OpenMP

SC23 Booth Talk Series

GPU Warp-Level Parallelism in LLVM/OpenMP

Eric Wright, University of Delaware

Goals of this project

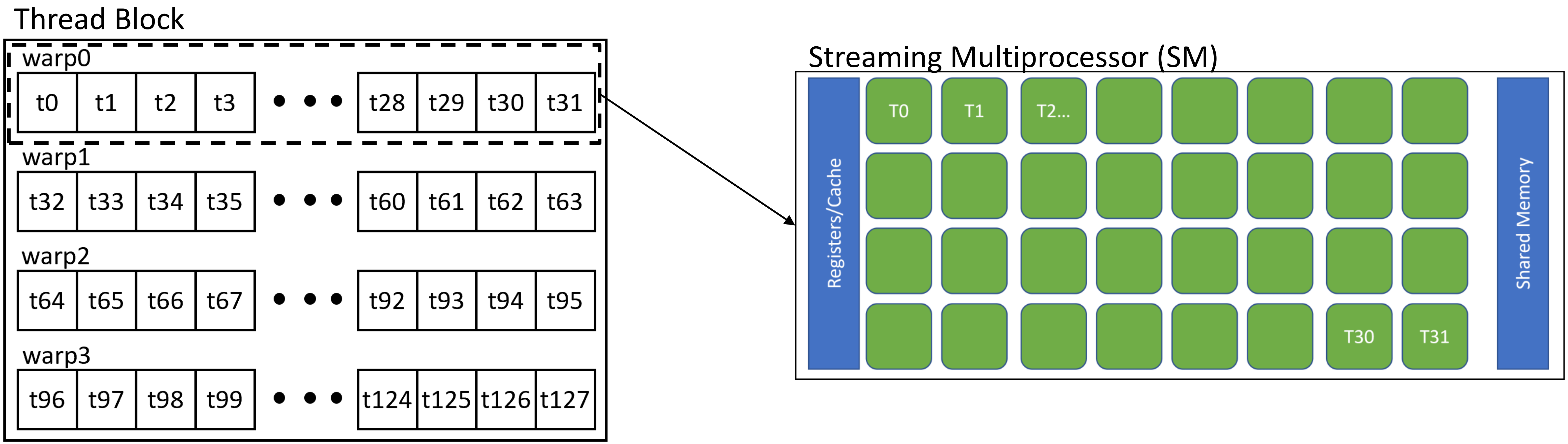
- Give explicit control of warp-level parallelism to the programmer in the OpenMP programming model through OpenMP's "simd" directive
- Design a model for GPUs that fits the CPU-centric programming scheme of OpenMP
- Create an optimized model that more closely follows GPU programming best practices

Outline

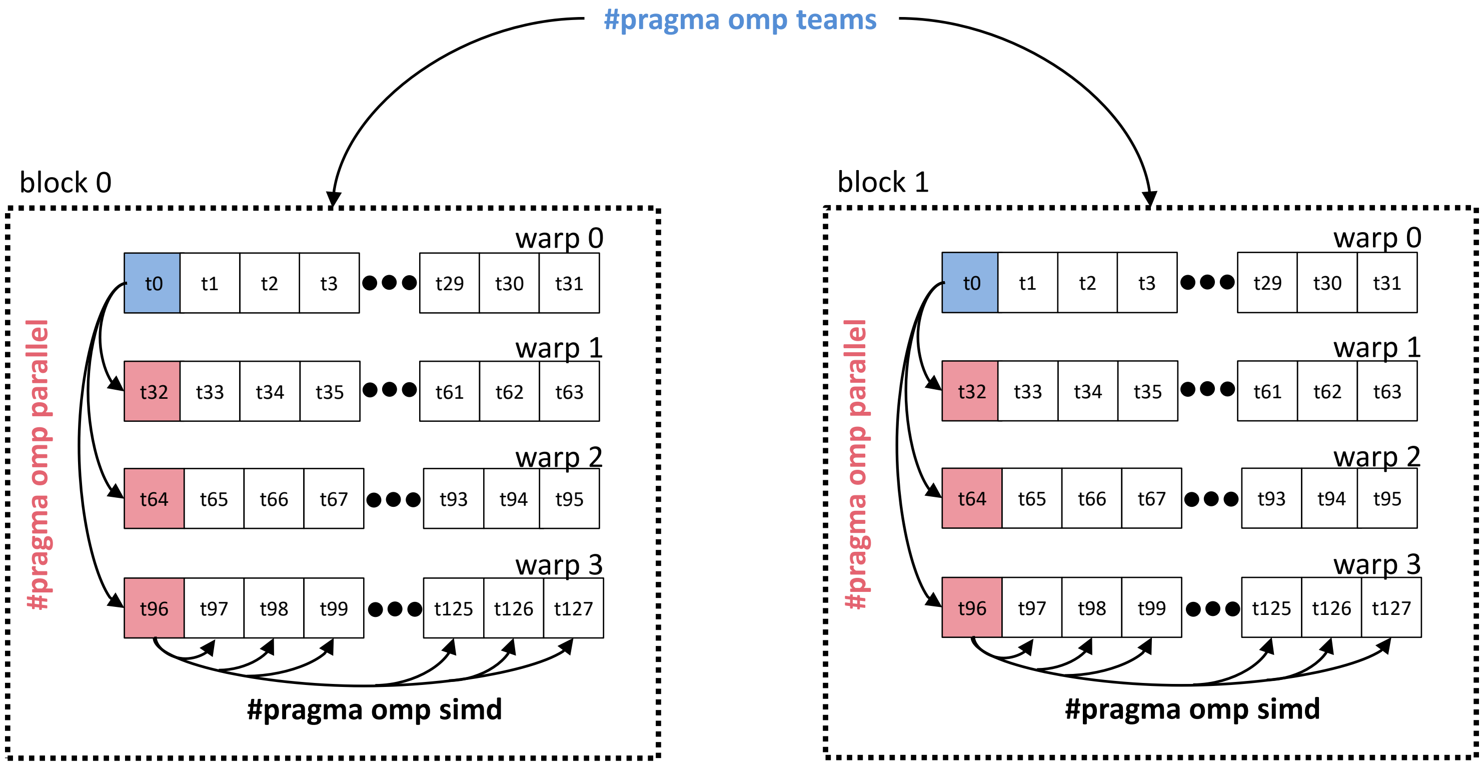
- OpenMP GPU Execution Model
  – Generic "CPU-Centric" Model
  – SPMD "GPU-Centric" Model
