TASKING IN ACCELERATORS

MNHACK19

Leonel Toledo, Orestis Korakitis, Kazuaki Matsumura /
Barcelona Supercomputing Center

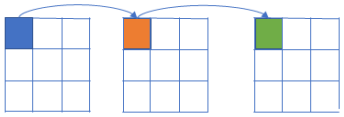
The EPEEC project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement Nº 801051

Tasking in Accelerators

GPU compute capabilities have been significantly increasing; however, applications and scalability of algorithms still face some important challenges. One important problem regarding scalability is hardware resource assignment.

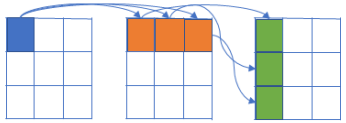
Tasking, on the other hand makes it possible for algorithms with run-time dependent execution flow, to be parallelized.

```
for (int i =0; i<TILES; i++){
        for (int j=0; j< TILES; j++){
                for (int k=0; k< TILES; k++){
                Dgemm(…)
                synch();
                }
        }
}
```



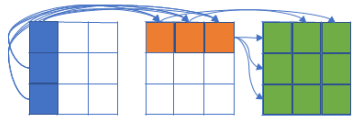
A[i][k] * B[k][j] = C[i][j]

```
for (int j =0; j<TILES; j++){
        for (int k=0; k< TILES; k++){
                for (int i=0; i< TILES; i++){
                Dgemm(…)
                }
        synch();
        }
}
```



A[i][k] * B[k][j] = C[i][j]

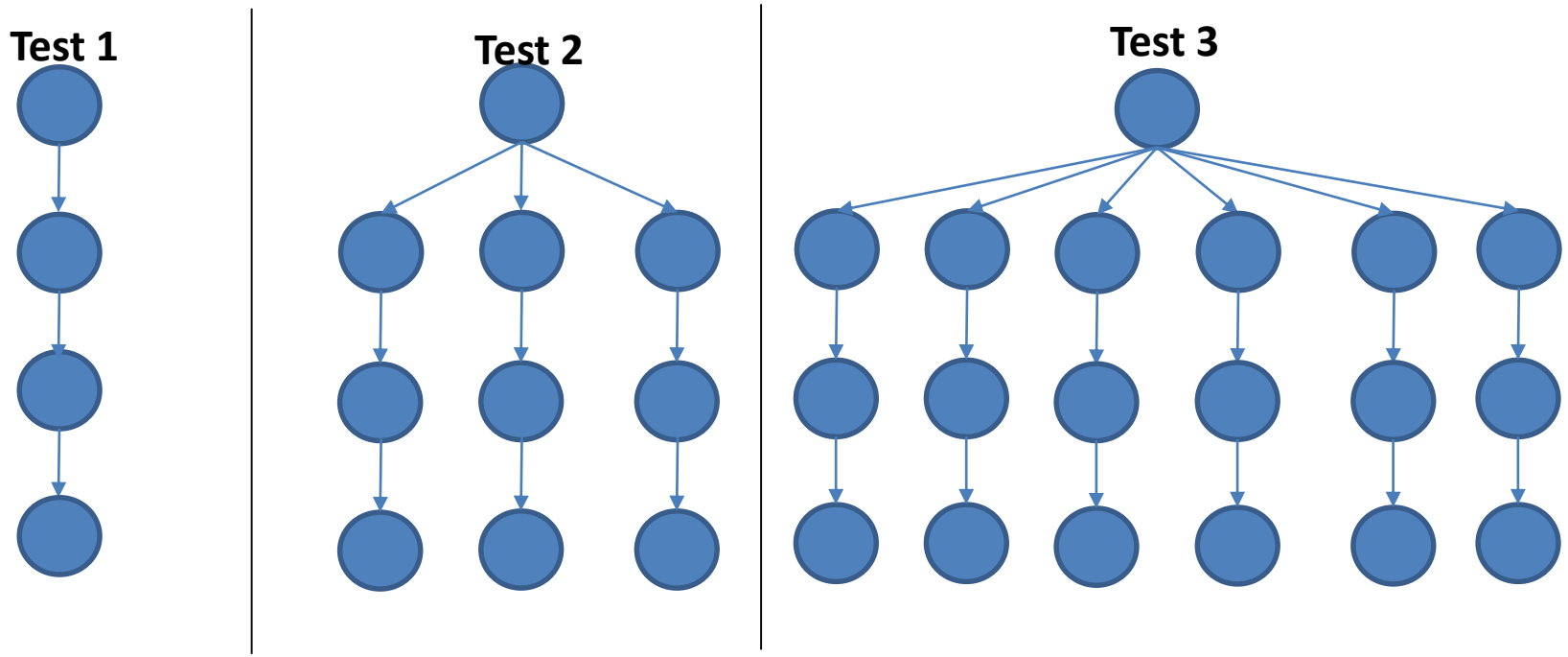
```
for (int k =0; k<TILES; k++){
        for (int i=0; i< TILES; i++){
                for (int j=0; j< TILES; j++){
                Dgemm(…)
                }
        }
        synch();
}
```

