DE LA RECHERCHE À L'INDUSTRIE

Enhancing MPI + OpenMP Task based Applications for Heterogenous Architectures with GPU support

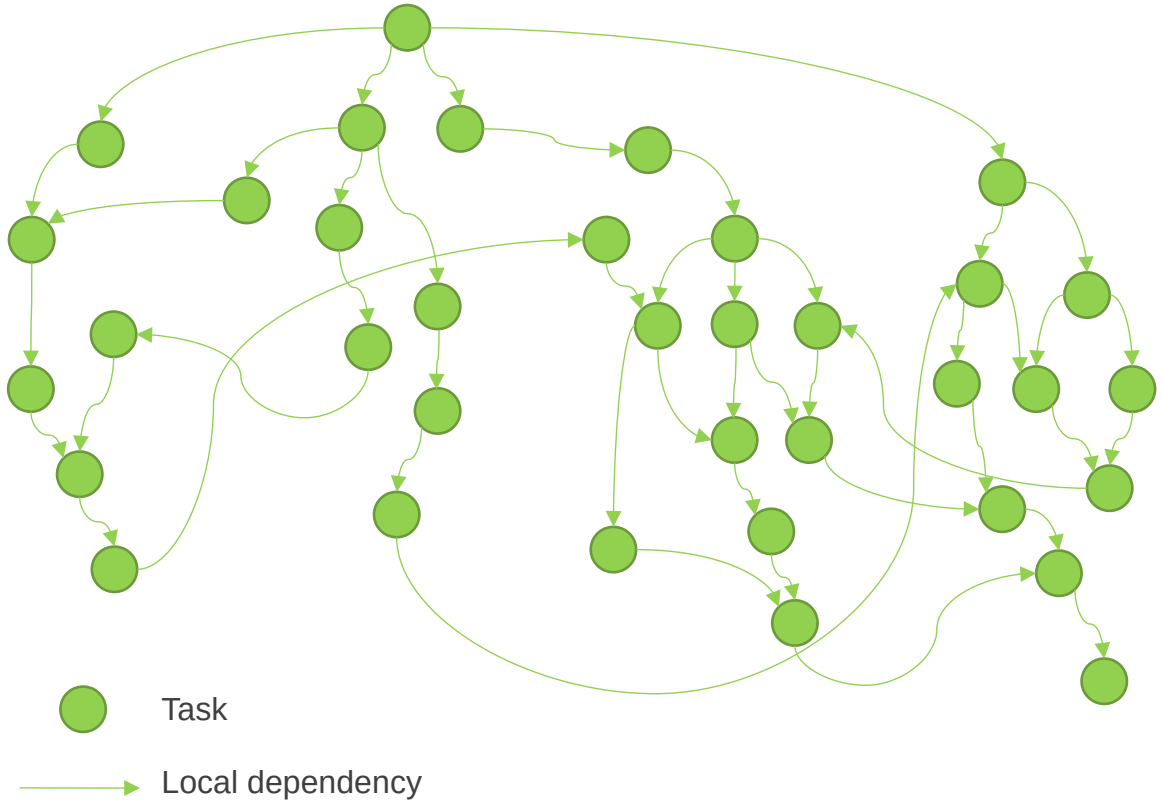
18th International Workshop on OpenMP, IWOMP 2022 Chattanooga, TN, USA, September 27–30, 2022

Manuel FERAT[1] – Romain PEREIRA[2,4] – Adrien ROUSSEL[2,3] – Patrick CARRIBAULT[2,3] – Luiz Angelo STEFFENEL[1] – Thierry GAUTIER[4]

[1] Laboratoire d'informatique en calcul intensif et image pour la simulation
[2] CEA, DAM, DIF, F-91297 Arpajon Cedex
[3] Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, 91680 Bruyères-le-Châtel, France
[4] Project Team AVALON INRIA, LIP, ENS-Lyon, Lyon, France

