Integrating GPU Support for OpenMP Offloading Directives into Clang

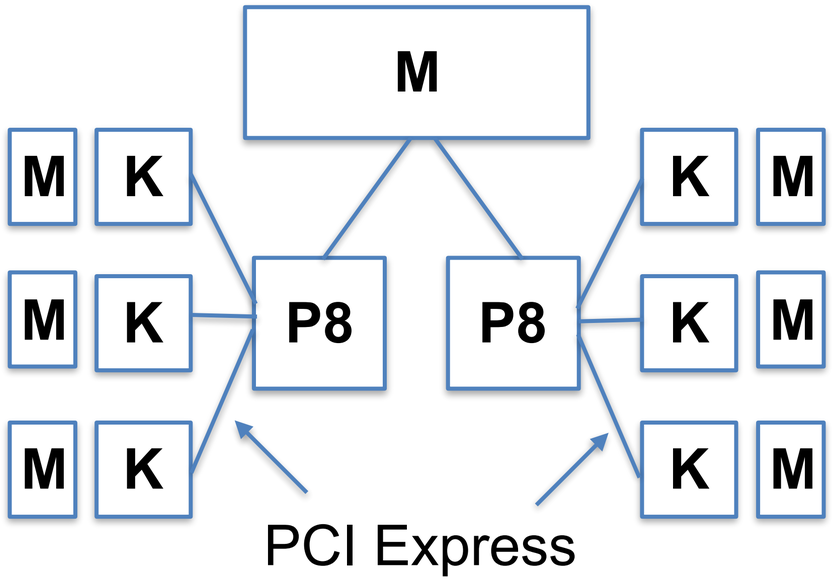
Carlo Bertolli, Samuel F. Antao, Gheorghe-Teodor Bercea,
Arpith C. Jacob, Alexandre E. Eichenberger, Tong Chen,
Zehra Sura, Hyojin Sung, Georgios Rokos,
David Appelhans, Kevin O'Brien