- OpenMP Code Generation
- Implementing OpenMP Warp-Level Parallelism
  – Generic Model
  – SPMD Model
  – Results

OPENMP GPU EXECUTION MODEL

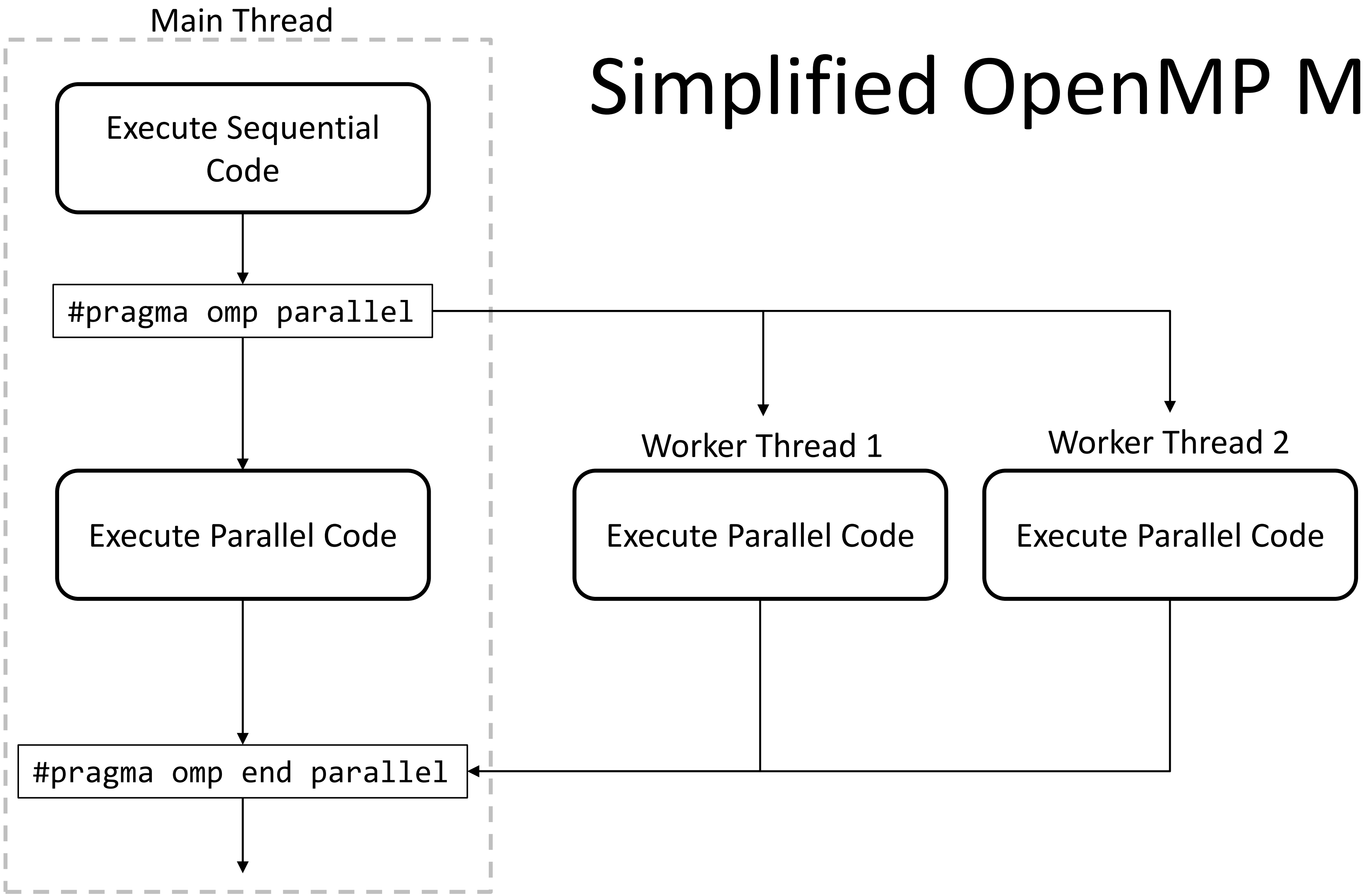
Threads to GPU Hardware



OpenMP Parallelism to GPU Threads



Simplified OpenMP Model



CPU-Centric OpenMP

```
#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

Bertolli, C., Antao, S. F., Eichenberger, A. E., Sura, K. O. Z., Jacob, A. C., Chen, T., & Sallenave, O. (2014, November). Coordinating GPU threads for OpenMP 4.0 in LLVM. In *2014 LLVM Compiler Infrastructure in HPC* (pp. 12-21). IEEE.

