OpenMP®

SC'21 Booth Talk Series


# Combining OpenMP tasking and target (GPU) offloading on heterogeneous systems

## A proposal for OpenMP target tasks

**Ph.D Pedro Valero-Lara**, Computer Scientist, ORNL

# Hello!

- Ph.D Pedro Valero-Lara (valerolarap@ornl.gov)
  - Computer Scientist at Oak Ridge National Laboratory
    - Computer Science and Mathematics Division
      - Advanced Computing Systems Research Section
        - » Programming System Group
- ECP project Proteas-tune
  - Lead by Jeffrey Vetter (head of the Advanced Computing Systems Research Section)

U.S. DEPARTMENT OF ENERGY

OAK RIDGE National Laboratory

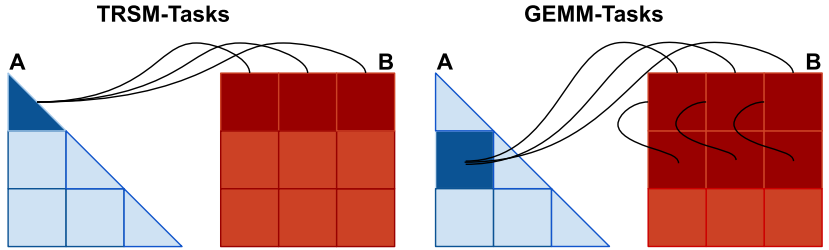
ECP EXASCALE COMPUTING PROJECT

# Motivation

- Heterogeneous programming
  - Integration of OpenMP tasking with GPU offloading (OpenMP 4.5)
- Motivating example
  - BLAS-3 TRSM
- OpenMP new specification proposal
  - OpenMP target tasking

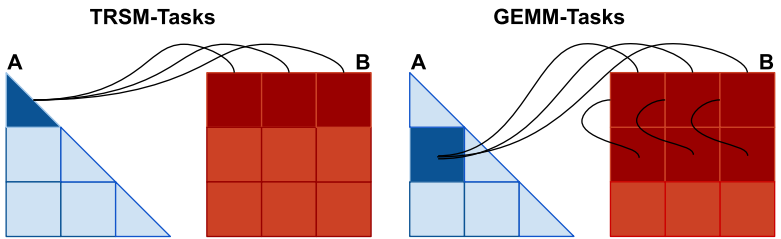
# TRSM (Triangular solve) - BLAS Level 3 routine

- Widely used dense linear algebra operation
  - Ax = B -> A is either a Lower or Upper triangular matrix and B is a dense matrix
- Parallel TRSM
  - Decompose matrices into tiles
  - Two main operations:
    - TRSM -> Good performance on CPU, not so good on GPU
    - GEMM -> Good performance on GPU and CPU
  - Parallelism and data-dependences can be described by using tasking
- Very similar to other BLAS Level 3 and LAPACK routines



# Heterogeneous TRSM - OpenMP tasking and target

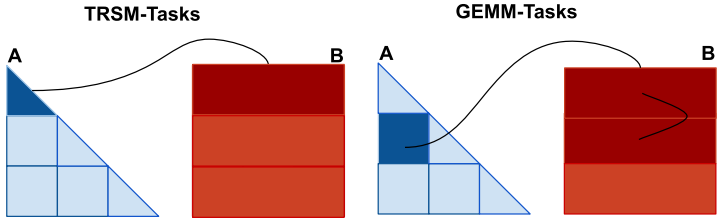
- Uniform decomposition
  - Same tile size in A and B decomposition
- High number of tasks
- High number of CPU-GPU transfers
- Low IPC on GPU-GEMM
  - GEMM on “small” matrices




```
1  aSIZE = TILE_SIZE*TILE_SIZE;
2  for(d = 0; d < dt; d++) {
3    for(c = 0; c < ct; c++) {
4      #pragma omp task depend(in:TILE_A[d][d]) \
5                       depend(inout:TILE_B[d][c])
6      CPU-TRSM(L, L, N, N,
7               TILE_SIZE, TILE_SIZE,
8               ALPHA, TILE_A[d][d], TILE_SIZE,
9                      TILE_B[d][c], TILE_SIZE);
10   }
11   for(r = d+1; r < rt; r++) {
12     for(c = 0; c < ct; c++) {
13       #pragma omp task depend(in:TILE_A[r][d]) \
14                        depend(in:TILE_B[d][c]) \
15                        depend(inout:TILE_B[r][c]) {
16         TILE_A=TILE_A[r][d];TILE_B=TILE_B[d][c];
17         TILE_C=TILE_B[r][c];
18         #pragma omp target enter data map
           ↪  (to:TILE_A[0:aSIZE],TILE_B[0:aSIZE],TILE_C[0:aSIZE])
19         #pragma omp target data use_device_ptr(TILE_A,TILE_B,TILE_C) {
20           GPU-GEMM(N, N,
21                    TILE_SIZE, TILE_SIZE, TILE_SIZE,
22                    -1.0, TILE_A, TILE_SIZE,
23                          TILE_B, TILE_SIZE,
24                   ALPHA, TILE_C, TILE_SIZE);
25         }//End pragma target
26         #pragma omp target exit data map(from:TILE_C[aSIZE])
27       }//End pragma task
28     }//End for c
29   }//End for r
30 }//End for d
31 }
```

