SC23 Booth Talk Series

GPU Warp-Level Parallelism in LLVM/OpenMP

Eric Wright, University of Delaware
Goals of this project

• Give explicit control of warp-level parallelism to the programmer in the OpenMP programming model through OpenMP’s “simd” directive
• Design a model for GPUs that fits the CPU-centric programming scheme of OpenMP
• Create an optimized model that more closely follows GPU programming best practices
Outline

• OpenMP GPU Execution Model
  – Generic “CPU-Centric” Model
  – SPMD “GPU-Centric” Model
• OpenMP Code Generation
• Implementing OpenMP Warp-Level Parallelism
  – Generic Model
  – SPMD Model
  – Results
OPENMP GPU EXECUTION MODEL
Threads to GPU Hardware

Thread Block

- warp0
  - t0, t1, t2, t3
  - t28, t29, t30, t31

- warp1
  - t32, t33, t34, t35
  - t60, t61, t62, t63

- warp2
  - t64, t65, t66, t67
  - t92, t93, t94, t95

- warp3
  - t96, t97, t98, t99
  - t124, t125, t126, t127

Streaming Multiprocessor (SM)

- Registers/Cache
- Shared Memory
OpenMP Parallelism to GPU Threads
Simplified OpenMP Model

Main Thread

Execute Sequential Code

#pragma omp parallel

Execute Parallel Code

#pragma omp end parallel

Worker Thread 1

Execute Parallel Code

Worker Thread 2

Execute Parallel Code
#pragma omp target teams 
{ 
    float Val = some_fn();
    if(Val > 0) {
        #pragma omp parallel
            <Body1>
    } else {
        #pragma omp parallel
            <Body2>
    }
} 

• Main thread begins execution of the teams region
• The main thread calculates the value of “Val”
  • Worker threads should not compute “Val”
• Worker threads then need to be notified about which branching path should be taken

CPU-Centric OpenMP

OpenMP

```c
#pragma omp target teams
{
    float Val = some_fn();
    if(Val > 0) {
        #pragma omp parallel
        <Body1>
    } else {
        #pragma omp parallel
        <Body2>
    }
}
```

CUDA

```c
__shared__ int Val;
if(ThreadIdx.x == 0)
    Val = some_fn();
__syncthreads();
if(Val > 0) {
    <Body1>
} else {
    <Body2>
}
```

"Generic" GPU Execution Model

- Threads diverge immediately when region is initialized
  - Main thread executes user code
  - Worker threads wait idle for work
- Once the main thread encounters a parallel region the worker threads are notified of which region should be executed

Optimized GPU "SPMD" Execution Model

- Main and worker threads execute the entire region
  - Similar to how GPU kernel languages function
- Only works when full execution does not produce "side-effects"
  - Simplest case is when parallel regions are tightly-nested
- First implemented in IBM XL compiler

Optimizing CPU-Centric Code on GPUs

• Since OpenMP is a CPU-centric model, some codes translate poorly onto GPUs
  – i.e the existence of Generic mode
• Bad choices regarding memory usage are often made to ensure conformability to the CPU model
• Many Generic-mode codes could execute in SPMD-mode with proper thread guarding and synchronization to remove side-effects
LLVM/CLANG CODE GENERATION
Code Generation Summary

• Code generation done in the *OpenMP IR Builder*
  – Provides front-end portability for any OpenMP-enabled compiler
• Loop tasks (i.e the body of the loop) are packaged as separate functions to be passed into the OpenMP runtime
  – The runtime handles all of the important thread scheduling
An OpenMP SIMD Loop

#pragma omp simd
for(i=1; i<N; i++)
    printf("Hello world! %i", i);
An OpenMP SIMD Loop

```c
#pragma omp simd
for(i=1; i<N; i++)
{printf("Hello world! %i", i);}
```

Loop variable - `i`

Loop body - task

Trip count – N-1 iterations
An OpenMP SIMD Loop

void outlined_fn(int Iter) {
    int i = Iter+1;
    printf("Hello world! %i", i);
}

Device Runtime
OPENMP SIMD IN LLVM’S RUNTIME LIBRARY
OpenMP SIMD

• Single Instruction, Multiple Data (SIMD)
• “The simd construct can be applied to a loop to indicate that the loop can be transformed into a SIMD loop (that is, multiple iterations of the loop can be executed concurrently by using SIMD instructions).”
• For GPUs, this means iterations should be parallelized across adjacent threads within a warp
OpenMP Parallelism to GPU Hardware

Team Main Thread

SIMD Leader

SIMD Worker

#pragma omp teams

block 0

warp 0

warp 1

warp 2

warp 3

block 1

warp 0

warp 1

warp 2

warp 3

#pragma omp teams

#pragma omp simd
SIMD Generic Exec. Model

- **Target Init**
- **Team Main**
  - Assign parallel workload
  - Sync threads
- **Run User Code**
  - Could contain multiple serial and parallel regions
- **Target Deinit**

**Workers**
- Enter state machine
- Sync threads
- Fetch parallel workload
- Execute parallel workload
  - Could contain multiple serial regions and SIMD loops
- Sync threads
- **SimD Workers**
  - Assign SIMD workload
  - Sync warp
  - Execute SIMD workload
  - Sync warp
  - Fetch SIMD workload
  - Execute SIMD workload
  - Sync warp
SPMD Exec. Model

target init

run user code (could contain multiple serial and parallel regions)

team main

assign parallel workload

sync threads

assign parallel workload

target deinit

workers

enter state machine

sync threads

fetch parallel workload

execute parallel workload (could contain multiple serial regions and simd loops)

assign simd workload

sync warp

execute simd workload

fetch simd workload

execute simd workload

sync warp

sync warp

simd state machine
SPMD Exec. Model
(from the viewpoint of a worker thread)
PERFORMANCE RESULTS
Experimental Setup: Perlmutter

• Perlmutter is a HPE (Hewlett Packard Enterprise) Cray EX supercomputer
• 4,864 compute nodes with AMD EPYC 7763 processors
• 1,536 GPU nodes (+ 256 80GB GPU nodes)
  – 4 NVIDIA A100 GPUs per node
Speedup of SIMD for codes with a clear benefit
Cost of SIMD for codes with no clear benefit

Speedup over two-leveled baseline (higher is better)

<table>
<thead>
<tr>
<th></th>
<th>No SIMD</th>
<th>Generic SIMD</th>
<th>SPMD SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laplace3D</td>
<td>1.00</td>
<td>0.82</td>
<td>1.02</td>
</tr>
<tr>
<td>muram_Transpose</td>
<td>1.00</td>
<td>0.87</td>
<td>0.97</td>
</tr>
<tr>
<td>muram_Interpol</td>
<td>1.00</td>
<td>0.87</td>
<td>1.03</td>
</tr>
</tbody>
</table>
An OpenMP SIMD Loop

void outlined_fn(int Iter) {
    int i = Iter+1;
    printf("Hello world! %i", i);
}

SIMD Generic Exec. Model

Speedup of SIMD for codes with a clear benefit
Eric Wright – efwright@udel.edu
Sunita Chandrasekaran – schandra@udel.edu

https://crpl.cis.udel.edu/
OpenMP SC23 Booth Talk Series

openmp.org  OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc23  OpenMP SC23 booth talk videos and presentations