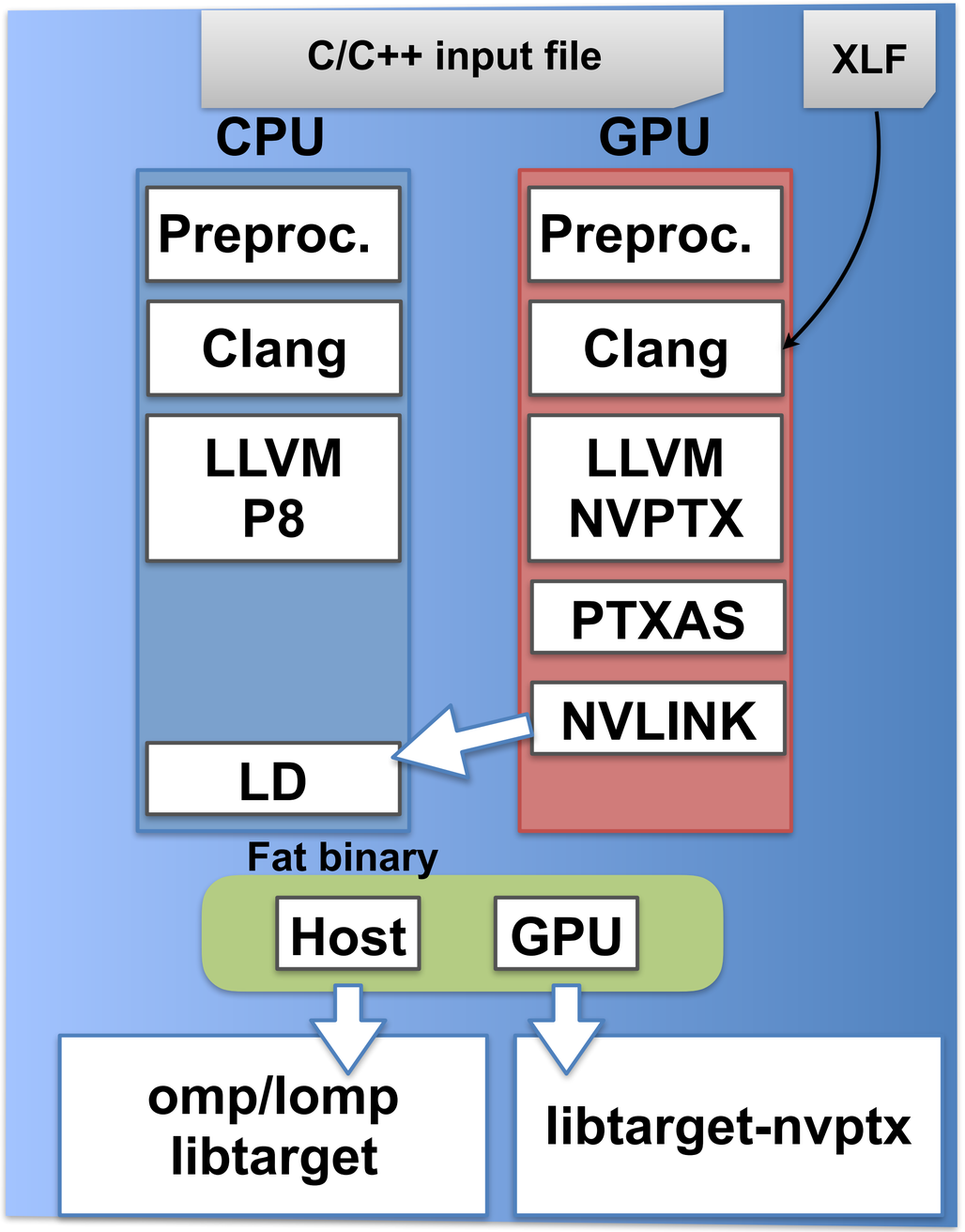
IBM T.J. Watson Research Center

The Second Workshop on the LLVM Compiler Infrastructure in HPC
11.15.15

OpenMP

OpenPOWER™

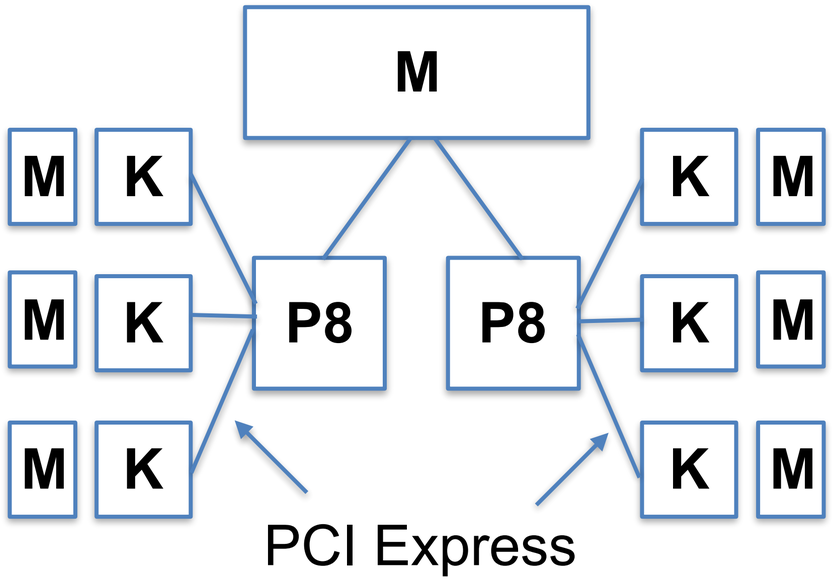


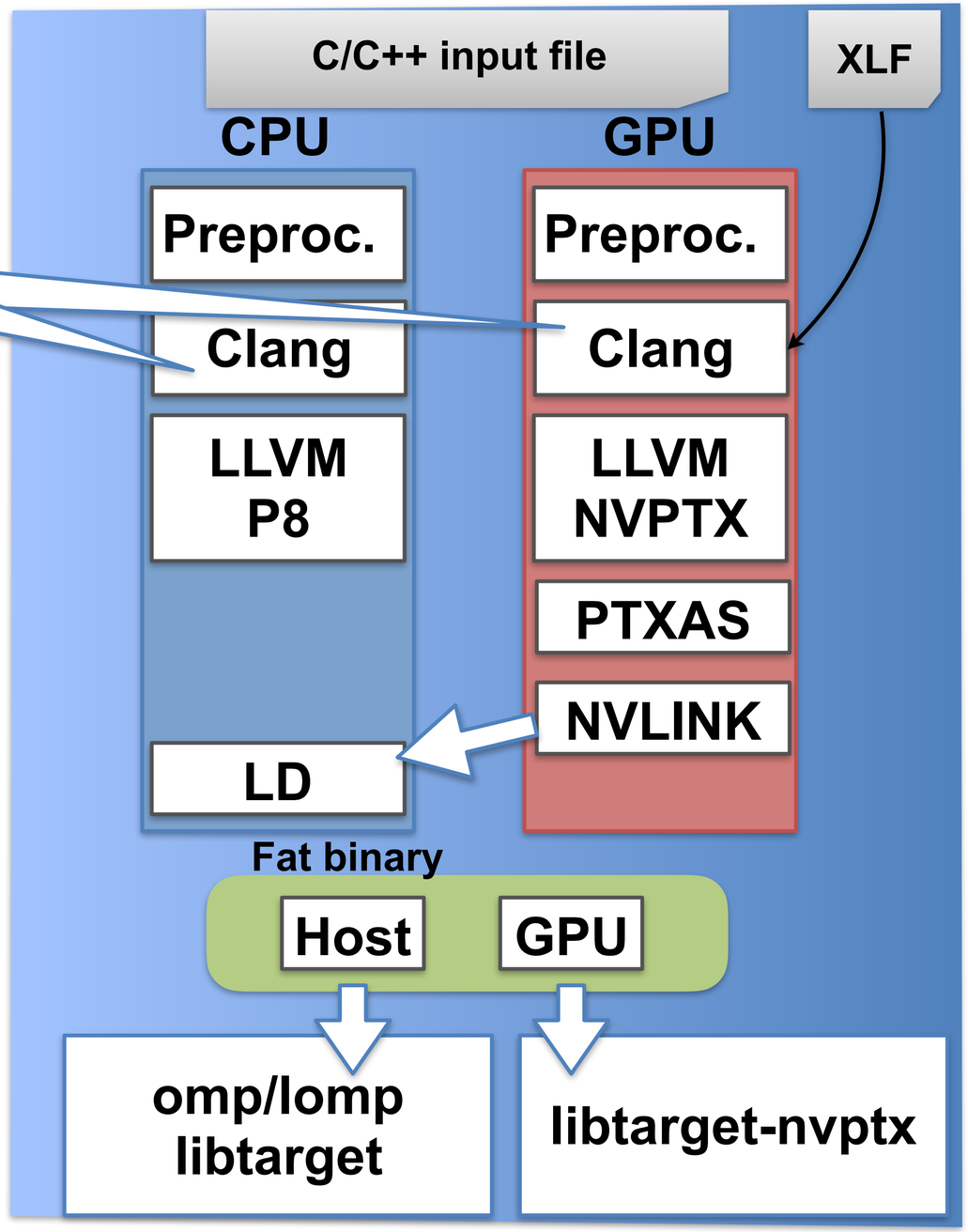


OpenMP

OpenPOWER™