# Heterogeneous TRSM - OpenMP tasking and target

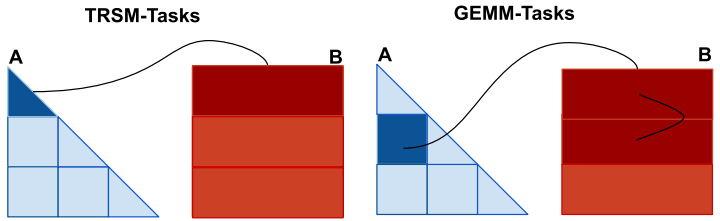
- Non-Uniform decomposition
  - Square tiles for A matrix
  - Rectangular tiles for B matrix
- **Maximize GPU computation**
- Lower number of tasks
- Lower number of CPU-GPU transfers
- Higher IPC on GPU-GEMM
  - GEMM on bigger matrices




```
1  aSIZE = TILE_SIZE*TILE_SIZE;
2  bSIZE = TILE_SIZE*MATRIX_SIZE;
3  for(d = 0; d < dt; d++) {
4    #pragma omp task depend(in:TILE_A[d][d]) \
5                     depend(inout:TILE_B[d])
6    CPU-TRSM(L, L, N, N,
7             TILE_SIZE, MATRIX_SIZE,
8             ALPHA, TILE_A[d][d], TILE_SIZE,
9                    TILE_B[d], TILE_SIZE);
10   for(r = d+1; r < rt; r++) {
11     #pragma omp task depend(in:TILE_A[d][r]) \
12                      depend(in:TILE_B[d]) \
13                      depend(inout:TILE_B[r]) {
14       TILE_A=TILE_A[r][d];TILE_B=TILE_B[d];TILE_C=TILE_B[r]
15       #pragma omp target enter data map
         ↪  (to:TILE_A[0:aSIZE],TILE_B[0:bSIZE],TILE_C[0:bSIZE])
16       #pragma omp target data use_device_ptr(TILE_A,TILE_B,TILE_C) {
17         GPU-GEMM(N, N,
18                  TILE_SIZE, MATRIX_SIZE, TILE_SIZE,
19                  -1.0, TILE_A, TILE_SIZE,
20                        TILE_B, TILE_SIZE,
21                  ALPHA, TILE_C, TILE_SIZE);
22       }//End pragma target
23       #pragma omp target exit data map(from:TILE_C[bSIZE])
24     }//End pragma task
25   }//End for r
26 }//End for d
```