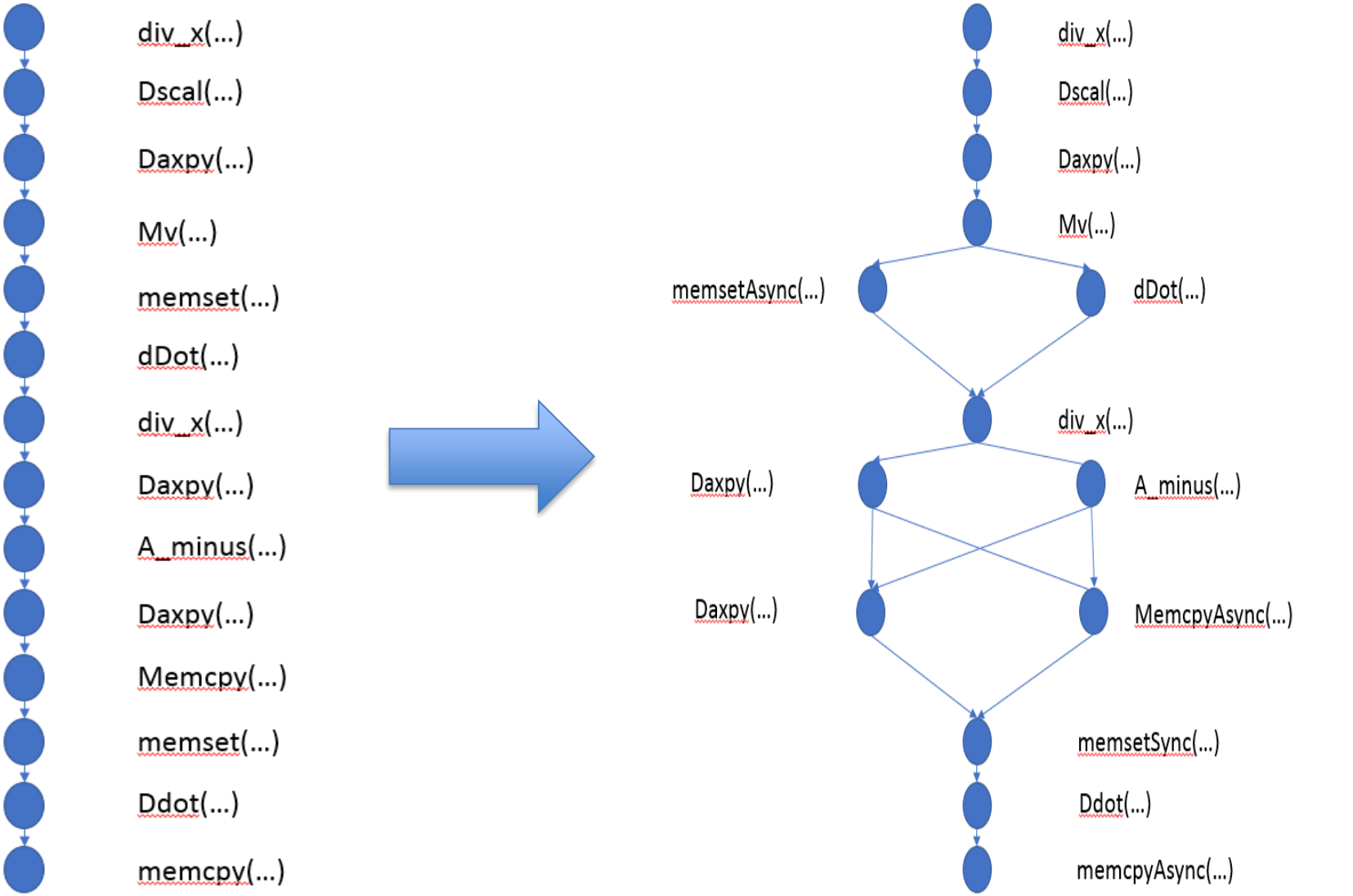


A[i][k] * B[k][j] = C[i][j]

