www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Exploring Dynamic Parallelism on OpenMP

Guray Ozen, Eduard Ayguadé, Jesús Labarta

WACCPD @ SC'15

Austin, Texas 2015

MACC: Introduction

MACC = **M**ercurium **AC**celerator **C**ompiler [1]

- **OpenMP Accelerator + OmpSs task model**
- Trying to influence the evolution of the OpenMP
- Extended OpenMP with experimental clauses
  - Three new clauses for `distribute` construct
    - Using team memory(shared) chunk by chunk without synchronization teams
    - `dist_private([CHUNK]data, …) ,dist_firstprivate([CHUNK]data, …), dist_lastprivate([CHUNK]data, …)`
- Using together Task & Target directives
  - Programmer only specifies directionality of task data, not the actual data movement
    - `#pragma omp task in(list) out(list) inout(list)`
  - Doesn't download data from GPU until `#pragma omp taskwait`
  - **Data transfer minimization** (host-2-gpu)
  - Automatically **Multi-GPU task scheduling**
  - **Ignored** target data & target update

[1] On the Roles of the Programmer, the Compiler and the Runtime System When Programming Accelerators in OpenMP. IWOMP 2014, Guray Ozen, Eduard Ayguadé, Jesús Labarta

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

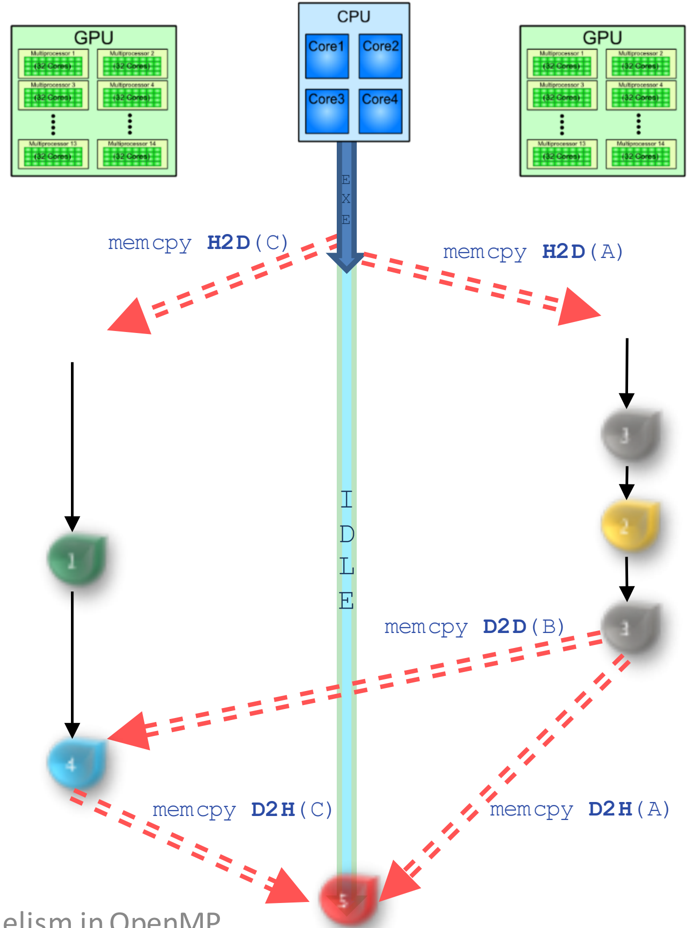
MACC: Introduction

```
int main(…) {
    double A[N], B[N], C[N] , D[N];
    while (0-> 2)
    {
      #pragma omp target device(acc)
      #pragma omp task inout(C) out(D)
      #pragma omp teams distribute parallel for
        <.. Sequential Codes to generate CUDA..>

      #pragma omp target device(acc)
      #pragma omp task in(A) out(B)
      #pragma omp teams distribute parallel for
       for(i=0 ; i< N; ++i)
        <..Sequential Codes to generate CUDA..>

      #pragma omp target device(acc)
      #pragma omp task inout(A,B)
      #pragma omp teams distribute parallel for
       for(i=0 ; i< N; ++i)
        <..Sequential Codes to generate CUDA..>
    }
    #pragma omp target device(acc)
    #pragma omp task inout(C,B) in(D)
    #pragma omp teams distribute parallel for
     for(i=0 ; i< N; ++i)
      <..Sequential Codes to generate CUDA..>

    #pragma omp target device(smp)
    #pragma omp task in(A, C)
      <..Sequential codes / Result Test..>
    #pragma omp taskwait
}
```



