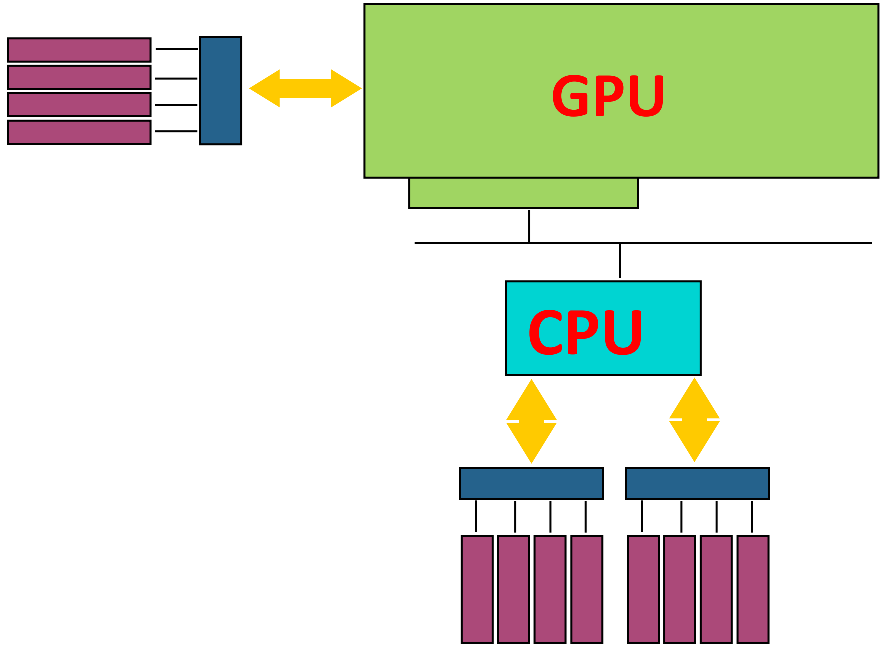
S6240 - High-Level GPU Programming Using OpenMP 4.5 and Clang/LLVM

Arpith Jacob, Alexandre Eichenberger, Samuel Antao, Carlo Bertolli, Tong Chen, Zehra Sura, Hyojin Sung, Georgios Rokos, Kevin O'Brien

IBM T. J. Watson Research Center

Overview

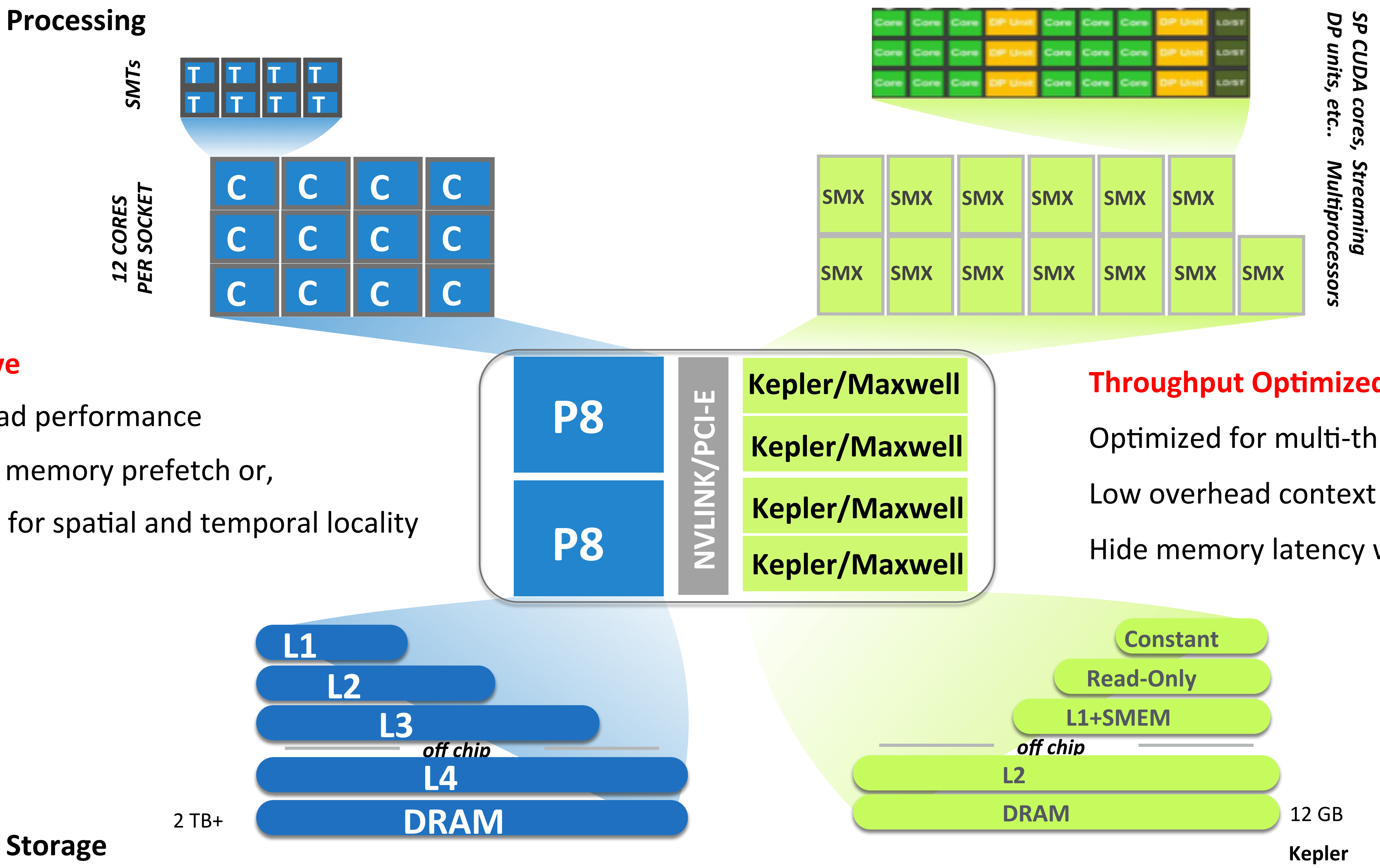
- IBM is building heterogeneous systems with Power + GPU



- Advocating the use of the OpenMP programming model
- IBM Research is contributing OpenMP support for NVIDIA GPUs in Clang/LLVM
- Upstreaming in progress. download at: [ibm.biz/ykt-omp](ibm.biz/ykt-omp)