CPU-Centric OpenMP

OpenMP

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

CUDA

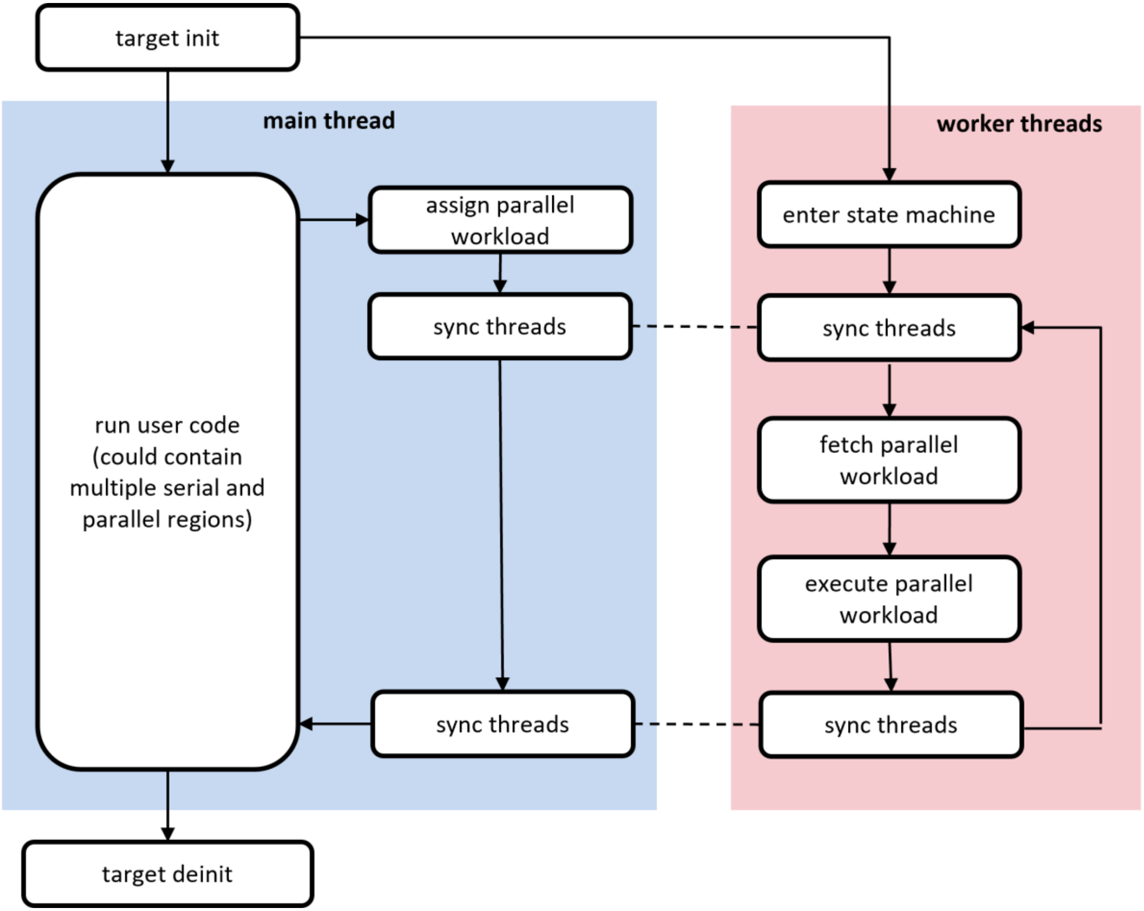
```
__shared__ int Val;
if(ThreadIdx.x == 0)
   Val = some_fn();
__syncthreads();

if(Val > 0) {
   <Body1>
} else {
   <Body2>
}
```

Bertolli, C., Antao, S. F., Eichenberger, A. E., Sura, K. O. Z., Jacob, A. C., Chen, T., & Sallenave, O. (2014, November). Coordinating GPU threads for OpenMP 4.0 in LLVM. In *2014 LLVM Compiler Infrastructure in HPC* (pp. 12-21). IEEE.

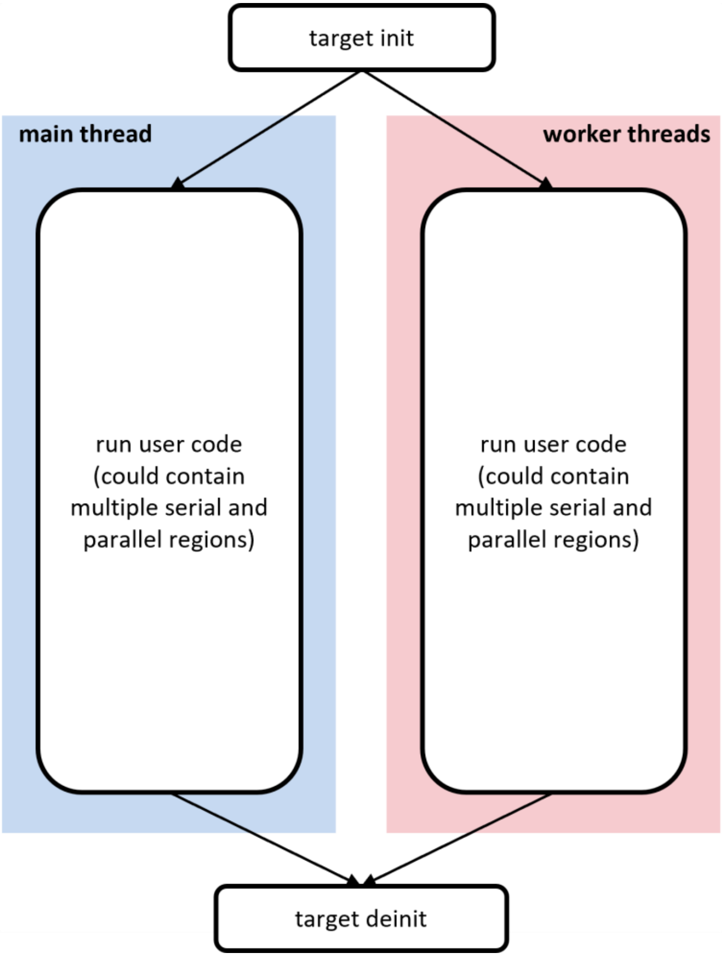
“Generic” GPU Execution Model

- 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



Jacob, A. C., Eichenberger, A. E., Sung, H., Antão, S. F., Bercea, G. T., Bertolli, C., ... & O'Brien, K. (2017, December). Efficient fork-join on GPUs through warp specialization. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)* (pp. 358-367). IEEE.