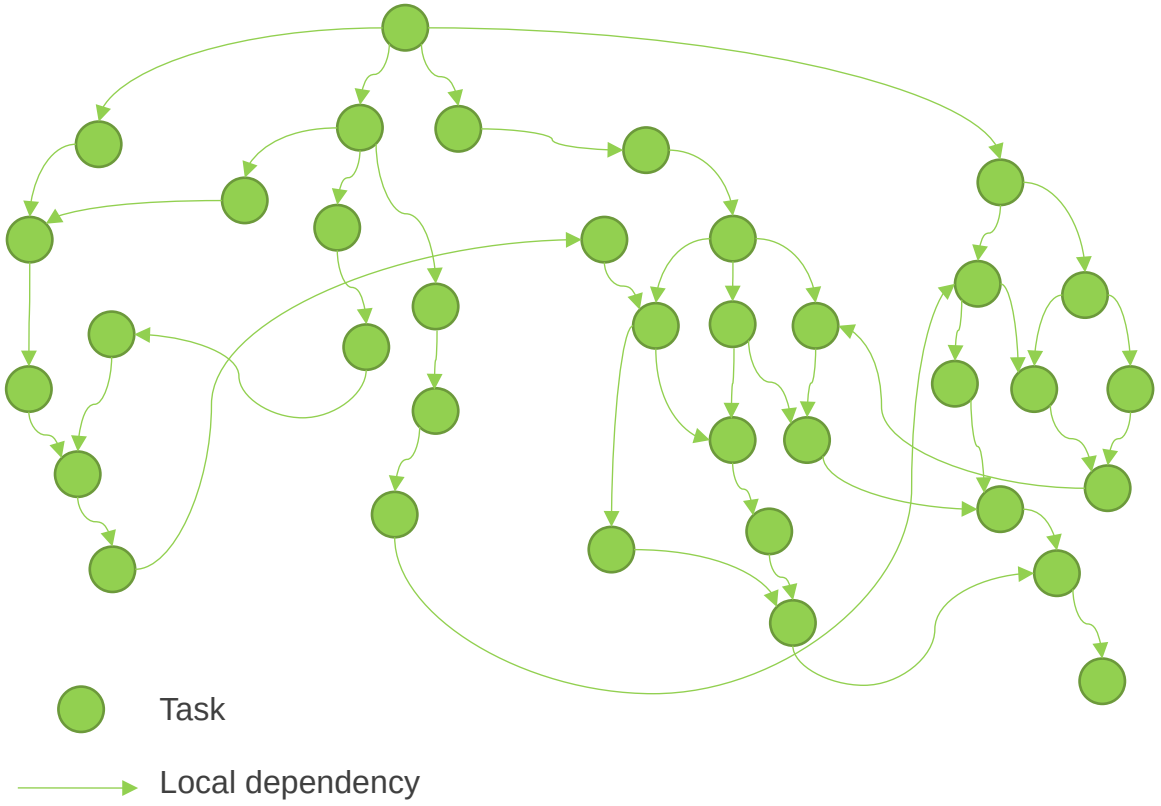
Context

- **Dependent Task-based programming model**
  - Application' parallelism represented as directed acyclic graph (DAG)
    - Node : tasks, code to execute with its associated data
    - Arcs : precedence constraints