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Dynamic Parallelism (DP)

What's DP

- Autonomously launching kernel without CPU-host intervention
- CUDA thread(parent) launches kernel(child)

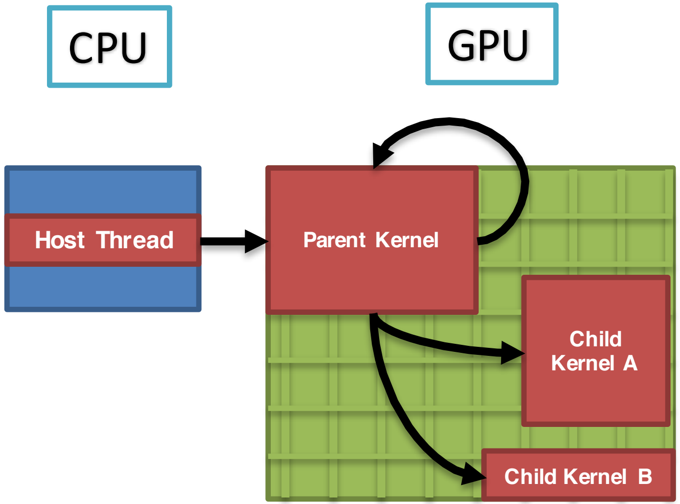
- **What can we do with DP ?**
  - Improved programmability
  - Dynamic load balancing
  - Increase occupancy
  - Ability to implement recursive algorithms

- **Issues with DP**
  - Device kernel launch might incur overhead
  - Shared is not visible between kernels
  - Global memory is visible between kernels
    - Weak consistent
    - With synchronization, can be strong

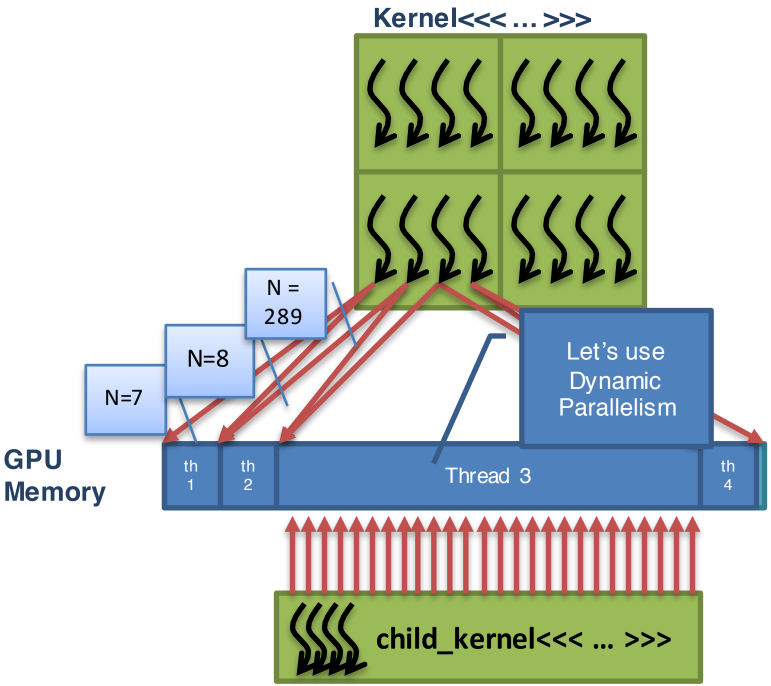
Syntax of Dynamic Parallelism (DP)

```
void main()
{
    parent_kernel<<<…>>>();
}
void __global__ parent_kernel()
{
    child_kernel_A<<<…>>>();
    child_kernel_B<<<…>>>();
    parent_kernel <<<…>>>();
}

void __global__ child_kernel_A()
{
    <.. Codes ..>
}
void __global__ child_kernel_B()
{
    <.. Codes ..>
}
```