OpenMP Implementation in Clang CG





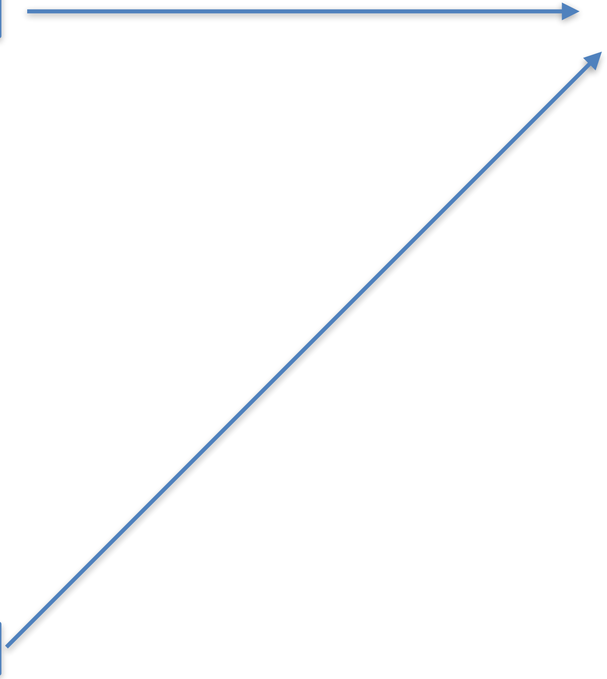
Research Topics

- **Implement OpenMP on GPU**
  - Hard to do for programming constraints
  - Cannot re-use OpenMP on CPU (codegen +lib)
  - Performance guaranteed to be trash in many cases
  - What should be optimized?