Context

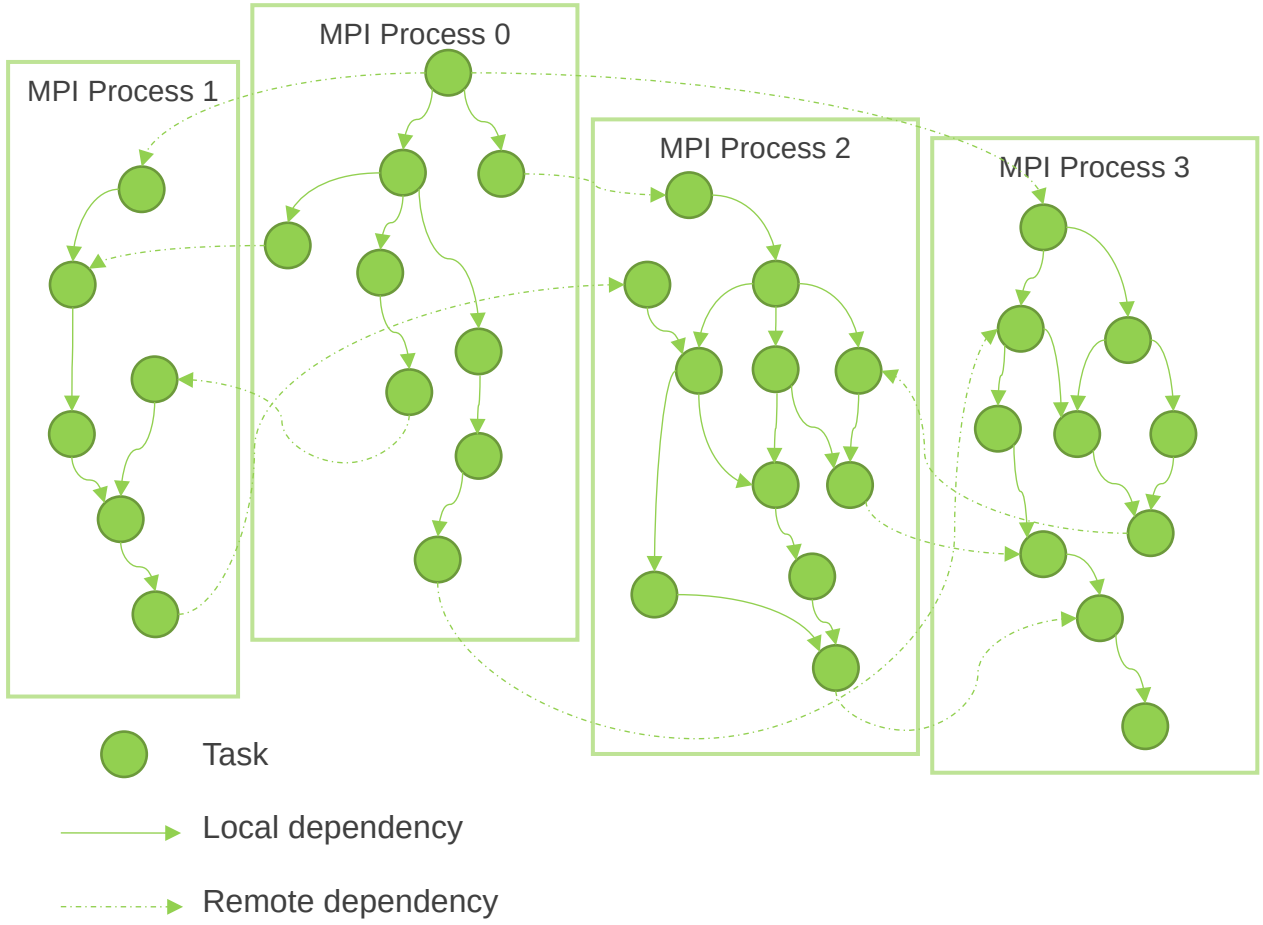
- **Dependent Task-based programming model**
  - Application' parallelism represented as directed acyclic graph (DAG)
    - Node : tasks, code to execute with its associated data
    - Arcs : precedence constraints
  - Efficiency of the model relies on the task scheduling over **processing units (CPU, GPU)**
    - **Automatic overlap of asynchronous operations through task-switches**



Context

- **Dependent Task-based programming model**
  - Application' parallelism represented as directed acyclic graph (DAG)
    - Node : tasks, code to execute with its associated data
    - Arcs : precedence constraints
  - Efficiency of the model relies on the task scheduling over **processing units (CPU, GPU)**
    - **Automatic overlap of asynchronous operations through task-switches**

- **Distributed task-based HPC applications**
  - MPI+OpenMP(tasks) applications
  - Distributed task dependency graph