Exploiting Heterogeneous Node Resources

IBM



Latency Sensitive

High single thread performance

Hide latency via memory prefetch or,

Cache hierarchy for spatial and temporal locality

Throughput Optimized

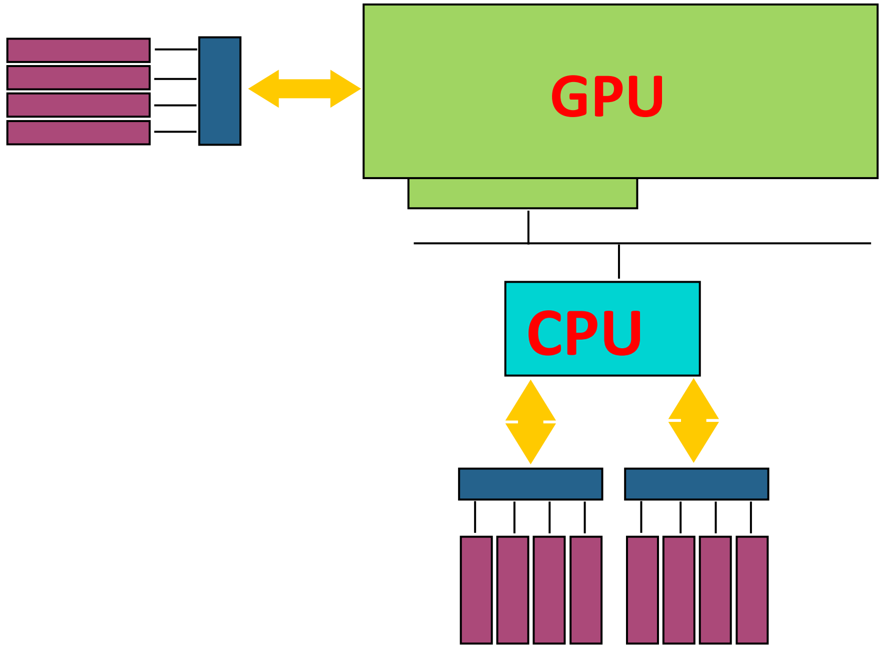
Optimized for multi-threaded code

Low overhead context switch

Hide memory latency with threads

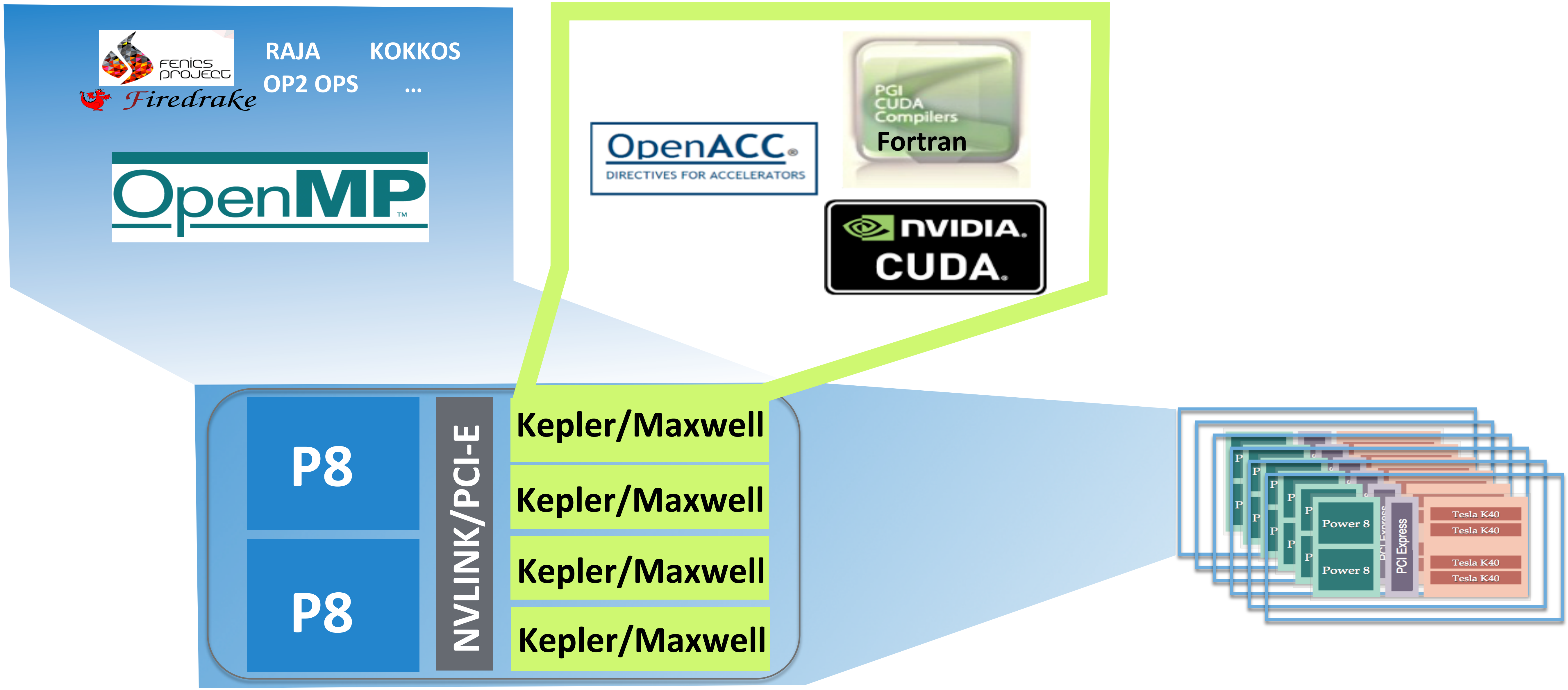
Programmability Challenge

- Applications must exploit heterogeneous resources in a performance portable manner



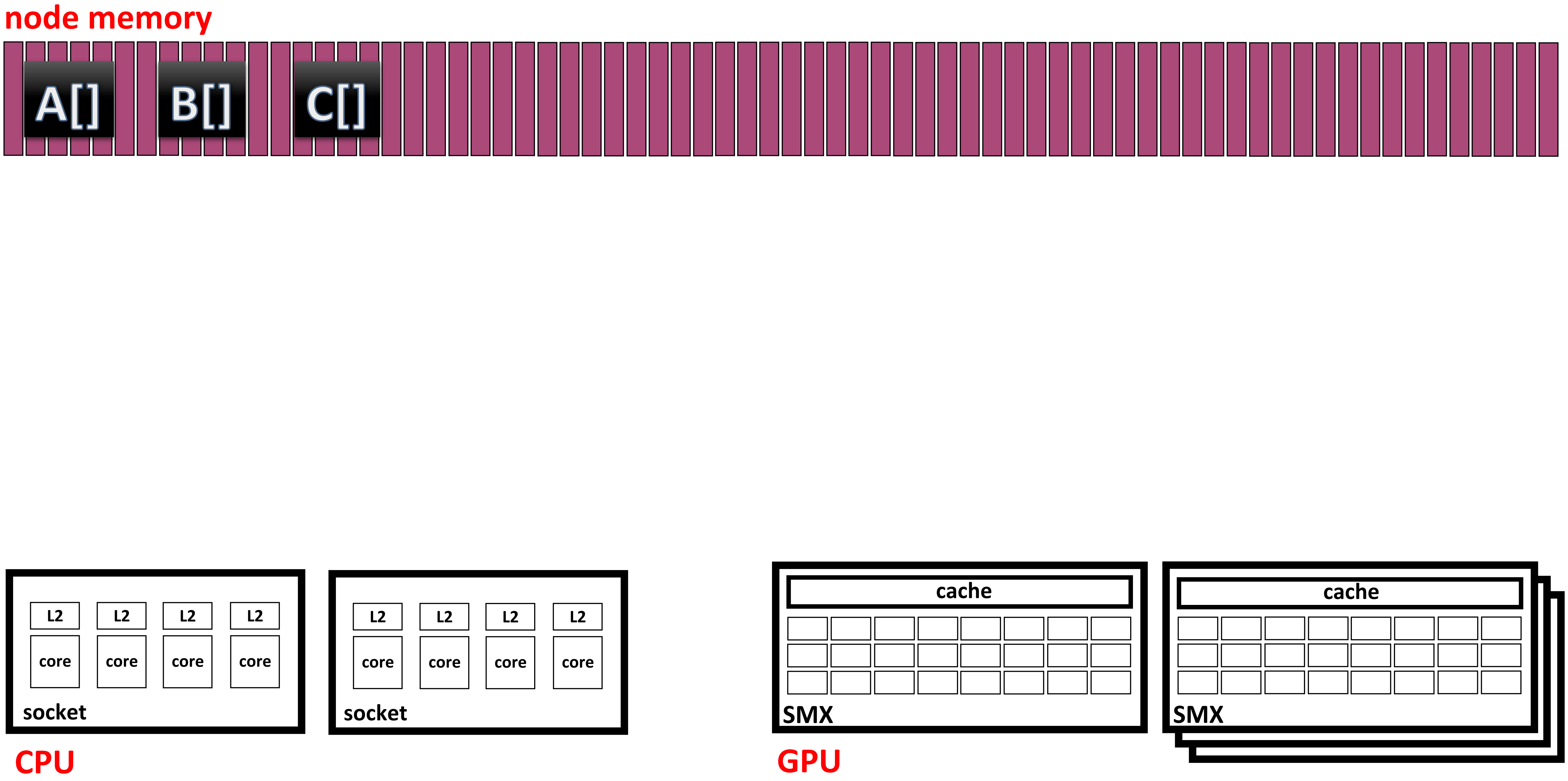
- Use vendor specific languages and directives?
- Compiler specific pragmas?
- Mix of programming models? OpenMP, OpenACC, CUDA