- **Integration into Open Source compiler**
  - Cannot be disruptive to compiler design and implementation
  - Based on existing assumptions: OpenMP is implemented in Clang codegen
  - Gather community interest for this implementation to land

OpenMP Challenges for GPUs

```
#pragma omp target teams
{
 if (a[0]++ > 0) {
   #pragma omp parallel for
   for (int i = 0 ; i < n ; i++) {
    if (omp_get_thread_num () > 0) {
      #pragma omp simd
      for (int s = 0 ; s < 32 ; s++) { .. }
    } else {
      #pragma omp simd
      for (int s = 0 ; s < 4 ; s++) { .. }
    }
   }
 } else if(b[0]++ > 0) {
    #pragma omp parallel for
    for (int i = 0 ; i < n*2 ; i++) { .. }
  }
}
```



Sequential within team:
only team master executes this

OpenMP Challenges for GPUs

```
#pragma omp target teams thread_limit(256)
{
  if (a[0]++ > 0) {
    #pragma omp parallel for num_threads(128)
    for (int i = 0 ; i < n ; i++) {
      if (omp_get_thread_num () > 0) {
        #pragma omp simd
        for (int s = 0 ; s < 32 ; s++) { .. }
      } else {
        #pragma omp simd
        for (int s = 0 ; s < 4 ; s++) { .. }
      }
    }
  } else if(b[0]++ > 0) {
    #pragma omp parallel for
    for (int i = 0 ; i < n*2 ; i++) { .. }
  }
}
```

Parallel threads:
some threads are executing this in parallel

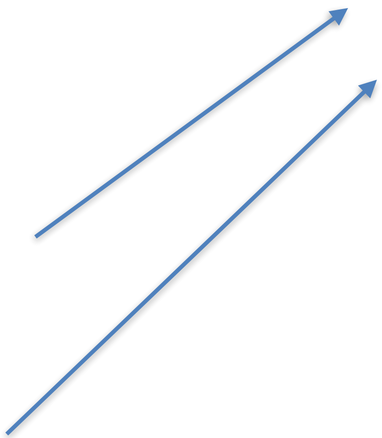
OpenMP Challenges for GPUs

```
#pragma omp target teams
{
  if (a[0]++ > 0) {
    #pragma omp parallel for
    for (int i = 0 ; i < n ; i++) {
      if (omp_get_thread_num () > 0) {
        #pragma omp simd
        for (int s = 0 ; s < 32 ; s++) { .. }
      } else {
        #pragma omp simd
        for (int s = 0 ; s < 4 ; s++) { .. }
      }
    }
  } else if(b[0]++ > 0) {
    #pragma omp parallel for nowait
    for (int i = 0 ; i < n*2 ; i++) { .. }
  }
}
```

Explicit and implicit divergence between threads

OpenMP Challenges for GPUs

```
#pragma omp target teams
{
  if (a[0]++ > 0) {
    #pragma omp parallel for
    for (int i = 0 ; i < n ; i++) {
      if (omp_get_thread_num () > 0) {
        #pragma omp simd
        for (int s = 0 ; s < 32 ; s++) { .. }
      } else {
        #pragma omp simd
        for (int s = 0 ; s < 4 ; s++) { .. }
      }
    }
  } else if(b[0]++ > 0) {
     #pragma omp parallel for nowait
     for (int i = 0 ; i < n*2 ; i++) { .. }
  }
}
```



No actual simd units on GPUs

Control Loop Scheme

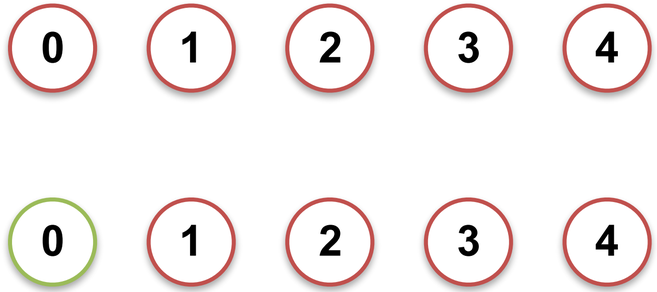
```
int tmp = 3;
#pragma omp target teams \
  thread_limit(5) \
  map(tofrom:tmp,a[:n])
{
   tmp += 3;
   #pragma omp parallel for \
     num_threads(4)
   for (int i = 0 ; i < n; i++)
     a[i] += tmp;
   tmp = -1;
}
```

Avoid dynamic parallelism and start all threads

```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;
      // sequential reg. 1
      nextState = PR1;
      break;
    case PR1:
      if(tid > 4) break;
      // parallel reg. 1
      if (tid == 0) nextState = SQ2;
      break;
    case SQ2:
      if(tid > 0) break;
      // sequential reg. 2
      finished = true;
      break;
  }
  __syncthreads();
}
```