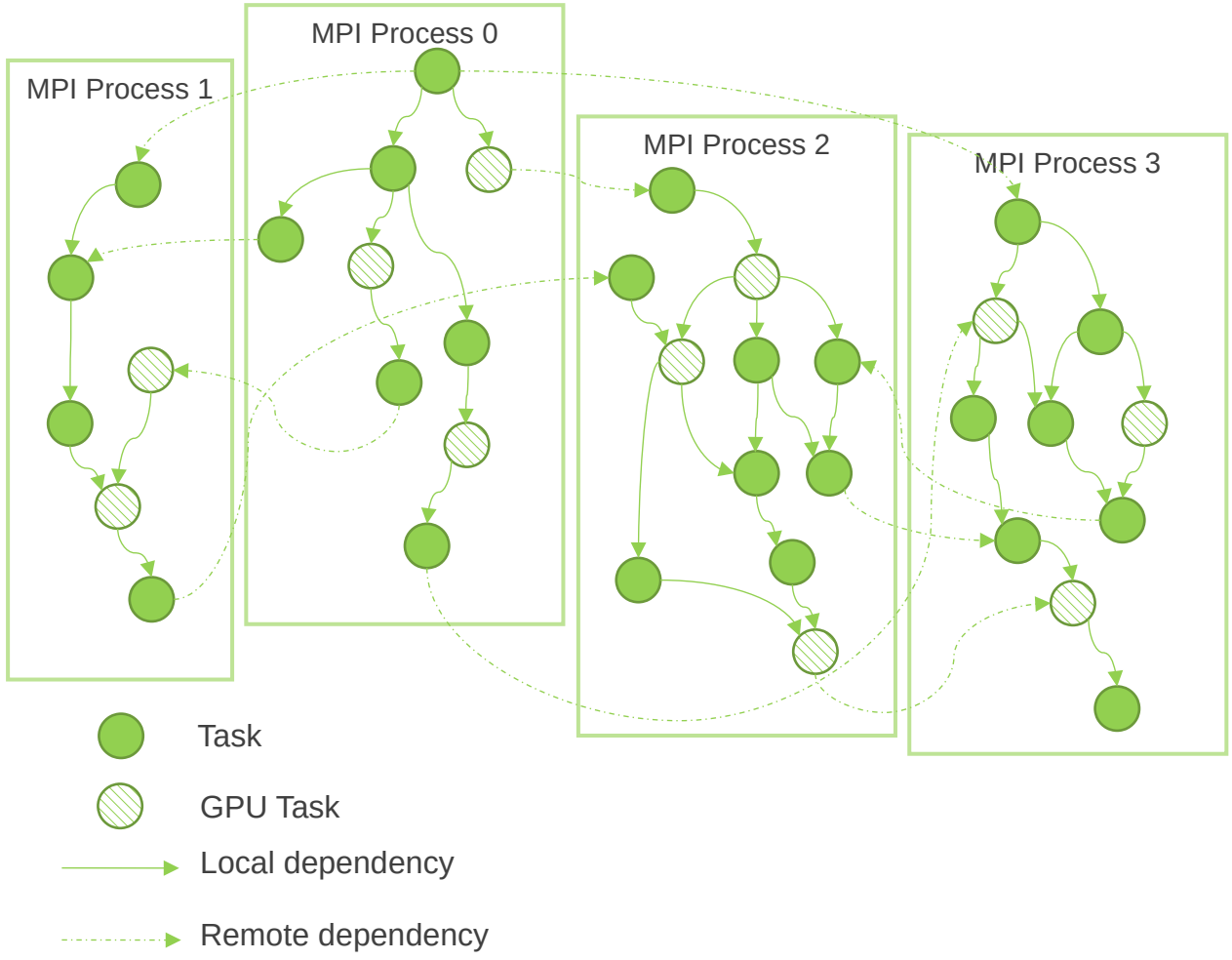


Context

- **Dependent Task-based programming model**
  - Application' parallelism represented as directed acyclic graph (DAG)
    - Node : tasks, code to execute with its associated data
    - Arcs : precedence constraints
  - Efficiency of the model relies on the task scheduling over **processing units (CPU, GPU)**
    - **Automatic overlap of asynchronous operations through task-switches**

- **Distributed task-based HPC applications**
  - MPI+OpenMP(tasks) applications
  - Distributed task dependency graph

