVM Test suite
LLVM Test Suite

C, C++ (-O3, -ffast-math)
- 432 applicable C/C++ tests (for both NCC and Clang)

<table>
<thead>
<tr>
<th></th>
<th>LLVM-VE-RV dev</th>
<th>NCC 3.0.6*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile fail</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>Compile pass</td>
<td>430 (99.5 %)</td>
<td>332 (77 %)</td>
</tr>
<tr>
<td>Test fail</td>
<td>57</td>
<td>57</td>
</tr>
<tr>
<td>Test pass</td>
<td>373 (86%)</td>
<td>275 (64%)</td>
</tr>
</tbody>
</table>

* translating clang/gcc options into ncc options (wrapper script)
Future Work

• Improving LLVM’s support for vector architectures

• Tuning of OpenMP Runtime

• Better Code Generation

• Making VE an official LLVM Backend

github.com/sx-aurora-dev/llvm-project