Programming Overview

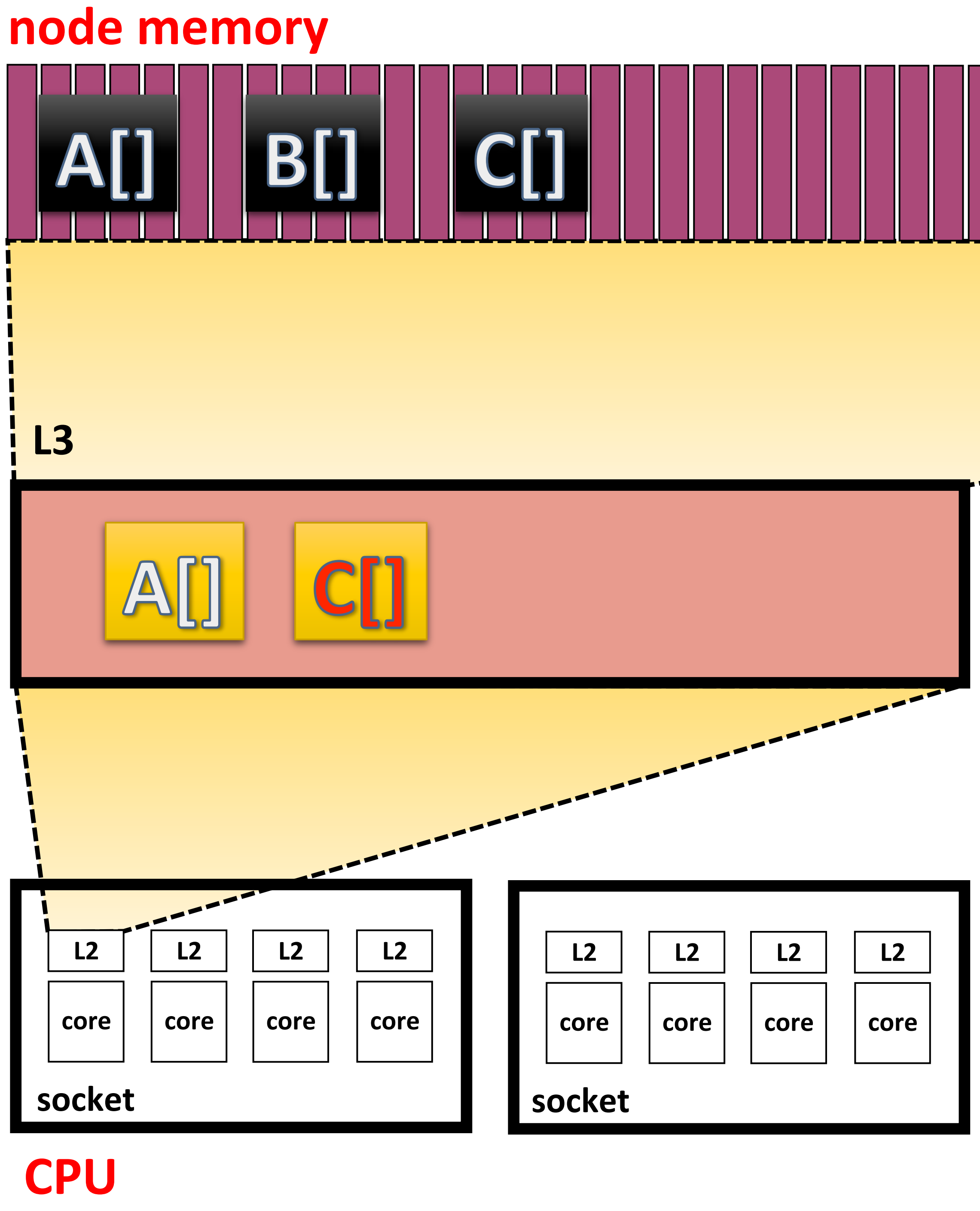


- OpenMP is widely used to program CPUs; latest specs support accelerators
- Write **performance portable** code using **flexible parallelism models**
- **Industry-wide acceptance:** IBM, Intel, PathScale, Cray, PGI, Oracle, MS

OpenMP Memory Model

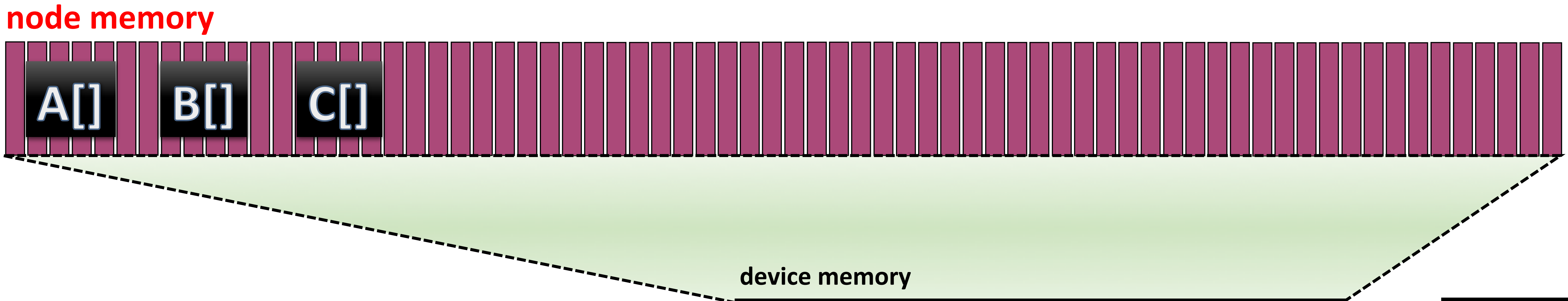


OpenMP Memory Model

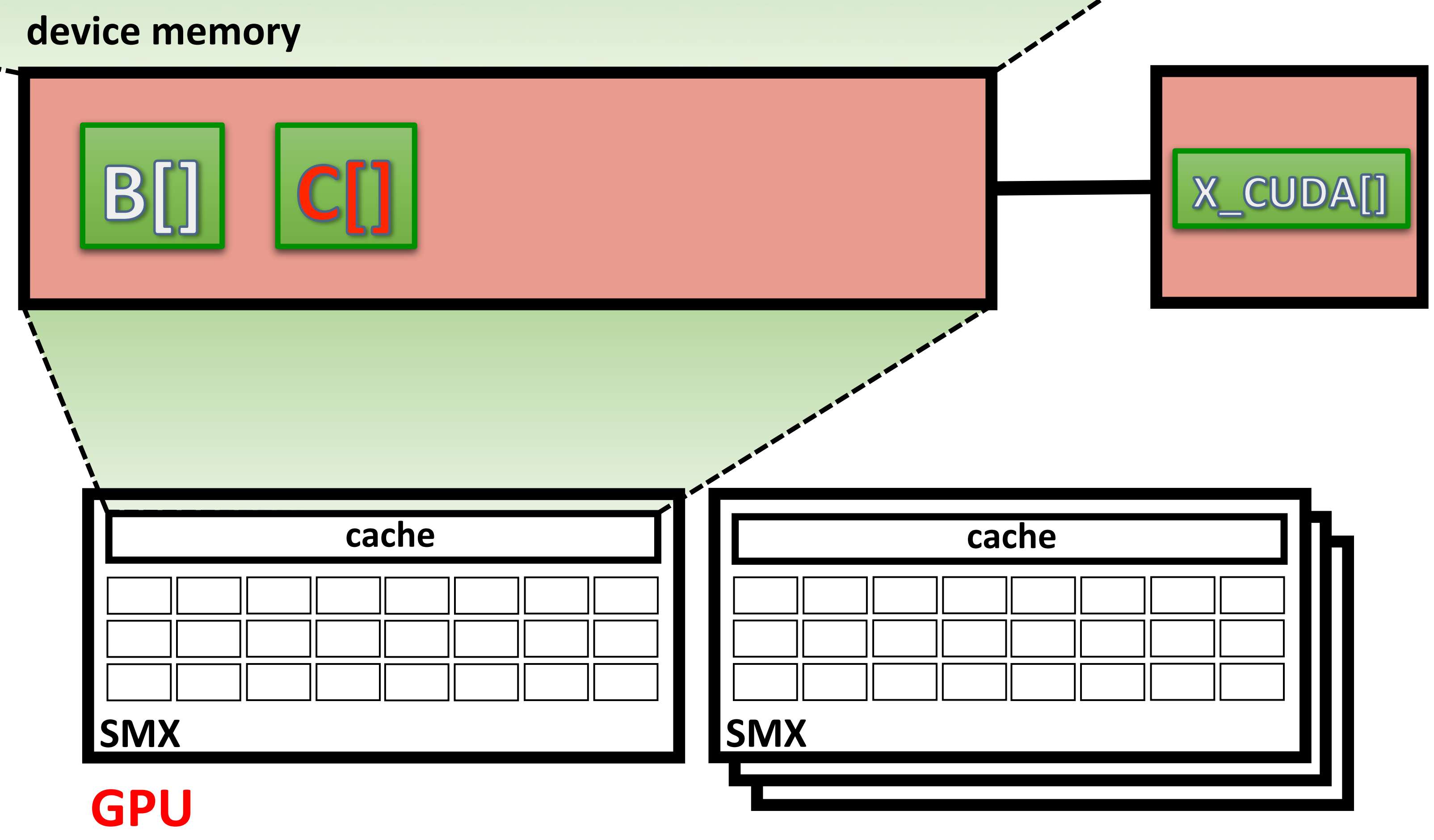


- CPU threads may have their own view of shared variables
  - Relaxed consistency
  - Explicit flush operations on host required
- Unsynchronized accesses may lead to data races