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

Tiotto, E., Mahjour, B., Tsang, W., Xue, X., Islam, T., & Chen, W. (2019). OpenMP 4.5 compiler optimization for GPU offloading. *IBM Journal of Research and Development*, *64*(3/4), 14-1.

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

Huber, J., Cornelius, M., Georgakoudis, G., Tian, S., Diaz, J. M. M., Dinel, K., ... & Doerfert, J. (2022, April). Efficient execution of OpenMP on GPUs. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (pp. 41-52). IEEE.

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

Loop variable - i

Trip count – N-1 iterations

Loop body - task

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

An OpenMP SIMD Loop

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

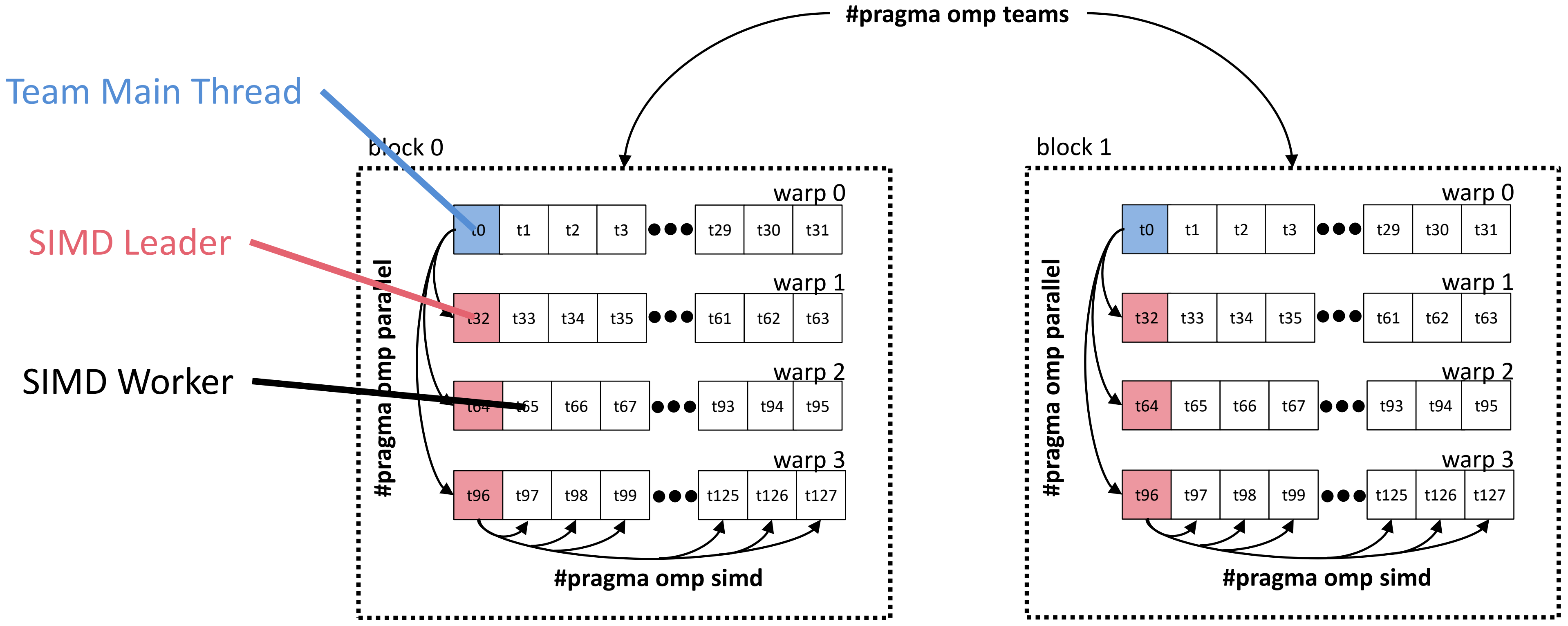
Device Runtime

OPENMP SIMD IN LLVM'S RUNTIME LIBRARY

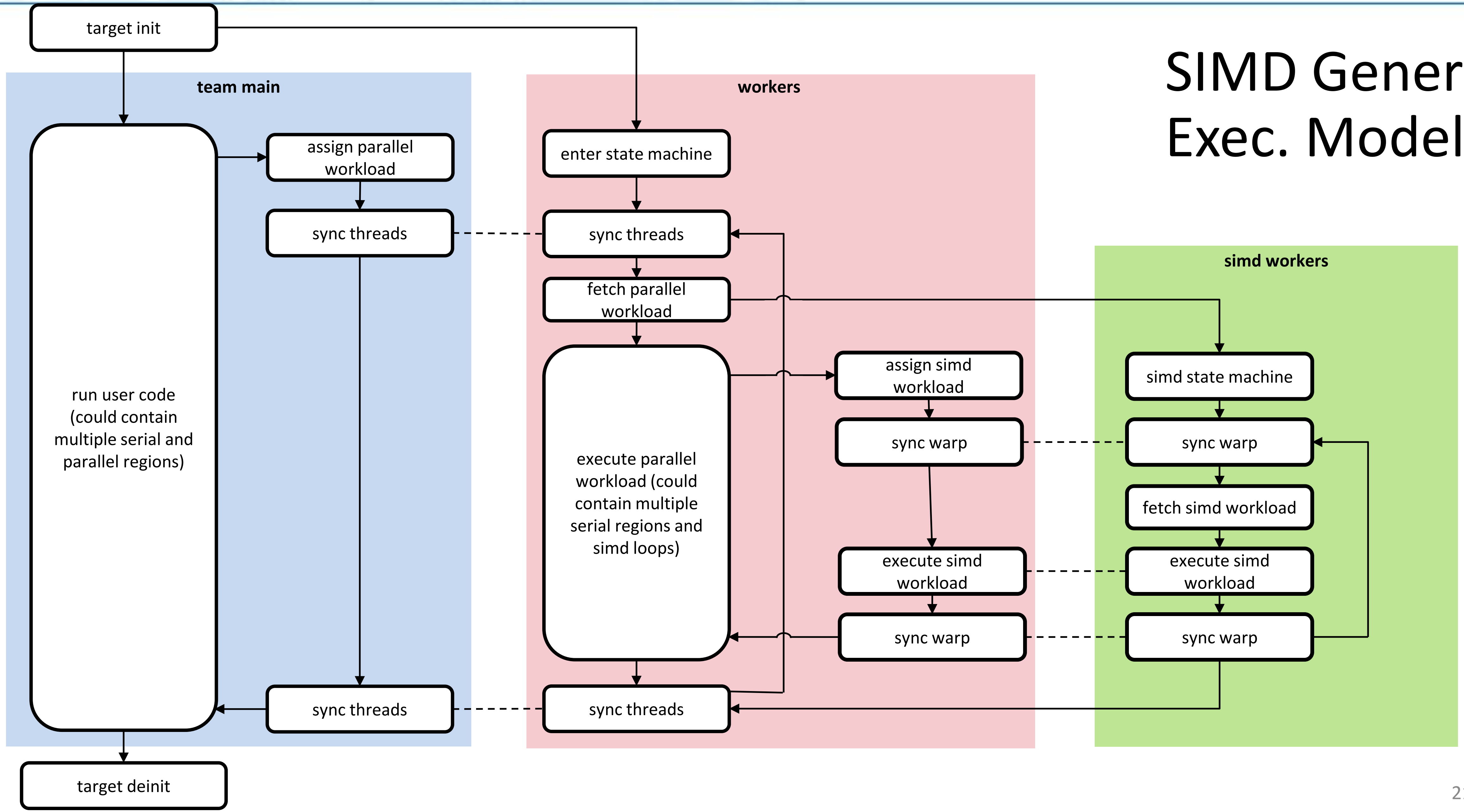
OpenMP SIMD

- 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

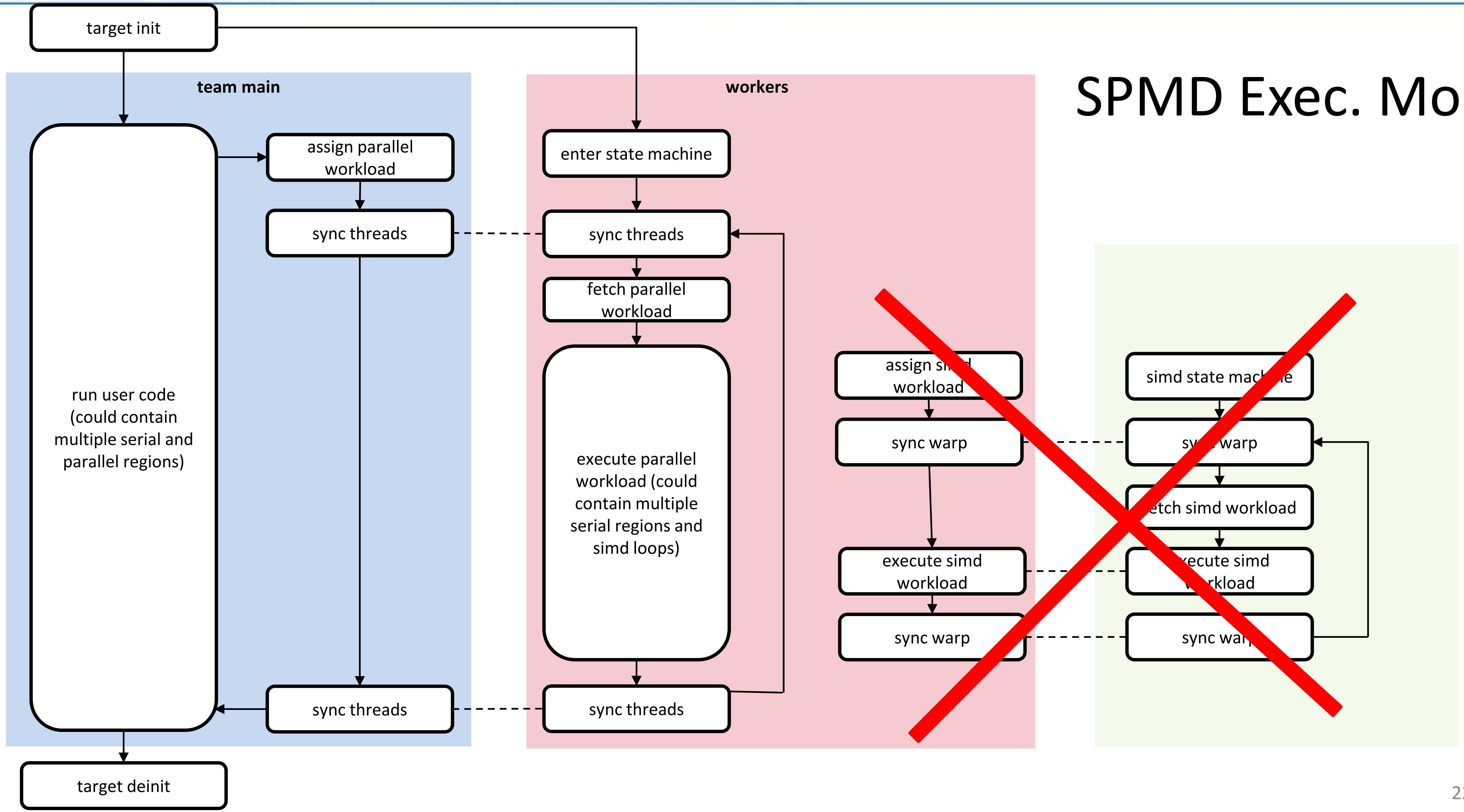
OpenMP Parallelism to GPU Hardware



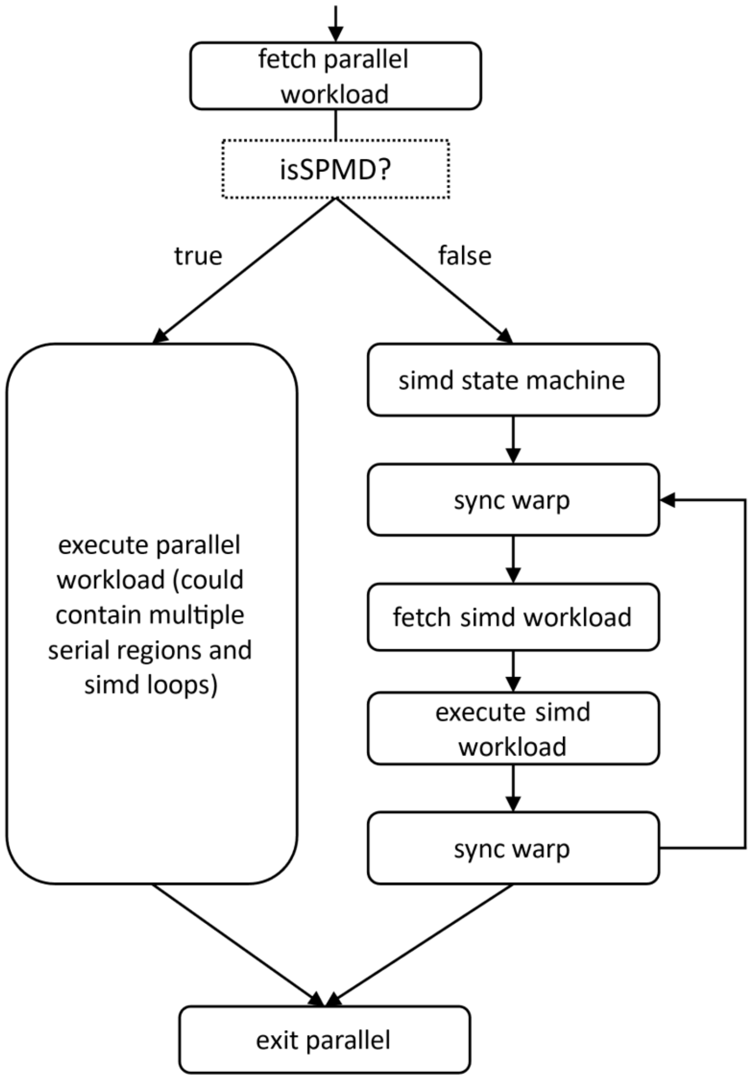