# Heterogeneous TRSM - CUDA

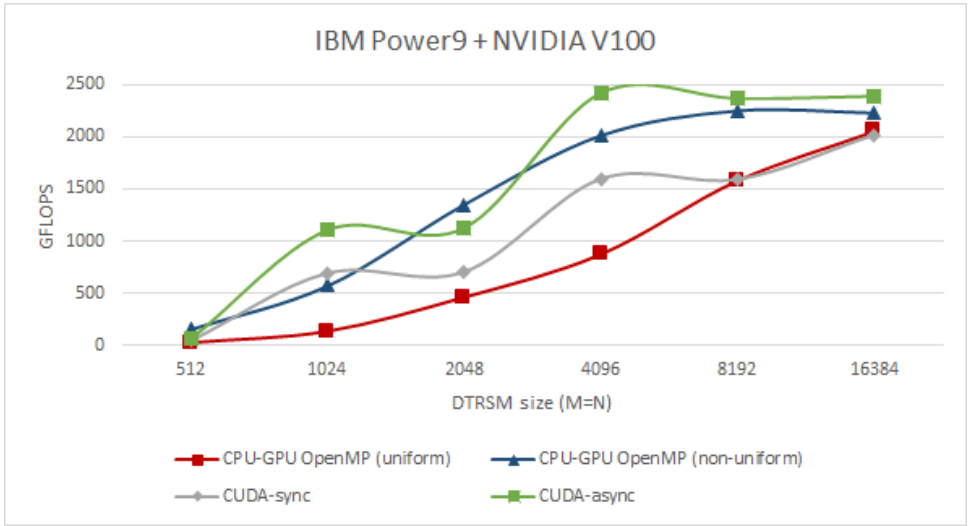
- More complex to implement
  - Synchronization
    - Streams, async APIs, explicit synchronization, events, …
  - CPU-GPU communication
    - Explicit use of communication functions
  - Combining Libraries (CUDA and cuBLAS)
    - cublasSetStream, …




```
cuStream_t stream0, stream1;
cuEvent_t event;
cuHandle_t handle;
//First iteration
cblas_dtrsm(L, L, N, N,
        TILE_SIZE, MATRIX_SIZE,
        ALPHA, TILE_A[0][0], TILE_SIZE,
                    TILE_B0, TILE_SIZE);
cudaEventRecord(event);
cudaEventSynchronize(event);
cublasSetMatrixAsync(TILE_SIZE,MATRIX_SIZE,sizeof(precision),
                     TILE_B[0], TILE_SIZE,
                    TILE_BGPU0, TILE_SIZE, streams[0]);
//First GEMM block
cublasSetMatrixAsync(TILE_SIZE, TILE_SIZE, sizeof(precision),
                  TILE_A[1][0], TILE_SIZE,
                    TILE_AGPU1, TILE_SIZE, streams[0]);
cublasSetMatrixAsync(TILE_SIZE,MATRIX_SIZE,sizeof(precision),
                     TILE_B[1], TILE_SIZE,
                     TILE_GPUB1, TILE_SIZE, streams[0]);
cublasSetStream(handle, streams[0]);
GPU-GEMM(N, N,
        TILE_SIZE, MATRIX_SIZE, TILE_SIZE,
             -1.0, TILE_GPUA1, TILE_SIZE,
                   TILE_GPUB0, TILE_SIZE,
            ALPHA, TILE_GPUB1, TILE_SIZE);
cublasGetMatrixAsync(TILE_SIZE,MATRIX_SIZE,sizeof(precision),
                     TILE_GPUB1, TILE_SIZE,
                      TILE_B[1], TILE_SIZE, streams[0]);
//Second GEMM block
cublasSetMatrixAsync(TILE_SIZE, TILE_SIZE, sizeof(precision),
                  TILE_A[2][0], TILE_SIZE,
                    TILE_AGPU2, TILE_SIZE, streams[1]);
cublasSetMatrixAsync(TILE_SIZE,MATRIX_SIZE,sizeof(precision),
                     TILE_B[2], TILE_SIZE,
                     TILE_GPUB2, TILE_SIZE, streams[1]);
cublasSetStream(handle, streams[1]);
cublasDgemm(N, N,
        TILE_SIZE,MATRIX_SIZE,TILE_SIZE,
             -1.0, TILE_GPUA2, TILE_SIZE,
                   TILE_GPUB0, TILE_SIZE,
            ALPHA, TILE_GPUB2, TILE_SIZE);
...
```

# Performance

- SUMMIT
  - 2 IBM CPU Power 9
  - 1 NVIDIA GPU V100 (Volta)
- CPU-GPU
  - Non-uniform beats uniform
  - CUDA-async faster than OpenMP
  - OpenMP gets 85-95% performance of CUDA



# Proposal - Target tasks

- Motivation
  - Better programming productivity
    - Simpler heterogeneous code
  - Better CPU-GPU communication
    - Minimize memory transfers
- OpenMP target task proposal:

  **#pragma omp target task depend(in:TILE_A[r][d]) depend(in:TILE_B[d][c]) …**
- Current specification:

  #pragma omp task depend(in:TILE_A[r][d]) depend(in:TILE_B[d][c])…

  #pragma omp target enter data map(to:TILE_A[0:aSIZE],TILE_B[0:aSIZE], …)

  #pragma omp target data use_device_ptr(TILE_A,TILE_B,TILE_C)

  #pragma omp target exit data map(from:TILE_C[aSIZE])

# Conclusions

- Non-uniform (**tasking**) workflows are important for performance
- Heterogeneous OpenMP is possible
  - Integrating target offloading with tasking
- OpenMP, although not faster, presents a much higher programming productive and portable
  - Performance closed to the performance of an optimized CUDA code
- OpenMP target tasks
  - Increase programming productivity
  - Better and transparent exploitation of CPU and GPU

# OpenMP®

## SC'21 Booth Talk Series

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

**link.openmp.org/sc21** Videos and PDFs of OpenMP SC'21 presentations