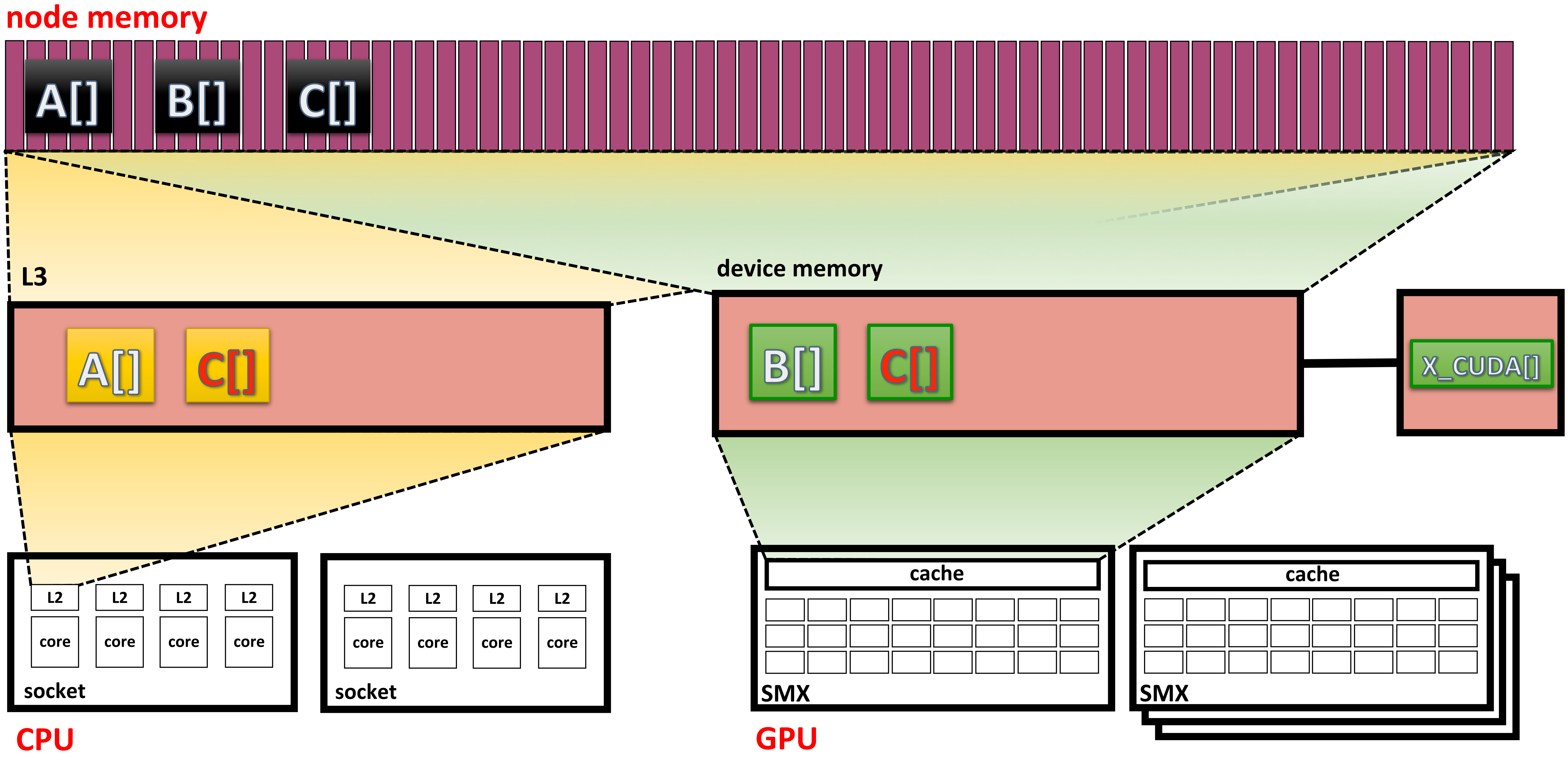
OpenMP Memory Model



- OMP4 extends views to target devices
  - Map: control data views
  - Target data enter/exit
  - Target update
- Unsynchronized accesses may lead to data races



OpenMP Memory Model

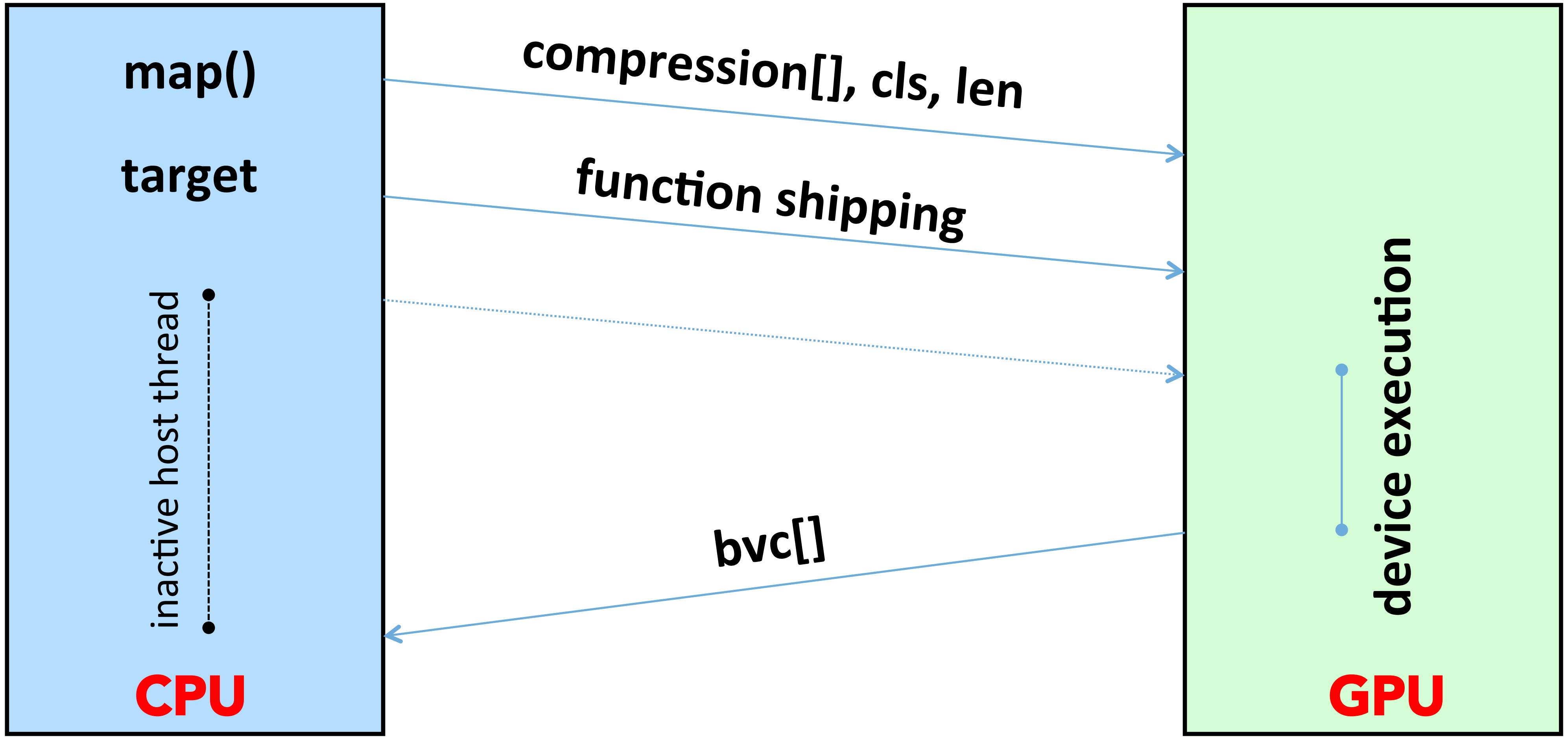


OpenMP Offload Model

How do we use OpenMP offload?

```
#pragma omp target map(to: cls, len, compression[0:len])   \
                   map(from: bvc[0:len])
for (int i=0; i<len; i++) {
    bvc[i] = cls * (compression[i] + 1.0);
}
```

OpenMP Offload Model

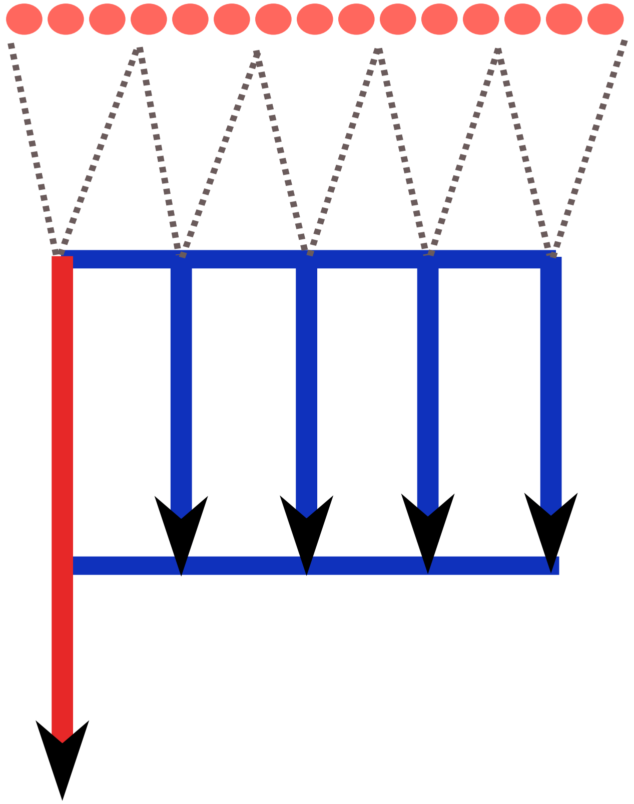



```
#pragma omp target map(to: cls, len, compression[0:len])   \
                   map(from: bvc[0:len])
for (int i=0; i<len; i++) {
    bvc[i] = cls * (compression[i] + 1.0);
}
```