SIMD Generic Exec. Model



SPMD Exec. Model



SPMD Exec. Model (from the viewpoint of a worker thread)

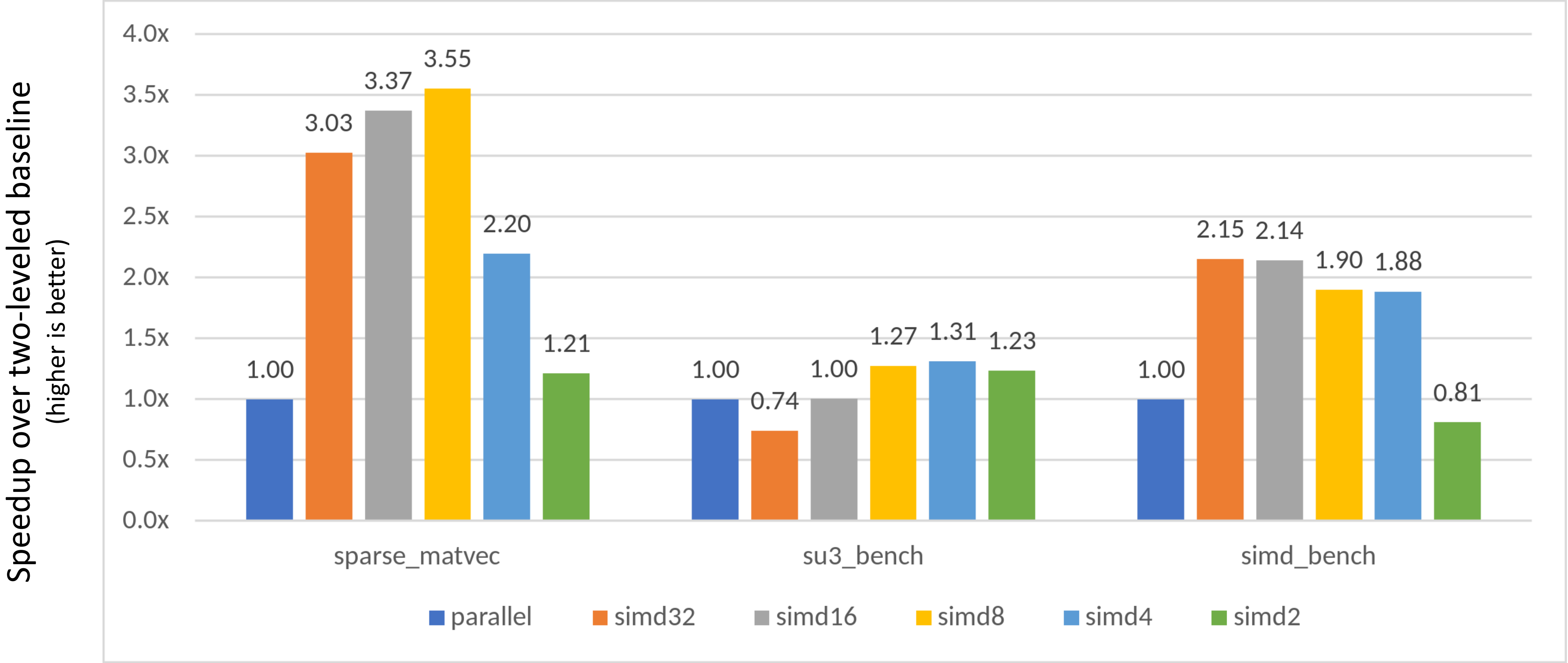


PERFORMANCE RESULTS

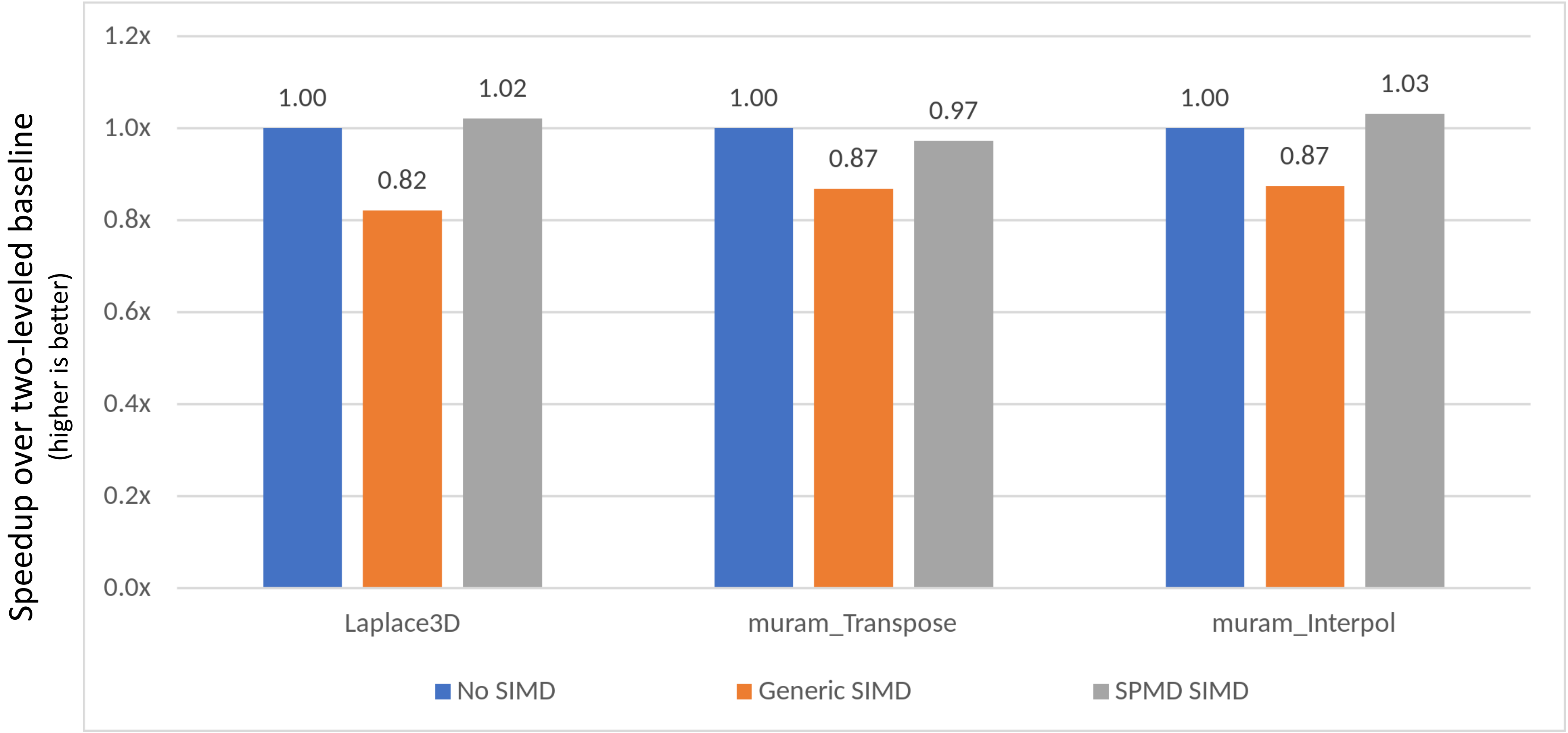
Experimental Setup: Perlmutter

- 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

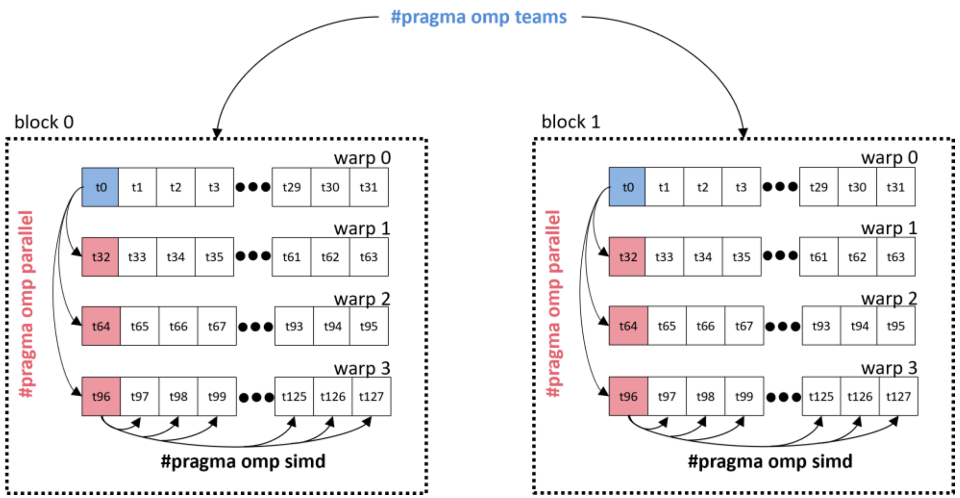