Control Loop Scheme

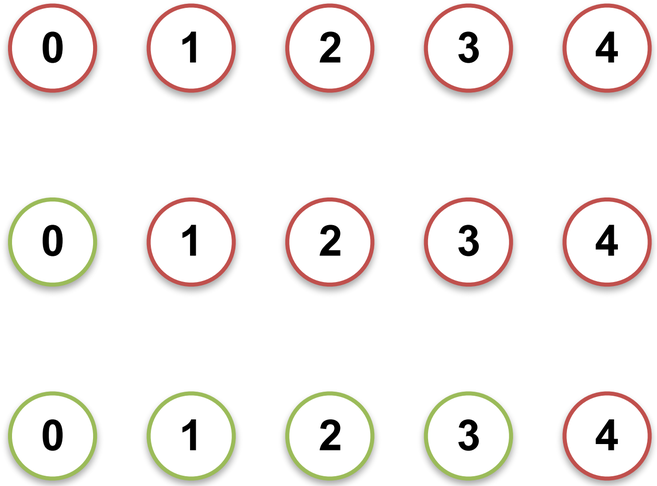
```
int tmp = 3;
#pragma omp target teams \
  thread_limit(5) \
  map(tofrom:tmp,a[:n])
{
   tmp += 3;
   #pragma omp parallel for \
     num_threads(4)
   for (int i = 0 ; i < n; i++)
     a[i] += tmp;
   tmp = -1;
}
```




```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;
      // sequential reg. 1
      nextState = PR1;
      break;
    case PR1:
      if(tid > 4) break;
      // parallel reg. 1
      if (tid == 0) nextState = SQ2;
      break;
    case SQ2:
      if(tid > 0) break;
      // sequential reg. 2
      finished = true;
      break;
  }
  __syncthreads();
}
```

Control Loop Scheme

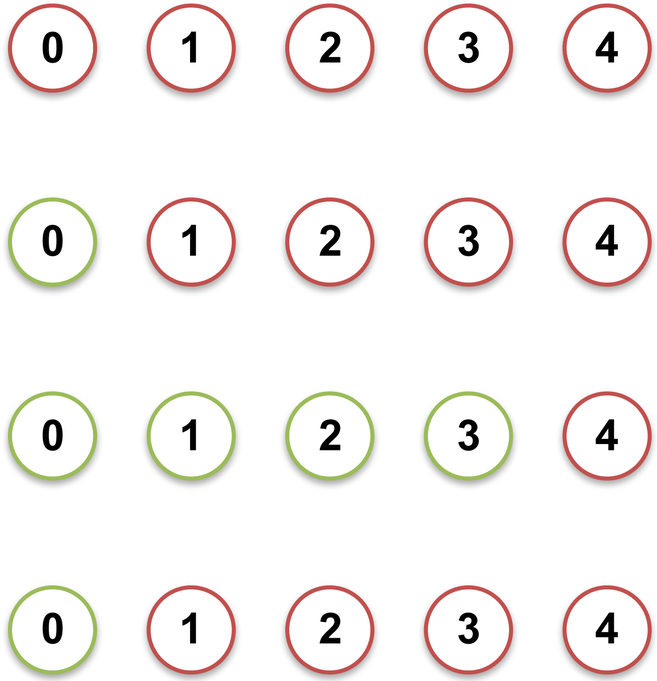
```
int tmp = 3;
#pragma omp target teams \
  thread_limit(5) \
  map(tofrom:tmp,a[:n])
{
   tmp += 3;
   #pragma omp parallel for \
     num_threads(4)
   for (int i = 0 ; i < n; i++)
     a[i] += tmp;
   tmp = -1;
}
```




```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;
      // sequential reg. 1
      nextState = PR1;
      break;
    case PR1:
      if(tid > 3) break;
      // parallel reg. 1
      if (tid == 0) nextState = SQ2;
      break;
    case SQ2:
      if(tid > 0) break;
      // sequential reg. 2
      finished = true;
      break;
  }
  __syncthreads();
}
```

Control Loop Scheme

```
int tmp = 3;
#pragma omp target teams \
  thread_limit(5) \
  map(tofrom:tmp,a[:n])
{
   tmp += 3;
   #pragma omp parallel for \
     num_threads(4)
   for (int i = 0 ; i < n; i++)
     a[i] += tmp;
   tmp = -1;
}
```




```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;
      // sequential reg. 1
      nextState = PR1;
      break;
    case PR1:
      if(tid > 4) break;
      // parallel reg. 1
      if (tid == 0) nextState = SQ2;
      break;
    case SQ2:
      if(tid > 0) break;
      // sequential reg. 2
      finished = true;
      break;
  }
  __syncthreads();
}
```