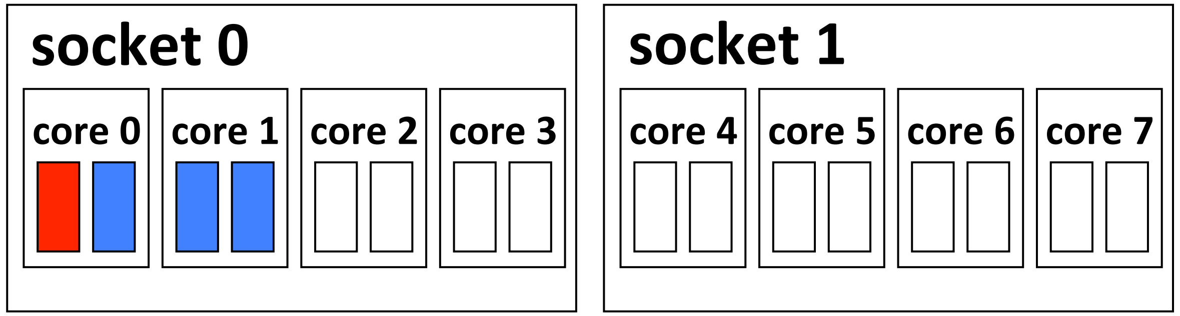
Exploit Hardware Threads on POWER CPUs

Loop work-sharing

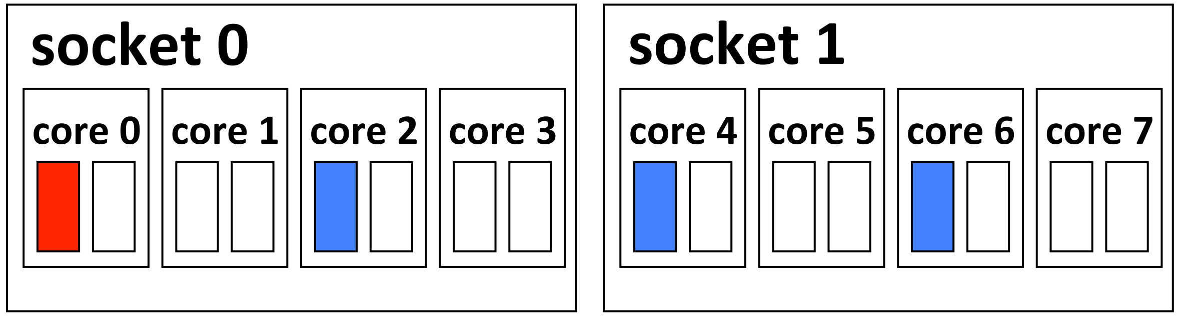
```
#pragma omp parallel for
for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
        A[i][j] += u1[i] * v1[j] + u2[i] * v2[j];
```



Affinity: pack threads to reuse cache locality

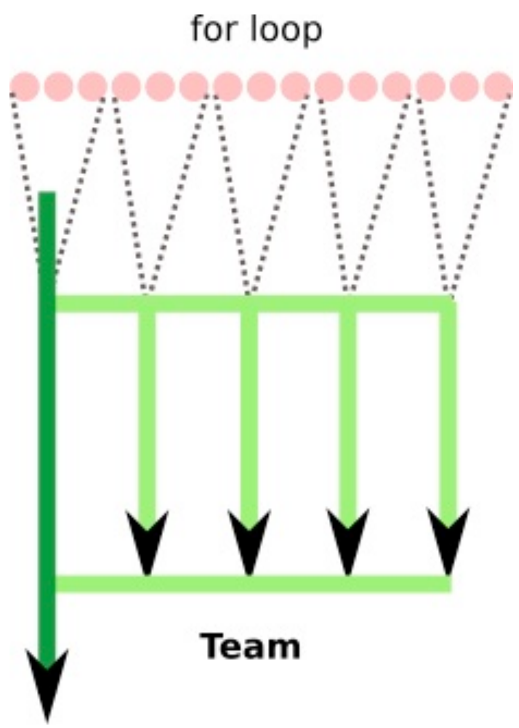


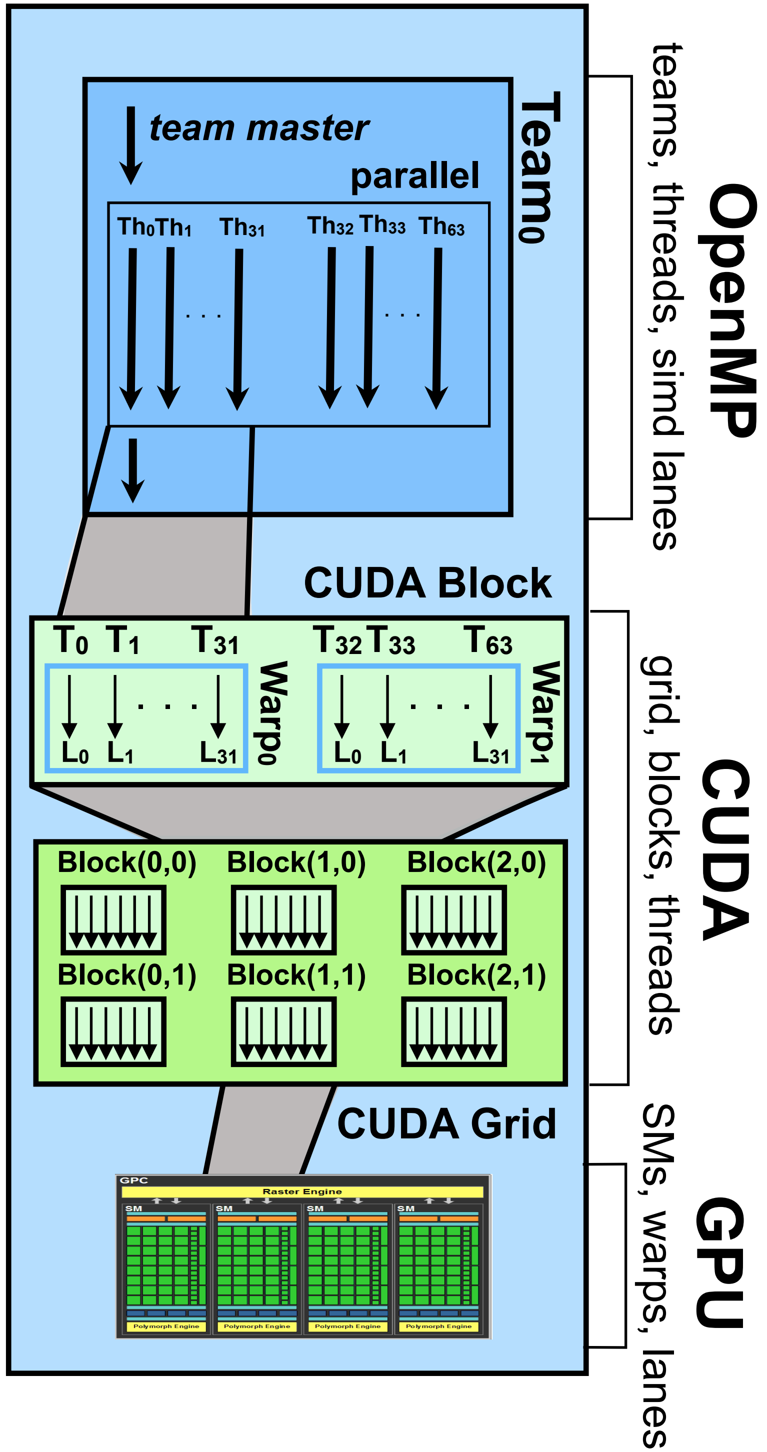
Affinity: spread threads to maximize bandwidth



Loop work-sharing on GPUs with a target task

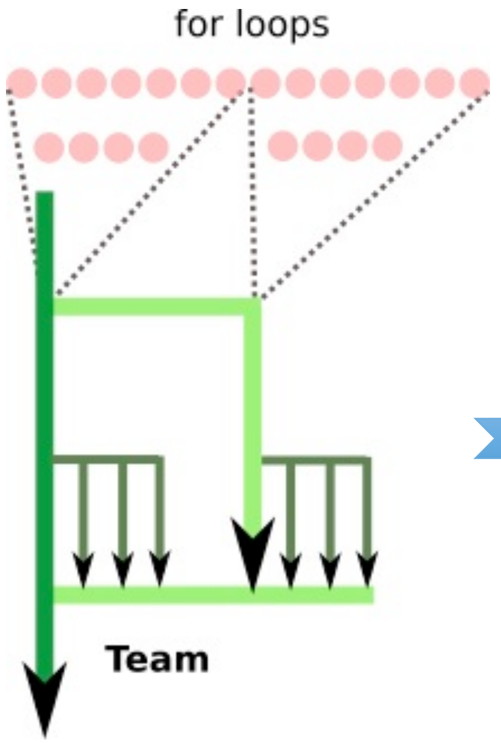
```
#pragma omp target teams distribute
#pragma omp parallel for
for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
        A[i][j] += u1[i] * v1[j] +
                   u2[i] * v2[j];
```