Motivation




```
__global__ kernel(double* A, double* X, int* iter, int N, double alpha)
{
    int i = threadIdx.x + blockDim.x * gridDim.x;
    <.. Code ..>
    N = iter[i] - iter[i+1];

    for(j = iter[i]; j < iter[i+1]; j++)
      X[j] += A[j] * alpha;

    if(N > 256)
      child_kernel<<<1,N>>>(A, X, alpha);
    else
    <.. Code ..>
      for (j = iter[i]; j < iter[i+1]; j++)
        X[j] += A[j] * alpha;
}
    <.. Code ..>
}
```

Workload dependency

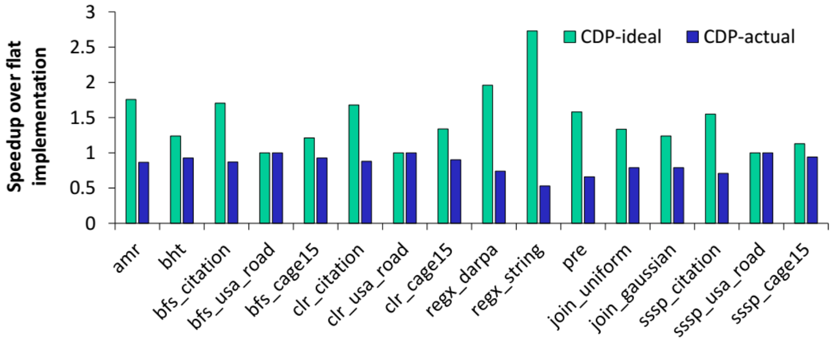
Also not coalesced

```
__global__ child_kernel(double* A, double* X, int* iter, int N, double alpha)
{
    int j = threadIdx.x + blockDim.x * gridDim.x;

    X[j] += A[j] * alpha;
}
```

UNMotivation

- DP can increase performance
- Kernel launching overhead could negate performance benefit



Overall Performance: Speedup over flat implementations

- CDP-ideal: excluding launch overhead, average 1.47x speedup
- CDP-actual: including launch overhead, average 1.21x slowdown

*IISWC'14, Characterization and Analysis of Dynamic Parallelism in Unstructured GPU Applications, Jin Wang, Sudhakar Yalamanchili

Dynamic Parallelism in Pragmatic Programming Models

- **OpenMP 4.x**
  - There are no directives which express dynamic parallelism
  - Using `teams/distribute` construct is prohibited

- **OpenACC 2.0**
  - Standard supports
  - As we know, only ENZO compiler involves its implementation

Proposal of Dynamic Parallelism for OpenMP

- We defined a semantic to **nested teams** constructs
  - To express explicitly DP
  - The teams construct already has definition clauses for number of teams/threads

- Inner device constructs (`distribute`, `parallel for` etc.) bind to **closest teams**
  - Allows inner teams' code scope to be specified parallelism level

- We propose **if** clause to conditionally activate nested teams construct
  - DP might incur overhead, conditional usage can prevent

Proposal of Dynamic Parallelism for OpenMP

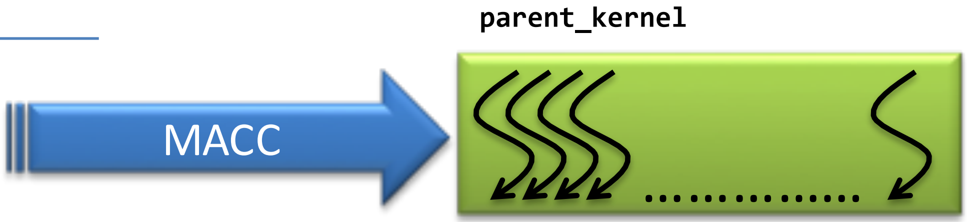
- The **if** clause can avoid redundant DP invocations
  - Disables pragma directive. Compiler creates if-then-else in the parent kernel
  - In this case, inner code block is executed sequentially