Control Loop & Clang

- Rules for modular integration
- **Do**'s
  - Extend OpenMP-related functions
  - Add new function calls
  - Add new runtime functions only for specific targets

- **Don't**s
  - OpenMP target implementation influences every C/C++ construct
  - Add metadata and process OpenMP later when more convenient

Example: Codegen Control Loop for **#target**

```
void CGF::EmitOMPTargetDirective(..) {
  // control flow will lead to…

  if (isTargetMode)
    CGM.getOpenMPRuntime().EnterTargetLoop();

  // emit target region statements
  CGF.EmitStmt(CS->getCapturedStmt());

  if (isTargetMode)
    CGM.getOpenMPRuntime().ExitTargetLoop();
}
```

codegen

```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;


  }
   __syncthreads();
}
```

Example: Codegen Control Loop for **#target**

```
void CGF::EmitOMPTargetDirective(..) {
  // control flow will lead to…

  if (isTargetMode)
    CGM.getOpenMPRuntime().EnterTargetLoop();

  // emit target region statements
  CGF.EmitStmt(CS->getCapturedStmt());

  if (isTargetMode)
    CGM.getOpenMPRuntime().ExitTargetLoop();
}
```

setInsertPoint

```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;


    }
    __syncthreads();
}
```

Example: Codegen Control Loop for **#target**

```
void CGF::EmitOMPTargetDirective(..) {
  // control flow will lead to…

  if (isTargetMode)
    CGM.getOpenMPRuntime().EnterTargetLoop();

  // emit target region statements
  CGF.EmitStmt(CS->getCapturedStmt());

  if (isTargetMode)
    CGM.getOpenMPRuntime().ExitTargetLoop();
}
```

codegen until **#parallel**

```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;


    }
    __syncthreads();
}
```

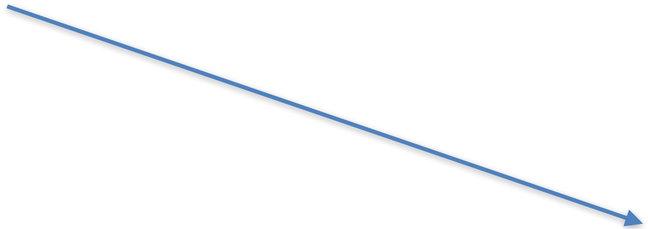
Example: Codegen Control Loop for #**parallel**

```
void CGF::EmitOMPParallelDirective(..) {
  // control flow will lead to…

  if (isTargetMode)
    CGM.getOpenMPRuntime().EnterParallel();

  // emit parallel region statements
  CGF.EmitStmt(CS->getCapturedStmt());

  if (isTargetMode)
    CGM.getOpenMPRuntime().ExitParallel();
}
```



```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;
       // sequential reg. 1
       nextState = PR1;
       break;
     case PR1:
       if(tid > num_threads) break;

  }
   __syncthreads();
}
```

Example: Codegen Control Loop for **#target**

```
void CGF::EmitOMPParallelDirective(..) {
  // control flow will lead to…

  if (isTargetMode)
    CGM.getOpenMPRuntime().EnterParallel();

  // emit parallel region statements
  CGF.EmitStmt(CS->getCapturedStmt());

  if (isTargetMode)
    CGM.getOpenMPRuntime().ExitParallel();
}
```

setInsertPoint

```
nextState = SQ1;
while(!finished) {
  switch(nextState) {
    case SQ1:
      if(tid > 0) break;
       // sequential reg. 1
       nextState = PR1;
       break;
     case PR1:
       if(tid > num_threads) break;


  }
   __syncthreads();
}
```

Control Loop Overhead vs CUDA (1/2)

```
#pragma omp target teams \
  distribute parallel for \
  schedule(static,1)
for (i = 0 ; i < n ; i++)
  a[i] += b[i] + c[i];
```

```
for (int i = threadIdx.x + blockIdx.x *
blockDim.x;
  i < n;
  i += blockDim.x * gridDim.x)
  a[i] += b[i] + c[i];
```

Nvidia Tesla K40m
-maxregcount=64

Vector Add	CUDA	Control Loop
#registers/thread	16	**64**
Shared Memory (bytes)	0	280
Occupancy	95.9%	26.6%
Execution Time (usec.)	1523.5	1988.5