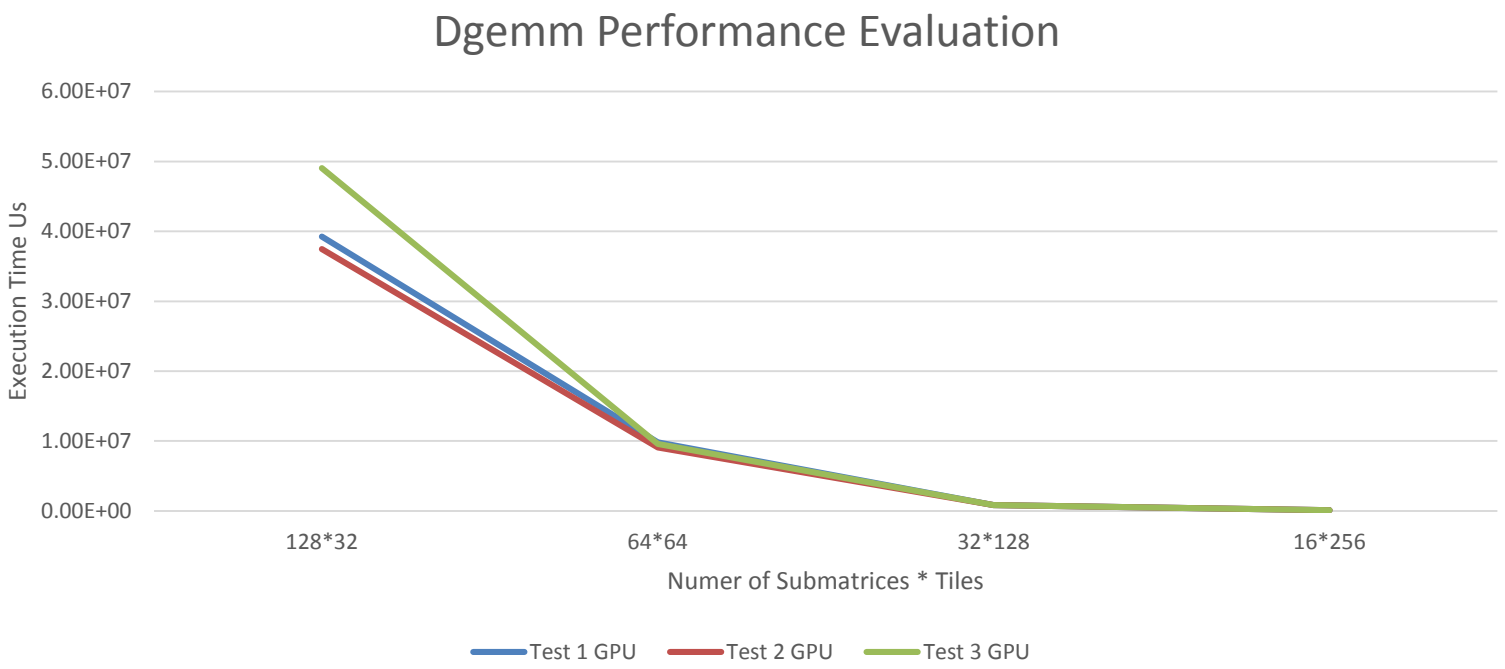
Tasking in Accelerators

- Many HPC applications such as deep neural network training and scientific simulations have an iterative structure where the same workflow is executed repeatedly. CUDA streams require that the work be resubmitted with every iteration, which consumes both time and CPU resources. Graphs enable a *define-once-run-repeatedly* execution flow.




div_x(...)
Dscal(...)
Daxpy(...)
Mv(...)
memset(...)
dDot(...)
div_x(...)
Daxpy(...)
A_minus(...)
Daxpy(...)
Memcpy(...)
memset(...)
Ddot(...)
memcpy(...)
div_x(...)
Dscal(...)
Daxpy(...)
Mv(...)
memsetAsync(...)
dDot(...)
div_x(...)
Daxpy(...)
A_minus(...)
Daxpy(...)
MemcpyAsync(...)
memsetSync(...)
Ddot(...)
memcpyAsync(...)