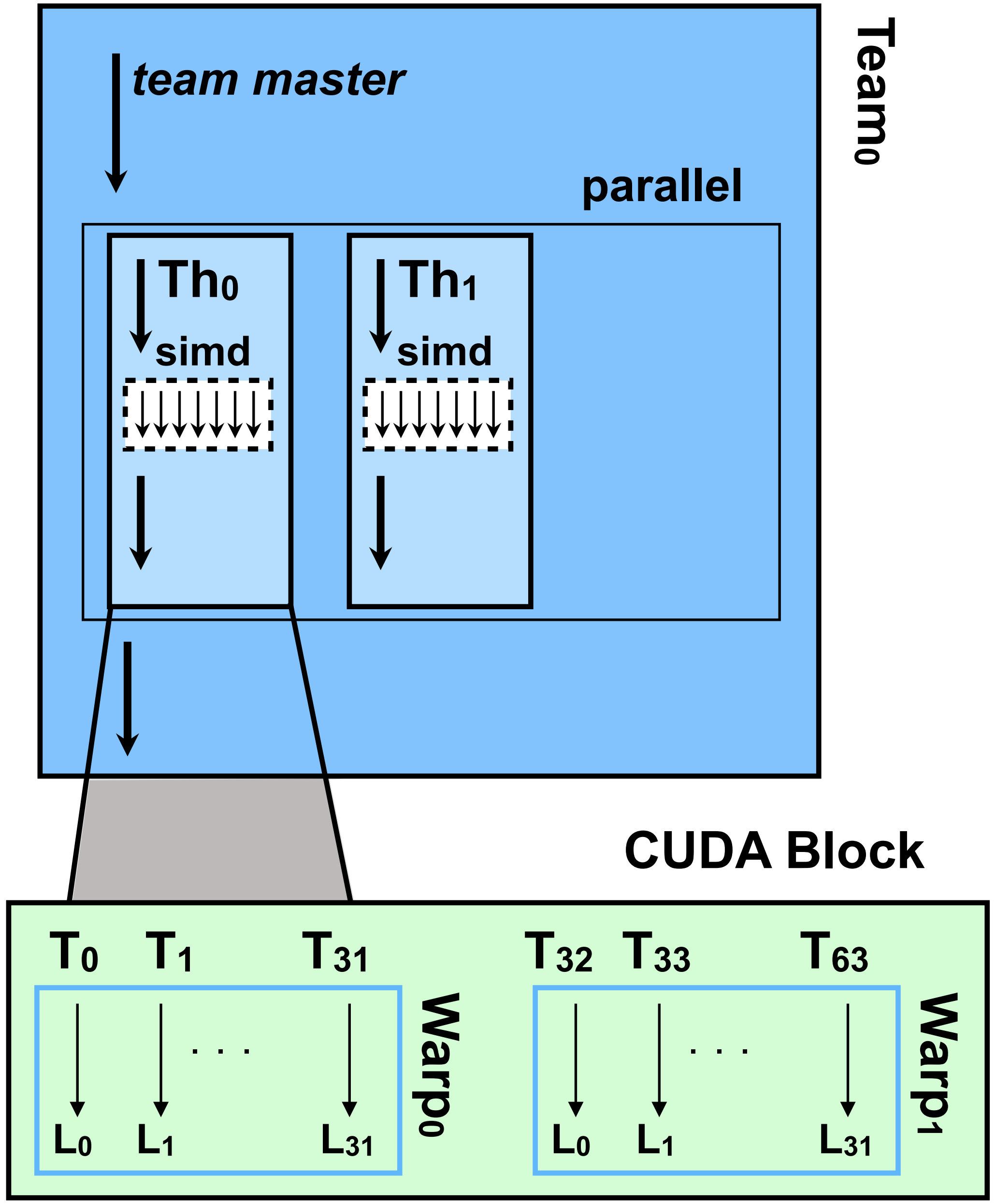




SIMD and other OpenMP forms supported on the GPU

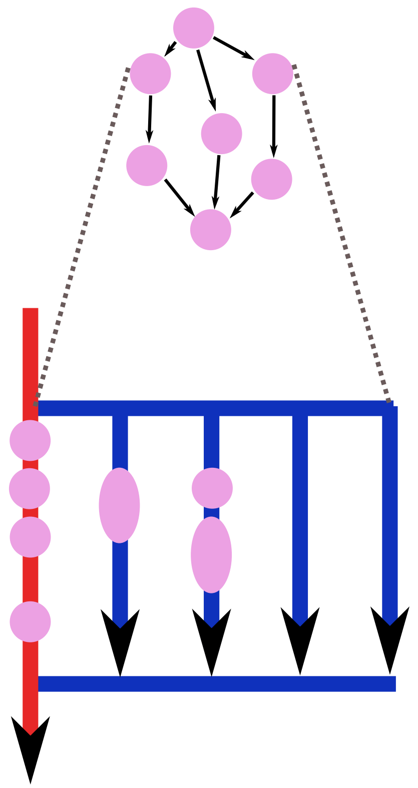
```
#pragma omp target teams distribute
#pragma omp parallel for
for (i = 0; i < M; i++)
    #pragma omp simd
    for (j = 0; j < N; j++)
        A[i][j] += u1[i] * v1[j] +
                   u2[i] * v2[j];
```





Task Parallelism

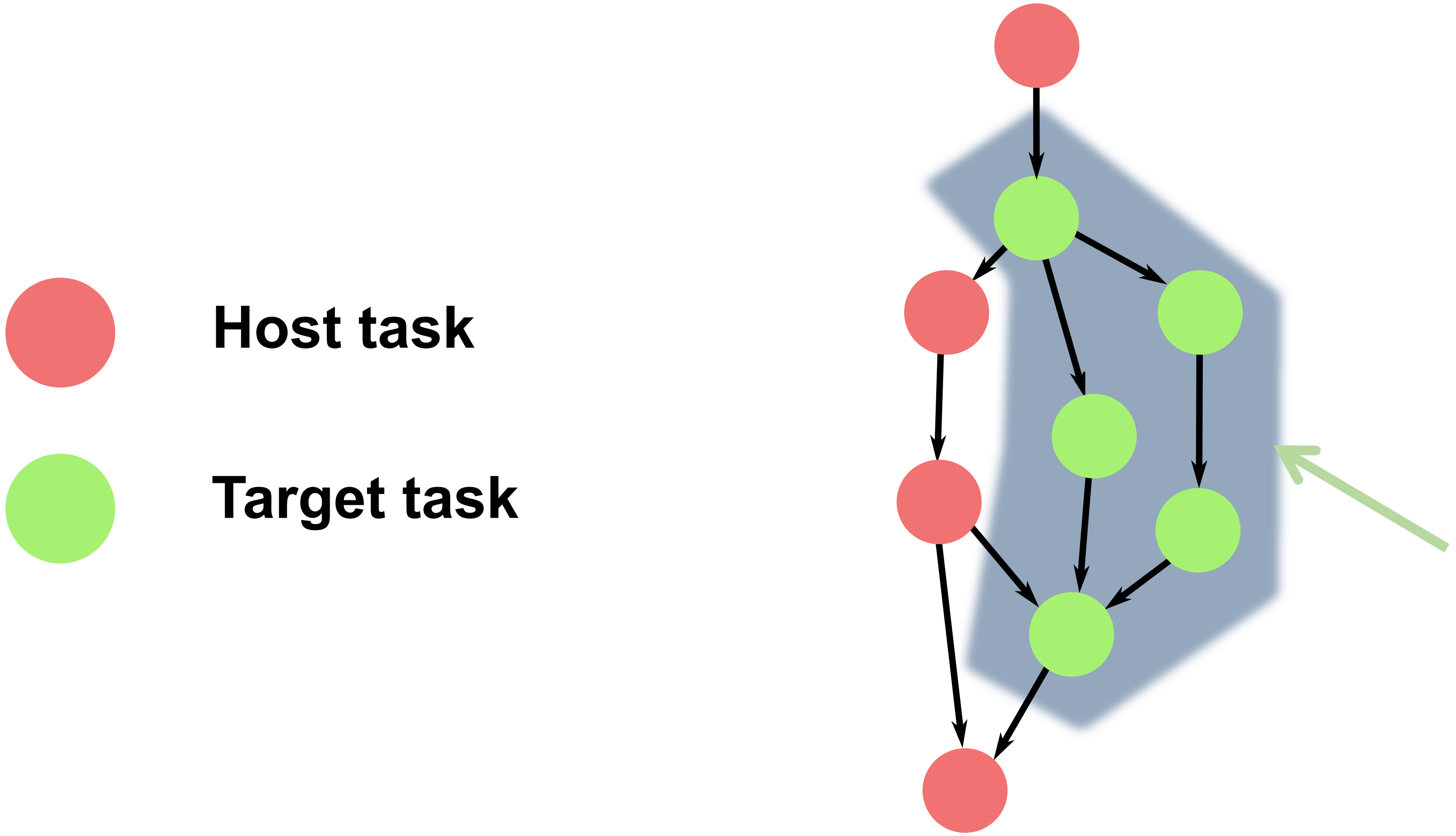
```
#pragma omp parallel
#pragma omp single
{
  #pragma omp task depend(out: a)
  TraverseForward(A);
  #pragma omp task depend(in: a)
  TraverseReverse(B);
  …
}
```