- **Heterogenous programming**

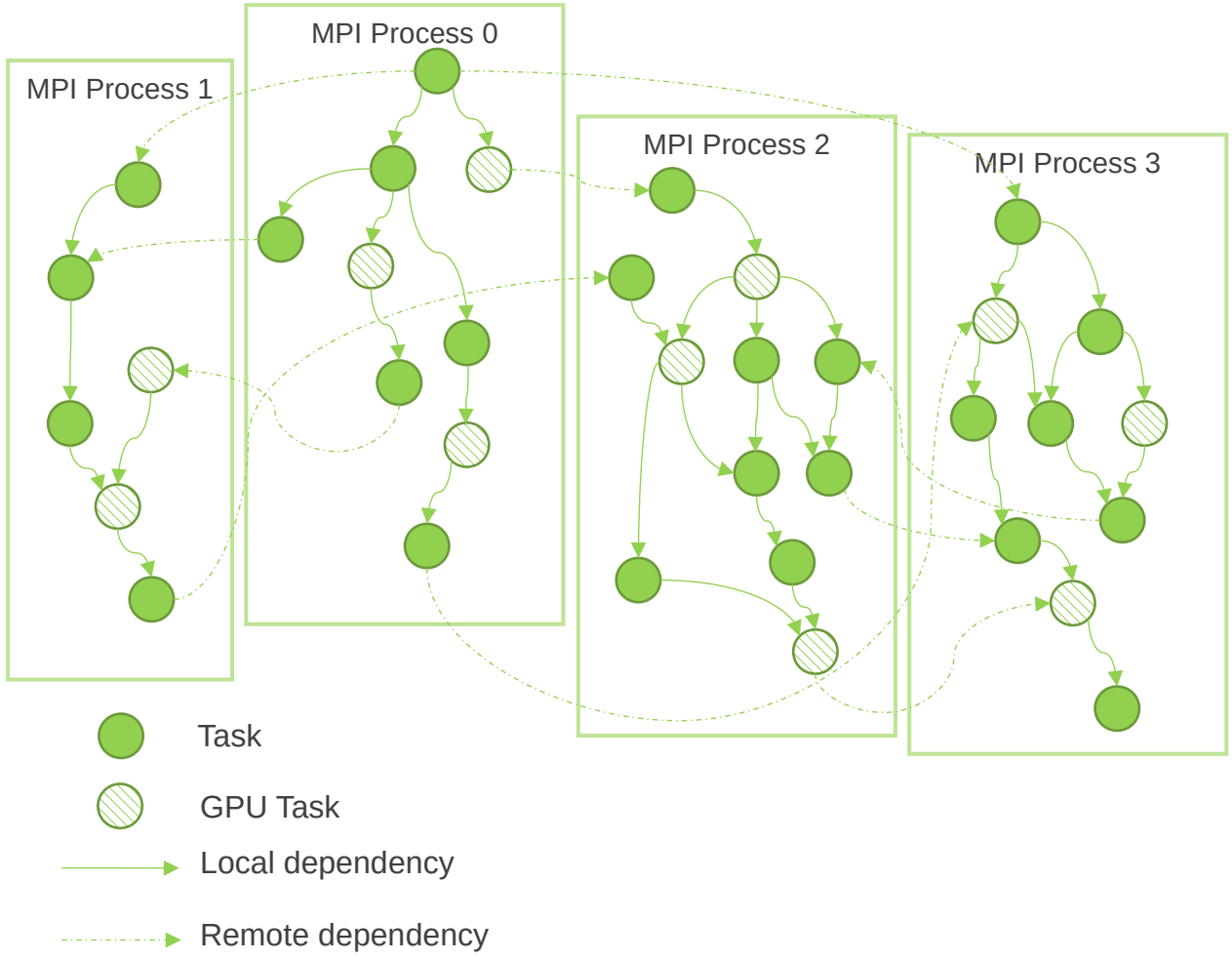


Context

- **Dependent Task-based programming model**
  - Application' parallelism represented as directed acyclic graph (DAG)
    - Node : tasks, code to execute with its associated data
    - Arcs : precedence constraints
  - Efficiency of the model relies on the task scheduling over **processing units (CPU, GPU)**
    - **Automatic overlap of asynchronous operations through task-switches**

- **Distributed task-based HPC applications**
  - MPI+OpenMP(tasks) applications
  - Distributed task dependency graph

- **Heterogenous programming**



⟹ How the task-based programming model is implemented by OpenMP

Context

- **OpenMP task model**
  - Version 3.0 – 2008 introduced tasks programming

Context

- **OpenMP task model**
  - Version 3.0 – 2008 introduced tasks programming
  - Version 4.0 – 2013
    - dependent tasks