Performance Evaluation



JIT Compilation of OpenACC

Original Code

```
#pragma acc data copyin (…)
#pragma acc data copyout (…)
for (int t = 0; t < TIME_STEP; t++) {
  #pragma acc parallel loop independent
  for (int i = 0; i < N; i++) { /* ... */ }

  #pragma acc parallel loop independent
  for (int i = 0; i < N; i++) {
    p_x[i] += v_x[i] * DT;
    p_y[i] += v_y[i] * DT;
    p_z[i] += v_z[i] * DT;
  }
```

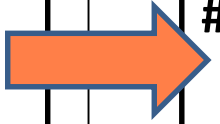
Translated Code

```
jacc_copyin(…);

for (int t = 0; t < TIME_STEP; t++) {
  jacc_kernel_push("#pragma acc …");
  jacc_kernel_push("#pragma acc …");
}

jacc_copyout(…);
```

for () {
#pragma acc parallel { /* (A) */ }
#pragma acc parallel { /* (B) */ } }

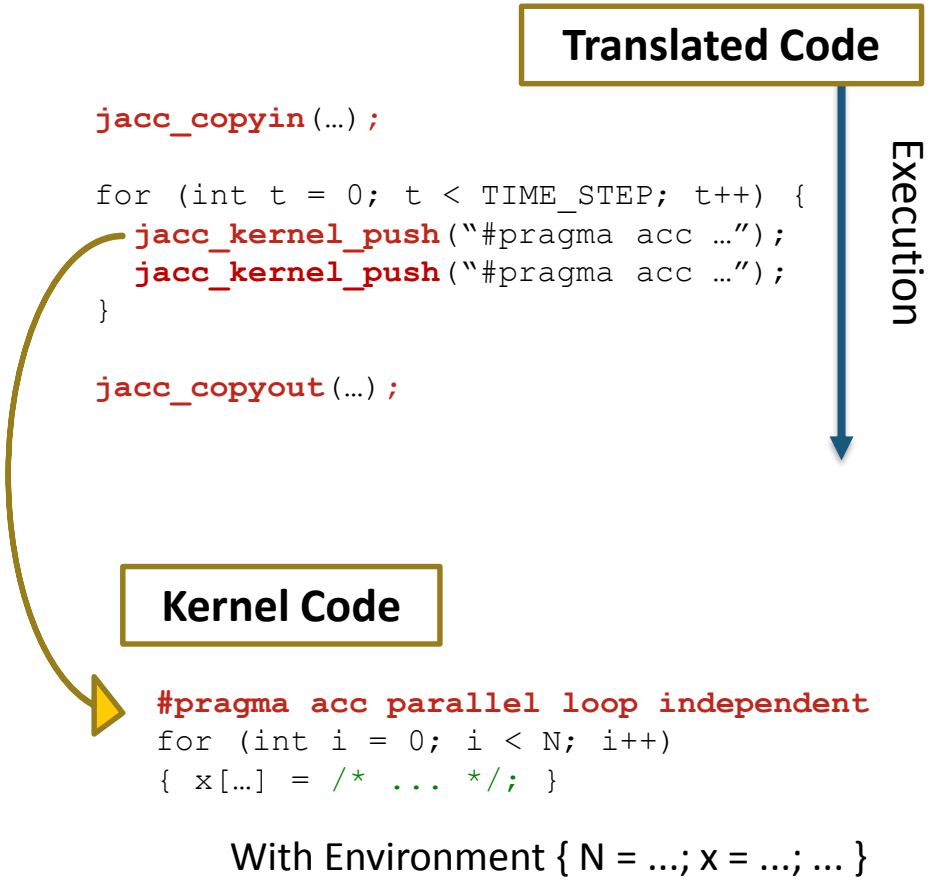


#pragma acc parallel
{ for () { /* (A) (B) */ } }

- Supporting just-in-time (runtime) compilation of a directive-based programming model by source-to-source conversion
- Allows kernel fusion and constant expansion for aggressive optimization

JIT Compilation of OpenACC

- Extracted kernels are compiled with dynamic information which the runtime informs (variables are optionally placed by constant)
- Compiled kernels are linked to the user program and executed on the device as with original code



OmpSs-2 + OpenACC

- Main Idea of OmpSs:
  - Taskifying complex programs
  - Let the runtime discover parallelism

- Support for devices
  - Programs can have tasks that run on different devices (GPUs, FPGAs)

OmpSs-2 + OpenACC

```
#pragma oss task device(openacc) in(…) out(…)
#pragma acc kernels
for (int y=ny0; y < nyf; y++) {
   for (int x=nx0; x < nxf; x++) {
      for (int z=nz0; z < nzf; z++) {
         …code…
      }
   }
}
```

Thank You!

EPEEC
PRODUCTIVITY AT EXASCALE

www.epeec-project.eu

The EPEEC project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement № 801051