- Tasks are well suited for parallelism that is dynamically uncovered: e.g. searches, graph processing
- Tasks are load balanced between threads in the parallel region
- A task is fired once all its dependent tasks have completed

CPU & GPU Parallelism using Tasks

- Target constructs are implicit tasks
- A host thread may initiate several target tasks asynchronously
- Target tasks may have dependencies

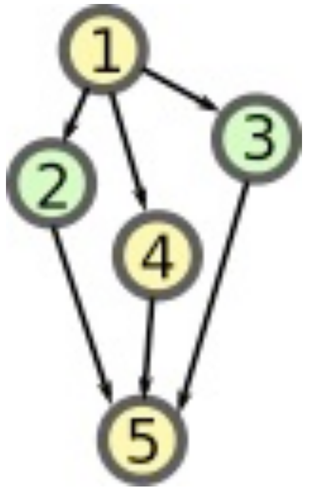


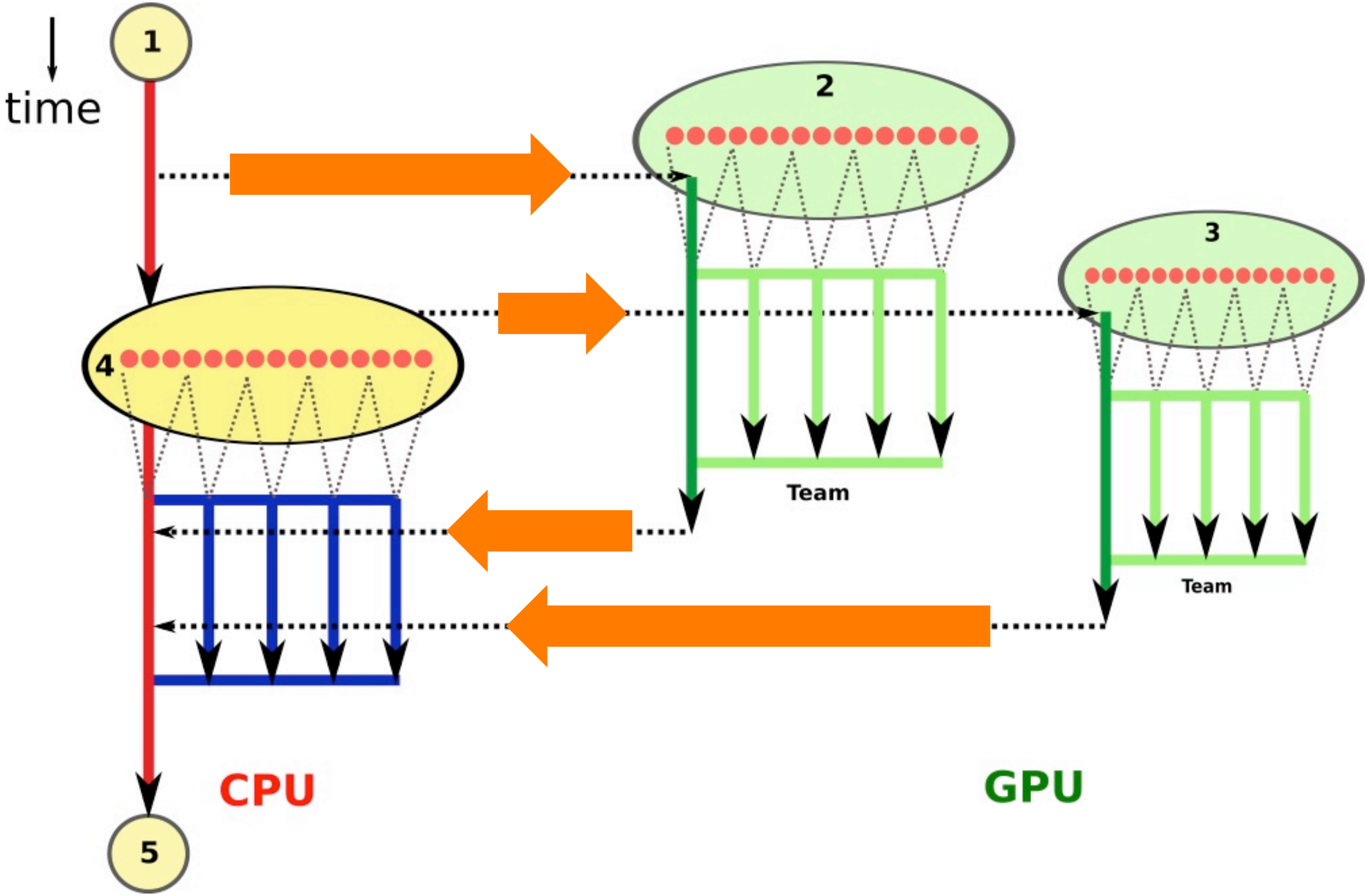
Dependencies between target tasks are resolved completely on the GPU without host intervention

CPU & GPU Parallelism using Tasks

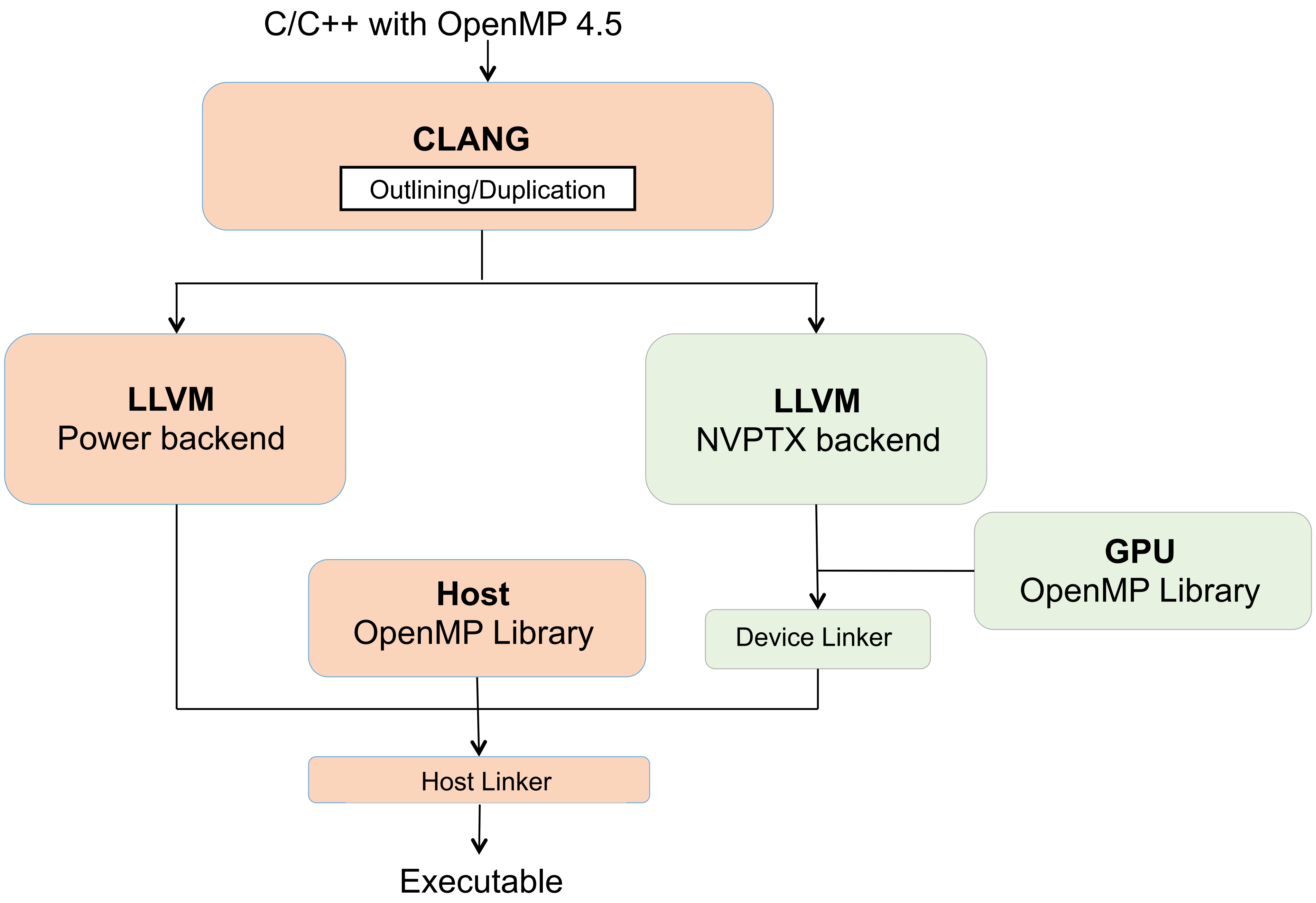
Concurrency in a node

- Host threads and device threads
- Multiple GPUs in a node
- Overlap device computation and communication
- Concurrent target tasks on a GPU with task dependencies