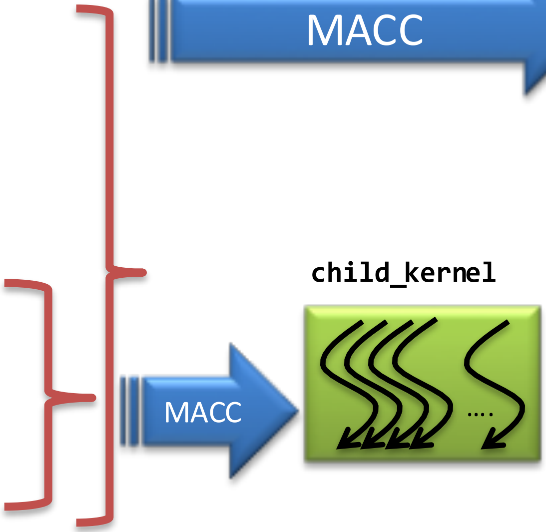
```
#define THRESHOLD 256

#pragma omp target map(…)
#pragma omp teams distribute parallel for
for (i = 0; i < N; ++i)
{
  <..Computation Code..>

  int M =  <..Compute M value..>

  #pragma omp teams distribute parallel for
                   nowait if( M>THRESHOLD )
  for (j = 0; j < M; ++j)
     for (k = 0; k < N; ++k)
     {
         <..Computation Code..>
     }
}
```






```
__global__ parent_kernel(…){
  <.Parallelized Computation Code.>

  int M =  <..Compute M value..>

  if ( M > THRESHOLD )
     child_kernel<<<…>>>(…);
  else
    for (j = 0; j < M; ++j)
     for (k = 0; k < N; ++k)
      <..Computation Code..>
}
```

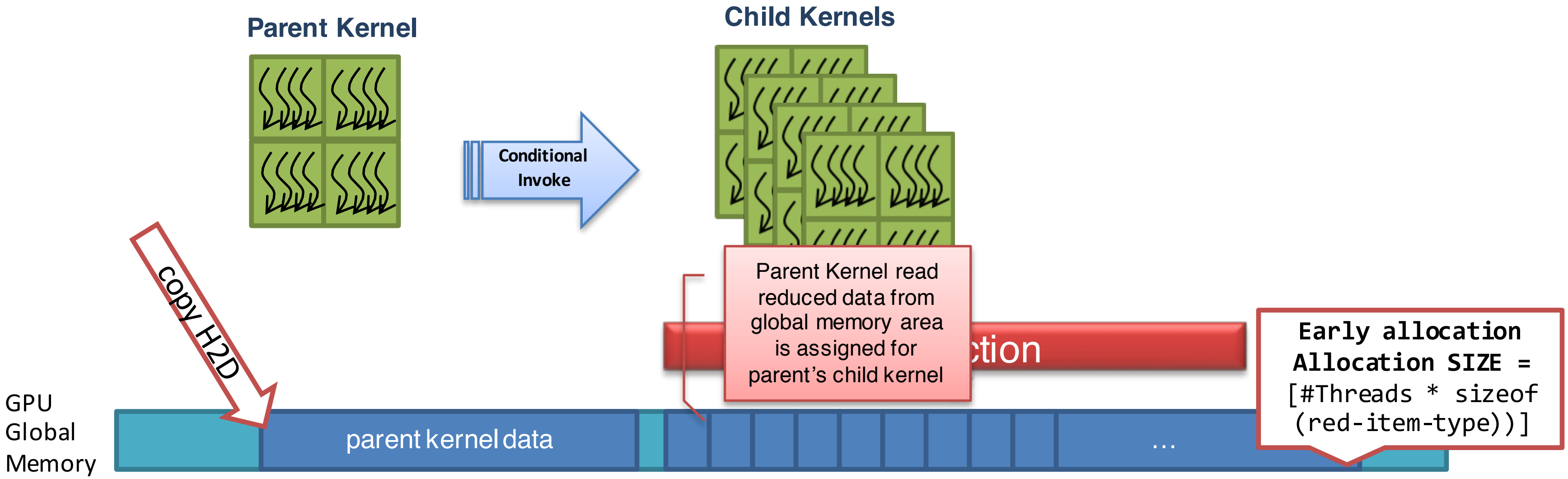
Proposal of Dynamic Parallelism for OpenMP

- **Issue** child-2-parent kernels can communicate only via global memory
  - When it's necessary, spare space from global memory is needed
  - However doing allocation in the runtime incurs overheads

- **Solution:**
  - We've implemented early memory allocation mechanism
  - Global memory for communication is allocated by host in advance
  - It's also used when `reduction` is used with inner `teams`
  - It allocates memory for every single parent thread even if not used