Control Loop Overhead vs CUDA (2/2)

```
#pragma omp target teams \
  distribute parallel for \
  schedule(static,1)
for (i = 0 ; i < n ; i++)
  for (j = 0 ; j < n_loop ; j++)
    a[i] += b[i] + c[i*n_loop + j];
```

```
for (int i = threadIdx.x + blockIdx.x *
blockDim.x;
  i < n;
  i += blockDim.x * gridDim.x)
  for (j = 0 ; j < n_loop ; j++)
    a[i] += b[i] + c[i*n_loop + j];
```

Nvidia Tesla K40m
-maxregcount=64

Vector Matrix Add	CUDA	Control Loop
#registers/thread	18	**64**
Shared Memory (bytes)	0	280
Occupancy	97.3%	49.5%
Execution Time (usec.)	70832.0	78333.0

Occupancy / Register Allocation

- Many reasons:
  - A while loop with a switch inside hits hard register allocation
  - In OpenMP 4.0 kernel parameters are passed as pointer to pointer
    - The kernel is allowed to do pointer arithmetic
    - This provokes an additional register for each parameter
    - Fixed by OpenMP 4.5: always pass pointer, what matters is the pointee address
  - CUDA and LLVM backends for NVPTX are different:
    - CUDA uses libnvvm, which is shipped as a library
    - LLVM uses the open source code in the trunk
    - Different optimization strategies

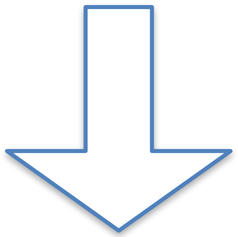
Optimizing “Good Cases”: LULESH

- Recurrent pragma patterns to be optimized
- Some hints
  - No OpenMP control flow divergence
  - No nested parallelism/pragmas
  - No hard stuff: locks, tasks, etc..

```
#pragma omp parallel for firstprivate(numNode)
  for( Index_t gnode=0 ; gnode<numNode ; ++gnode )
  {
    Index_t count = domain.nodeElemCount(gnode) ;
    Index_t *cornerList = domain.nodeElemCornerList(gnode) ;
    Real_t fx_tmp = Real_t(0.0) ;
    Real_t fy_tmp = Real_t(0.0) ;
    Real_t fz_tmp = Real_t(0.0) ;
    for (Index_t i=0 ; i < count ; ++i) {
      Index_t elem = cornerList[i] ;
      fx_tmp += fx_elem[elem] ;
      fy_tmp += fy_elem[elem] ;
      fz_tmp += fz_elem[elem] ;
    }
    domain.fx(gnode) = fx_tmp ;
    domain.fy(gnode) = fy_tmp ;
    domain.fz(gnode) = fz_tmp ;
  }
```

Porting LULESH to OpenMP 4

```
#pragma omp parallel for firstprivate(numNode)
for( Index_t gnode=0 ; gnode<numNode ; ++gnode )
{
}
```



OpenMP 4-ization

```
#pragma omp target teams distribute parallel for schedule(static,1) \
  firstprivate(numNode)
for( Index_t gnode=0 ; gnode<numNode ; ++gnode )
{
}
```

Implementation of **Combined** Construct

```
#pragma omp target teams distribute parallel for schedule(static,1) \
  firstprivate(numNode)
for( Index_t gnode=0 ; gnode<numNode ; ++gnode )
{
}
```

CUDA-style notation

```
for (int i = threadIdx.x + blockIdx.x * blockDim.x;
  i < n;
  i += blockDim.x * gridDim.x) {
  g_node = i;

  // codegen loop body
}
```

Compiler:

- Detect pragma combination
- Prove absence of nested pragmas

Combined Directive - Vector Add

Vector add	CUDA	Control Loop	Combined
#registers/thread	16	64	**21**
Shared Memory (bytes)	0	280	0
Occupancy	95.9%	26.6%	96%
Execution Time (usec.)	1523.5	1988.5	1523.1

Control Loop Overhead vs CUDA (2/2)

Vector-Matrix Add	CUDA	Control Loop	Combined
#registers/thread	18	**64**	**30**
Shared Memory (bytes)	0	280	0
Occupancy	97.3%	49.5%	97.7%
Execution Time (usec.)	70832.0	78333.0	70456.0