LLVM Compiler Schematic



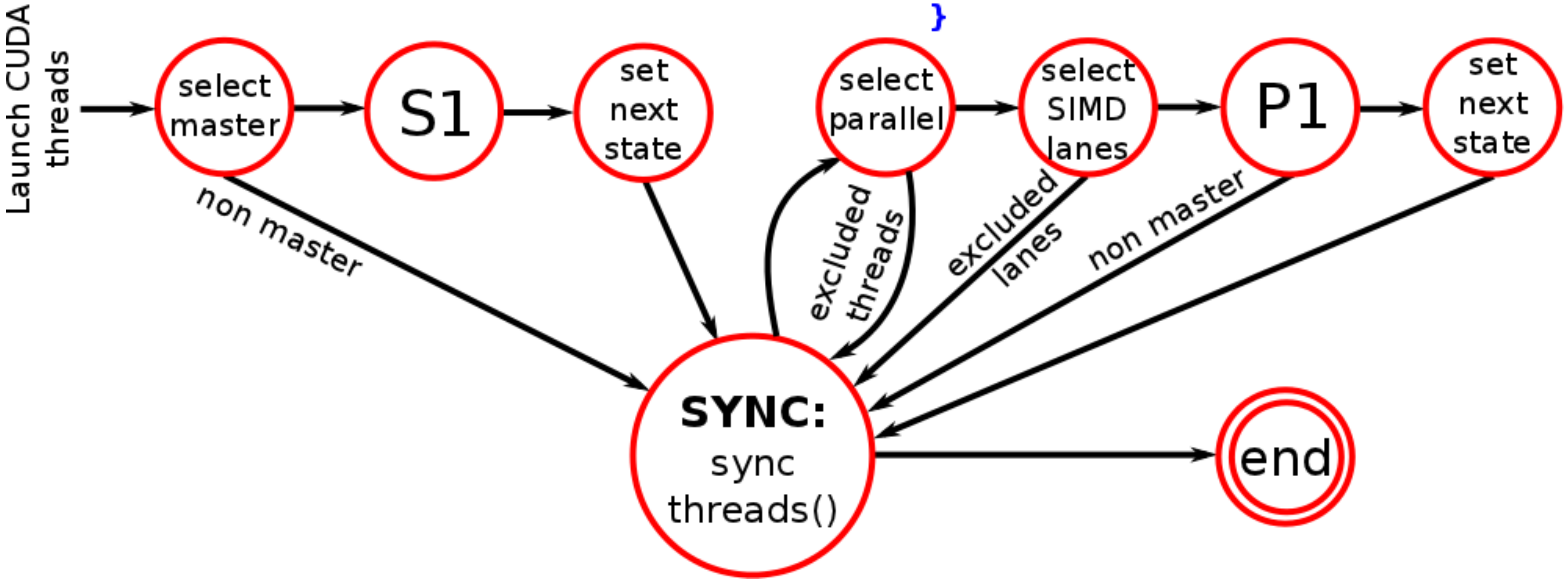
- **CLANG**
  - Front-end to parse source code and generate LLVM IR code
  - Modified to generate code for OpenMP device constructs
  - Produces two copies of code for target regions
  - Inserts calls to standardized OMP runtime interface functions
  - Compiler driver modified to process code copies through different backends
- **NVPTX backend**
  - Produces ptx code which is then processed through *ptxas* to generate CUDA binary

Collaborating with wider community and industry partners (LLVM open-source, OMP standards)

OpenMP Codegen Internals

- Compiler responsible for **thread-activation and thread-coordination**

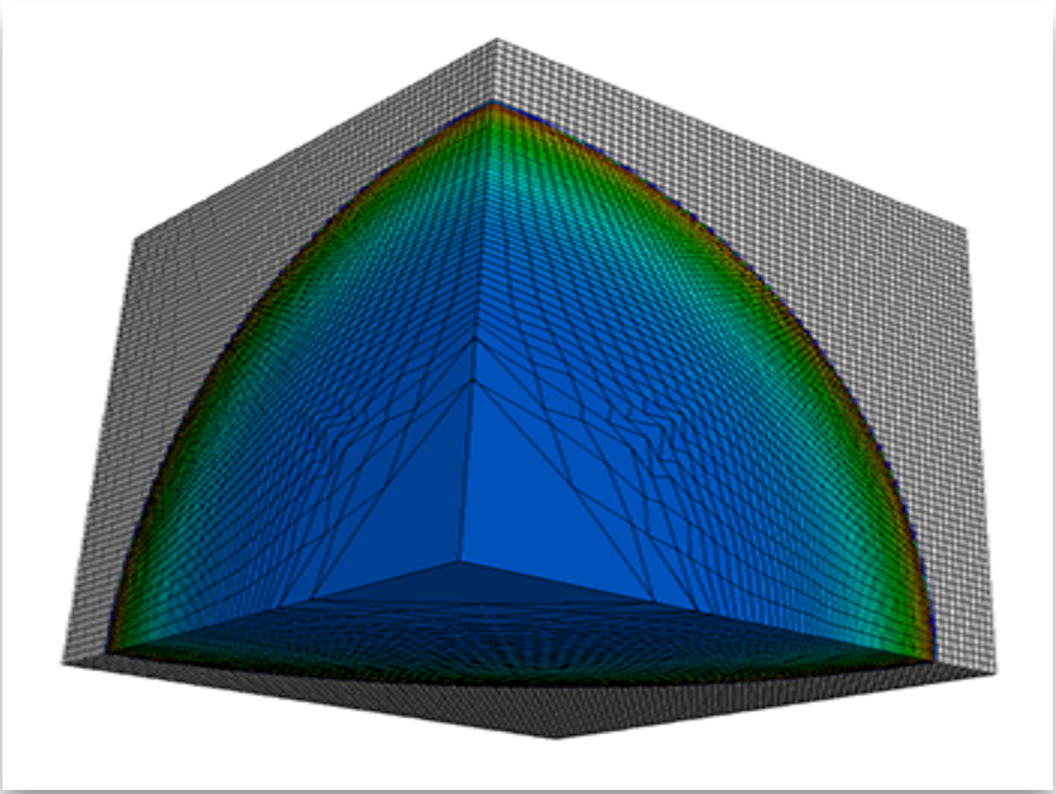
```
    #pragma omp target
    {
S1:     if (w = queue.pop()) {
        #pragma omp parallel num_threads(16)
        {
            #pragma omp simd safelen(8)
P1:         simd_work();
        }
    }
```



Performance Preview

- LULESH: proxy for hydrodynamics code

Kernel*	CUDA (us)	OpenMP 4.0 (us)
Acceleration Calculation	**3.2**	**4.3**
Apply Boundary Acceleration	**5.1**	**4.8**
Position and Velocity Calculation	**3.2**	**4.8**
		4.1
Kinematics and Monotonic Gradient Calculation	**17**	**6.5**
		58
		40
Monotonic Region Calculation	**11**	**15**
Apply Material Properties to Regions	**92**	**102.8**



https://codesign.llnl.gov/lulesh.php

Performance Analysis of OpenMP on a GPU Using a CORAL Proxy Application, Bercea et al. PMBS '15.

- **S6513 - GPU Optimization of the Kripke Neutral-Particle Transport Mini-App, Thursday, 15:30 at Marriott Salon 3**

Compiler Availability and Roadmap

- Opensource: download and installation instructions at:

  ibm.biz/ykt-omp

- Currently supports OpenMP 4.0, with offload to GPU
  - Open source host runtime based on Intel contributed KMPC lib
  - Open source GPU runtime developed and contributed by IBM Research
- Working on upstreaming 4.5 implementation to Clang/LLVM

- Contact: acjacob@us.ibm.com

This work is partially supported by the CORAL project LLNS Subcontract No. B604142.