Proposal of Dynamic Parallelism for OpenMP

How reduction works for inner teams



Proposal of Dynamic Parallelism for OpenMP

- Preliminary support for **recursion**
  - Max nesting depth is 24 for devices compute capability 3.5
  - Three functions are generated

```
#pragma omp declare target
 void foo ( ) {
   /* code 1 */

   #pragma omp target teams distribute parallel for
   for ( int i = 0 ; i < count ; ++i )
      foo ( ) ;

   /* code 2 */
}
#pragma omp end declare target
```

MACC

```
void foo ( ){
  /* code 1 */
  gpu_foo<<< . . . >>> (... , 0) ;
  /* code 2 */
}

__global__ gpu_foo (... , int depth ) {
  . . .
  if ( depth > MAXDEPTH)
     dev_in_foo ( ) ;
  else {
     /* code 1 */
     gpu_foo<<< . . . >>> ( . . . , depth + 1 ) ;
     /* code 2 */
  }
}

__device__ dev_in_foo( ) {
   /* code 1 */
   for ( int i = 0 ; i < count ; ++i )
      dev_in_foo( ) ;

   /* code 2 */
}
```

Evaluation

- **GOALS**
  - Apply DP onto workload-based applications to gain insight how it effects
  - Observe performance when DP is limited by using if clause

- **Hardware :** NVidia K40c (2888 CUDA cores, 12GB memory)
- **Software**
  - OpenACC | PGI compiler 15.7
  - OpenMP 4.0 | MACC compiler
  - CUDA | NVCC 7.0
  - Backend Host | GCC 4.9

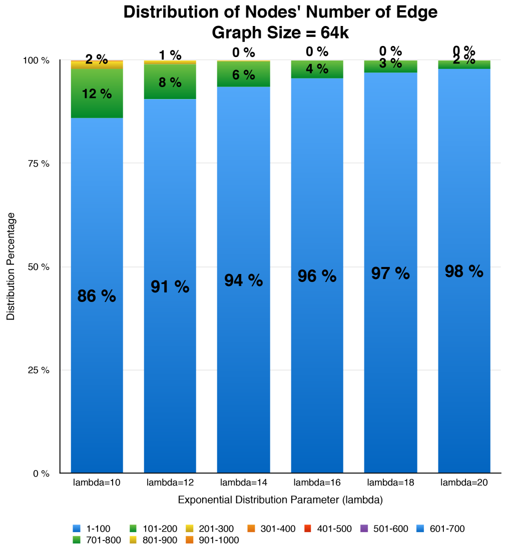
Sparse Matrix Vector Multiplication (CSR)

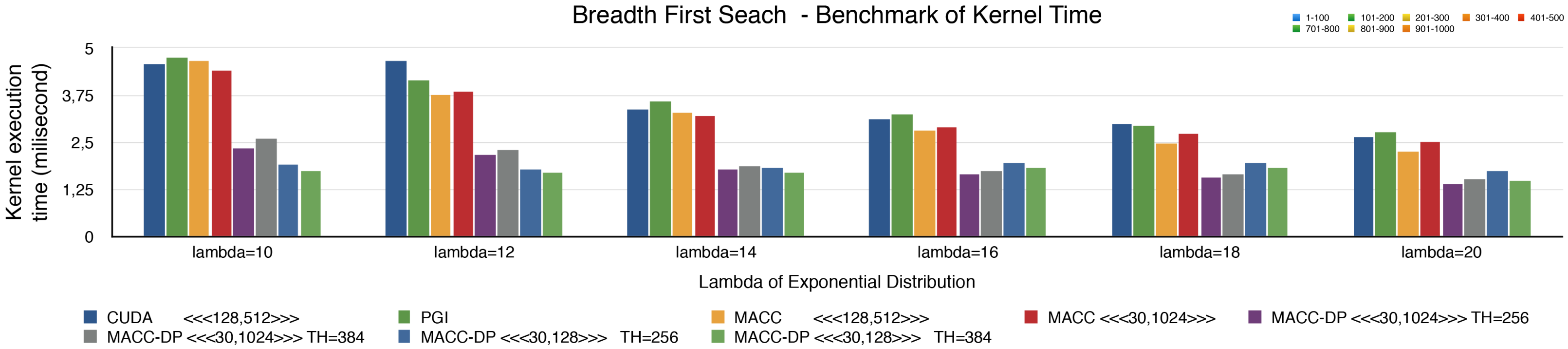
- 7 matrixes are used from University of Florida sparse matrix collection
- Matrixes are compressed CSR format
- They are mostly filled by zero
- Some rows have excessive item