Context

- **OpenMP task model**
  - Version 3.0 – 2008 introduced tasks programming
  - Version 4.0 – 2013
    - dependent tasks
    - GPU offloading through targets directive (not with tasks yet)

```
1   #pragma omp target teams distribute parallel for \
2   map(to: a[0:N], b[0:N]) map(from: c[0:N])
3   for (i = 0; i < N, i++){
4       c[i] = a[i] + b[i];
5   }
```

Vectors addiction OpenMP Target example

Context

- **OpenMP task model**
  - Version 3.0 – 2008 introduced tasks programming
  - Version 4.0 – 2013
    - dependent tasks
    - GPU offloading through targets directive (not with tasks yet)
  - Version 4.5 – 2015 extended target for task programming
    - depend and nowait clause

```
1   #pragma omp target teams distribute parallel for \
2   map(to: a[0:N], b[0:N]) map(from: c[0:N]) \
3   depend(in: a[0], b[0]) depend(out: c[0]) nowait
4   for (i = 0; i < N, i++){
5       c[i] = a[i] + b[i];
6   }
```

Vectors addiction OpenMP (Task+Target) example

« The nowait clause specifies that the generated task may be deferred »
OpenMP-API-Specification-5.2, page 308

Context

- **OpenMP task model**
  - Version 3.0 – 2008 introduced tasks programming
  - Version 4.0 – 2013
    - dependent tasks
    - GPU offloading through targets directive (not with tasks yet)
  - Version 4.5 – 2015 extended target for task programming
    - depend and nowait clause

```
1   #pragma omp target teams distribute parallel for \
2   map(to: a[0:N], b[0:N]) map(from: c[0:N]) \
3   depend(in: a[0], b[0]) depend(out: c[0]) nowait
4   for (i = 0; i < N, i++){
5       c[i] = a[i] + b[i];
6   }
```

Vectors addiction OpenMP (Task+Target) example

« The nowait clause specifies that the generated task may be deferred »
OpenMP-API-Specification-5.2, page 308

Problem

- **How to efficiently schedule CPU and GPU tasks over processing units?**
  - Automatic overlap of communication and computation with the GPU

Related works

- **LLVM – Hidden Helper Thread** [1]
  - Uses a team of kernel thread for scheduling target tasks
  - Limits : lack of flexibility on scheduling policy
    - Fixed team size (number of “target nowait” parallel limited)
    - Implicit kernel preemption scheduling decision made by the kernel

1 - Tian, S., Doerfert, J., Chapman, B.: Concurrent Execution of Deferred OpenMP Target Tasks with Hidden Helper Threads. In: Chapman, B., Moreira, J. (eds.) Languages and Compilers for Parallel Computing. pp. 41–56. Springer International Publishing, Cham (2022)

Related works

- **LLVM – Hidden Helper Thread** [1]
  - Uses a team of kernel thread for scheduling target tasks
  - Limits : lack of flexibility on scheduling policy
    - Fixed team size (number of “target nowait” parallel limited)
    - Implicit kernel preemption scheduling decision made by the kernel
- **Similar scheduling problem with MPI communication in tasks**
  - The impact of taskyield over MPI communication [2]
    - Characterizes the problem: in GOMP/Clang, blocking MPI call in a task retain CPU resources
    - No automatic overlap communication/calculation need for interoperability

2 - Schuchart, J., Tsugane, K., Gracia, J., Sato, M.: The Impact of Taskyield on the Design of Tasks Communicating Through MPI. In: de Supinski, B.R., Valero-Lara, P., Martorell, X., Mateo Bellido, S., Labarta, J. (eds.) Evolving OpenMP for Evolving Architectures. pp. 3–17. Springer International Publishing, Cham (2018)

Related works

- **LLVM – Hidden Helper Thread** [1]
  - Uses a team of kernel thread for scheduling target tasks
  - Limits : lack of flexibility on scheduling policy
    - Fixed team size (number of “target nowait” parallel limited)
    - Implicit kernel preemption scheduling decision made by the kernel
- **Similar scheduling problem with MPI communication in tasks**
  - The impact of taskyield over MPI communication [2]
    - Characterizes the problem: in GOMP/Clang, blocking MPI call in a task retain CPU resources
    - No automatic overlap communication/calculation need for interoperability
  - TAMPI[3] and MPC - added interoperability between MPI/OpenMP runtime