LULESH Kernels

Mesh Size		ApplyAccelBCForNode			CalcMonotonicQRegionForElems		
		CUDA	Control Loop	Combined	CUDA	Control Loop	Combined
	#registers	6	64	22	32	64	64
	Shared Memory	0	280	0	0	280	0
12^3	Occupancy	5.5%	6.7%	43.8%	5.9%	6.6%	5.6%
	Execution Time (µsec)	5.184	20.928	5.024	11.169	62.751	15.775
30^3	Occupancy	6.1%	3.3%	14.7%	70.6%	26.5%	43.3%
	Execution Time (µsec)	5.568	22.912	4.96	29.184	178.18	67.296
100^3	Occupancy	27.8%	13.2%	33.8%	93.0%	26.4%	47.3%
	Execution Time (µsec)	6.72	50.944	14.976	1287.4	4833.2	2563.4

LULESH Kernels

Mesh Size		ApplyAccelBCForNode			CalcMonotonicQRegionForElems		
		CUDA	Control Loop	Combined	CUDA	Control Loop	Combined
	#registers	6	64	22	32	64	64
	Shared Memory	0	280	0	0	280	0
12^3	Occupancy	5.5%	6.7%	43.8%	5.9%	6.6%	5.6%
	Execution Time (µsec)	5.184	20.928	5.024	11.169	62.751	15.775
30^3	Occupancy	6.1%	3.3%	14.7%	70.6%	26.5%	43.3%
	Execution Time (µsec)	5.568	22.912	4.96	29.184	178.18	67.296
100^3	Occupancy	27.8%	13.2%	33.8%	93.0%	26.4%	47.3%
	Execution Time (µsec)	6.72	50.944	14.976	1287.4	4833.2	2563.4

LULESH Kernels

Mesh Size		ApplyAccelBCForNode			CalcMonotonicQRegionForElems		
		CUDA	Control Loop	Combined	CUDA	Control Loop	Combined
	#registers	6	64	22	32	64	64
	Shared Memory	0	280	0	0	280	0
12^3	Occupancy	5.5%	6.7%	43.8%	5.9%	6.6%	5.6%
	Execution Time (µsec)	5.184	20.928	5.024	11.169	62.751	15.775
30^3	Occupancy	6.1%	3.3%	14.7%	70.6%	26.5%	43.3%
	Execution Time (µsec)	5.568	22.912	4.96	29.184	178.18	67.296
100^3	Occupancy	27.8%	13.2%	33.8%	93.0%	26.4%	47.3%
	Execution Time (µsec)	6.72	50.944	14.976	1287.4	4833.2	2563.4

LULESH Kernels

Mesh Size		ApplyAccelBCForNode			CalcMonotonicQRegionForElems		
		CUDA	Control Loop	Combined	CUDA	Control Loop	Combined
	#registers	6	64	22	32	64	64
	Shared Memory	0	280	0	0	280	0
12^3	Occupancy	5.5%	6.7%	43.8%	5.9%	6.6%	5.6%
	Execution Time (µsec)	5.184	20.928	5.024	11.169	62.751	15.775
30^3	Occupancy	6.1%	3.3%	14.7%	70.6%	26.5%	43.3%
	Execution Time (µsec)	5.568	22.912	4.96	29.184	178.18	67.296
100^3	Occupancy	27.8%	13.2%	33.8%	93.0%	26.4%	47.3%
	Execution Time (µsec)	6.72	50.944	14.976	1287.4	4833.2	2563.4

LULESH Kernels

Mesh Size		ApplyAccelBCForNode			CalcMonotonicQRegionForElems		
		CUDA	Control Loop	Combined	CUDA	Control Loop	Combined
	#registers	6	64	22	32	64	64
	Shared Memory	0	280	0	0	280	0
12^3	Occupancy	5.5%	6.7%	43.8%	5.9%	6.6%	5.6%
	Execution Time (µsec)	5.184	20.928	5.024	11.169	62.751	15.775
30^3	Occupancy	6.1%	3.3%	14.7%	70.6%	26.5%	43.3%
	Execution Time (µsec)	5.568	22.912	4.96	29.184	178.18	67.296
100^3	Occupancy	27.8%	13.2%	33.8%	93.0%	26.4%	47.3%
	Execution Time (µsec)	6.72	50.944	14.976	1287.4	4833.2	2563.4

Conclusion

- Generality of OpenMP has a large performance cost
  - even in simpler cases (no tasks, no locks, etc..)

- Optimized schemes are possible
  - More schemes in the near future
  - Interesting cases require control-flow analysis in Clang

- Optimize register allocation for control loop
  - Better low level optimizations in NVPTX
  - Optimize control loop scheme