Matrix	Size	NNZ	IIT > 256	IIT > 128
rajat30	643994	6175377	121	277
ASIC 680k	682862	3871773	2	76
rajat29	643994	4866270	22	27
transient	178866	961790	13	31
c-big	345241	2341011	2	160
Raj-1	263743	1302464	65	93
trans5	116835	766396	8	32

Breadth First Search

- BFS is ported from Rodinia Benchmark Suite
- Six graphs are used. Their nodes' number of edge are generated by exponential distribution.
- Some nodes of graph have massive amount of edges
  - DP is activated for these nodes

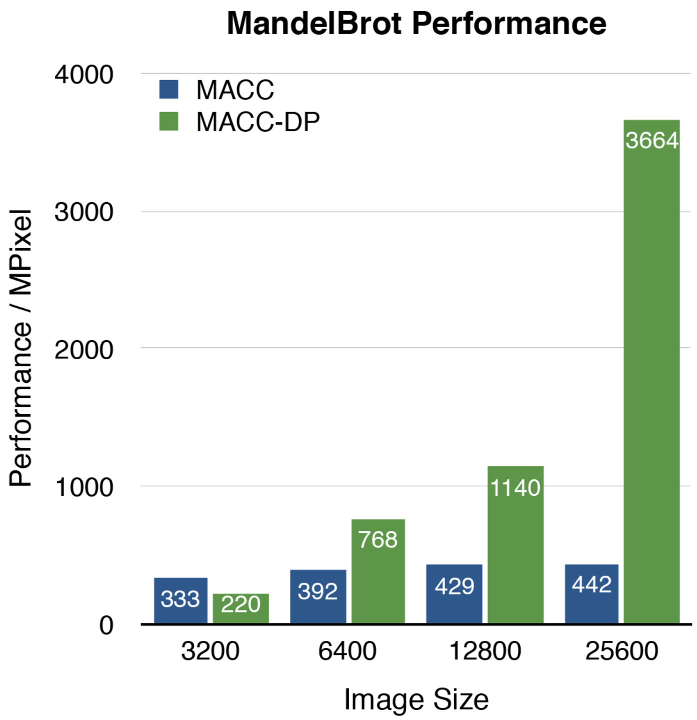




Mandelbrot

Two algorithms are used

- **Escape algorithm**
  - Each pixel is handled by CUDA thread
- **Mariani-Silver algorithm**
  - Block by block
  - Involves recursion



Conclusion & Future work

- Dynamic parallelism is able to gain performance, but it incurs some overheads
- **In this paper we showed:**
  1. We don’t need activate DP in every single case
  2. Essentially it is good for irregular cases
  3. When it is used conditionally (only irregular case), it could increase performance
- **In directive based Programming Models:**
  1. Expressing explicitly DP is useful since user might know their application behavior
  2. Conditional usage is possible