Speedup of SIMD for codes with a clear benefit



Cost of SIMD for codes with no clear benefit



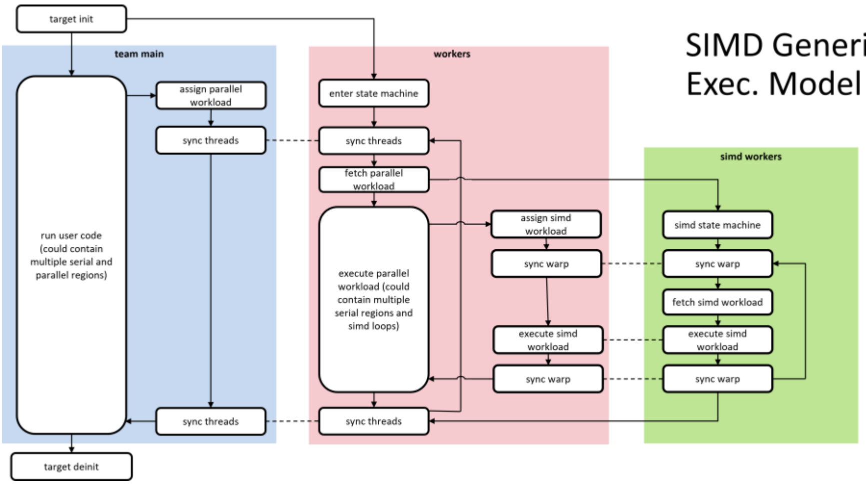
An OpenMP SIMD Loop



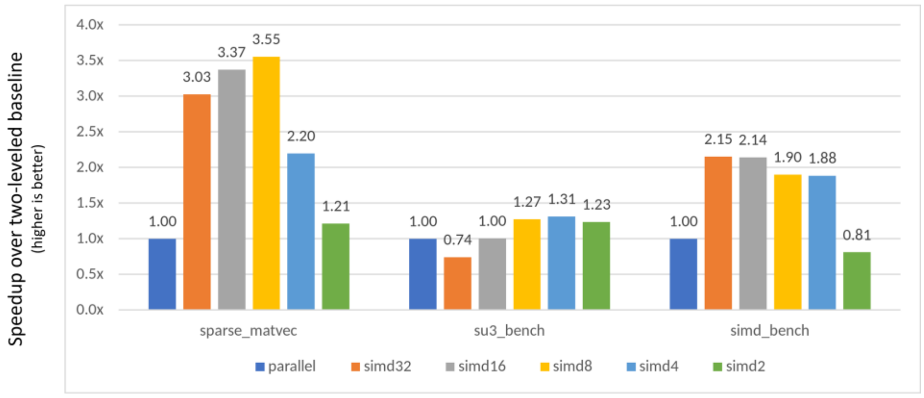

```
void outlined_fn(int Iter) {
  int i = Iter+1;
  printf("Hello world! %i", i);
}
```

Device Runtime

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