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

To download MACC OpenMP 4.0+OmpSs compiler

PLEASE ASK

guray.ozen@bsc.es

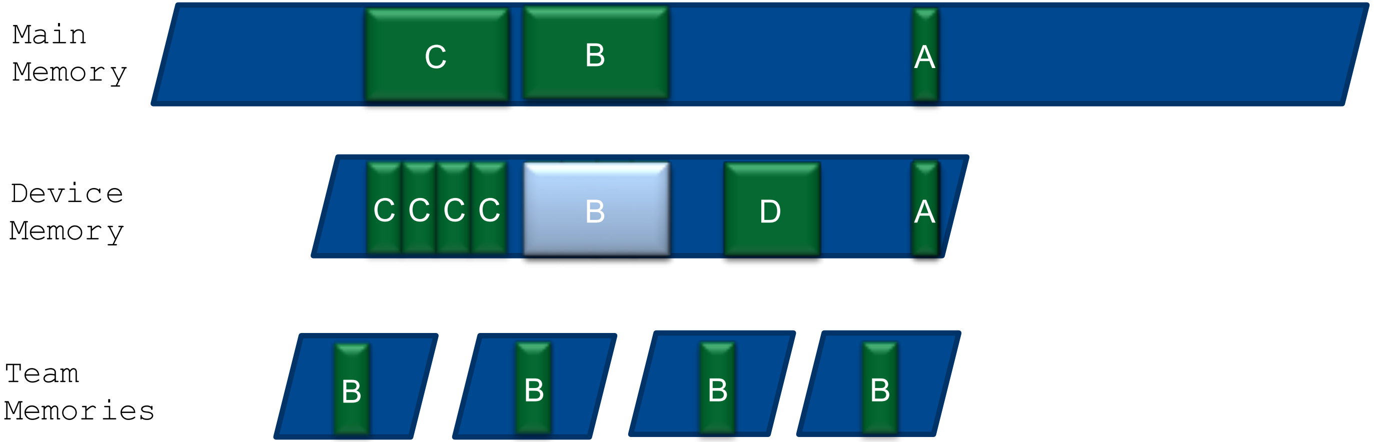
- Data sharing clauses with `teams` | `private` | `first_private`
- Offers experimental 3 new clauses for `distribute` directive
  - `dist_private([CHUNK]data1, [CHUNK]data2 …)`
  - `dist_firstprivate([CHUNK]data1, [CHUNK]data2 …)`
  - `dist_lastprivate([CHUNK]data1, [CHUNK]data2 …)`

Data movement to **Device Memory**

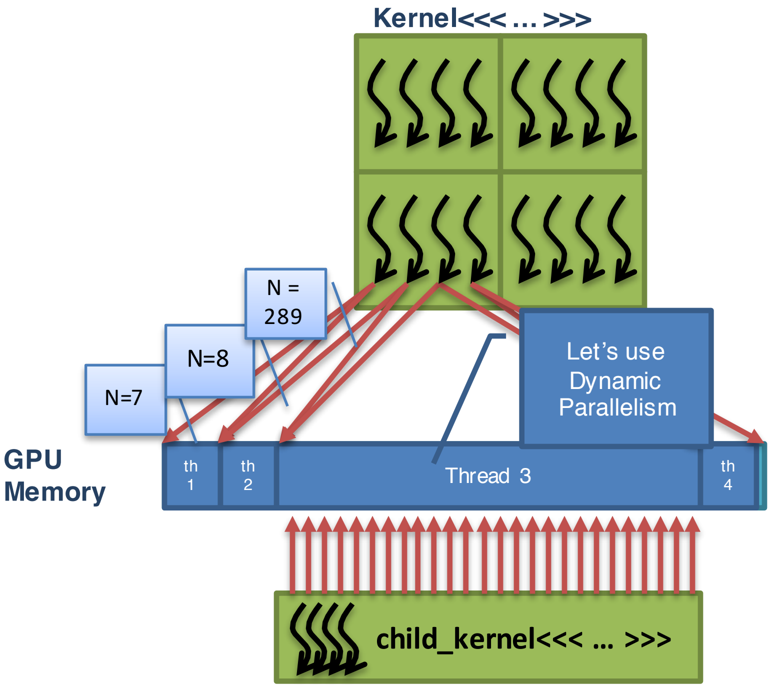
```
#pragma omp target device(acc) copy_deps
#pragma omp task in(A[0:SMALL],C[0:HUGE]) inout(B[0:HUGE]) out(0:D[BIG])
#pragma omp teams first_private(A)
#pragma omp distribute parallel for dist_first_private([CHUNK]C) dist_first_last_private([CHUNK]B)
for(...)
   <<..Computation..>>
```

Using **TeamMem** with **Small DATA**

Using **TeamMem** with **Big DATA**



Motivation




```
__global__ kernel(double* A, double* X, int* iter, int N, double alpha)
{
   int i = threadIdx.x + blockDim.x * gridDim.x;
   <.. Code ..>
   N = iter[i] - iter[i+1];

   for (j = iter[i]; j < iter[i+1]; j++)
     X[j] += A[j] * alpha;

   <.. Code ..>
}
```

Workload dependency

Also not coalesced

- What has changed when DP is activated ?
  - Granularity is increased
  - Memory behavior could be changed
  - Control-flow behavior could be changed