3 - Sala, K., Teruel, X., Pérez, J., Peña, A., Beltran, V., Labarta, J.: Integrating Blocking and Non-Blocking MPI Primitives with Task-Based Programming Models. Parallel Computing 85, 153–166 (07 2019). https://doi.org/10.1016/j.parco.2018.12.008

Related works

- **LLVM – Hidden Helper Thread** [1]
  - Uses a team of kernel thread for scheduling target tasks
  - Limits : lack of flexibility on scheduling policy
    - Fixed team size (number of “target nowait” parallel limited)
    - Implicit kernel preemption scheduling decision made by the kernel
- **Similar scheduling problem with MPI communication in tasks**
  - The impact of taskyield over MPI communication [2]
    - Characterizes the problem: in GOMP/Clang, blocking MPI call in a task retain CPU resources
    - No automatic overlap communication/calculation need for interoperability
  - TAMPI[3] and MPC - added interoperability between MPI/OpenMP runtime
    - In particular we are interested in MPC
      - Standard implementation of OpenMP 3.0 [4]
      - Full support for all tasks except targets
      - The MPC-OpenMP runtime adds "light context (fiber)" to tasks [5]
      - Managing user-space scheduling with task-switches rather than kernel preemptions

4 - Clet-Ortega, J., Carribault, P., Pérache, M.: Evaluation of OpenMP task scheduling algorithms for large NUMA architectures, Proc. Eur. Conf. Parallel Process., pp. 596-607, 2014

5 - Pereira, R., Roussel, A., Carribault, P., Gautier, T.: Communication-Aware Task Scheduling Strategy in Hybrid MPI+OpenMP Applications. In: McIntosh-Smith,S., de Supinski, B.R., Klinkenberg, J. (eds.) OpenMP: Enabling Massive Node-Level Parallelism. pp. 197–210. Springer International Publishing, Cham (2021)

Related works

- **LLVM – Hidden Helper Thread** [1]
  - Uses a team of kernel thread for scheduling target tasks
  - Limits : lack of flexibility on scheduling policy
    - Fixed team size (number of “target nowait” parallel limited)
    - Implicit kernel preemption scheduling decision made by the kernel
- **Similar scheduling problem with MPI communication in tasks**
  - The impact of taskyield over MPI communication [2]
    - Characterizes the problem: in GOMP/Clang, blocking MPI call in a task retain CPU resources
    - No automatic overlap communication/calculation need for interoperability
  - TAMPI[3] and MPC - added interoperability between MPI/OpenMP runtime
    - In particular we are interested in MPC
      - Standard implementation of OpenMP 3.0 [4]
      - Full support for all tasks except targets
      - The MPC-OpenMP runtime adds "light context (fiber)" to tasks [5]
      - Managing user-space scheduling with task-switches rather than kernel preemptions

Contribution

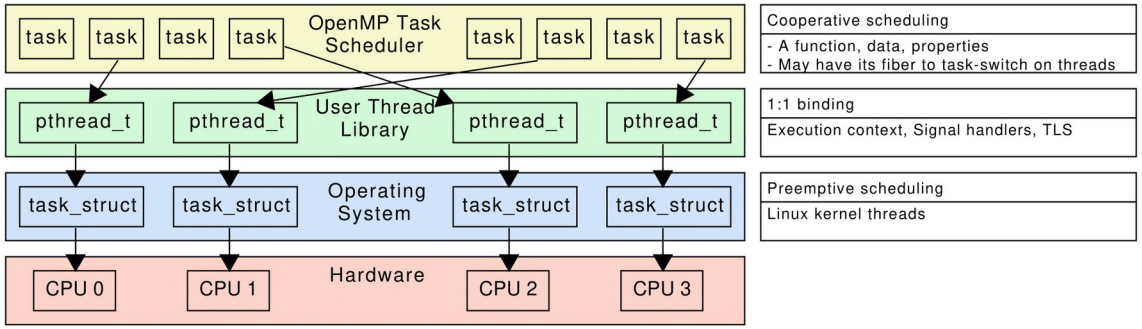
- **Addition of GPU task support with the target-nowait directives in MPC-OpenMP**
- **Enhance asynchronous executions through user-space task scheduling**

Contents

- Introduction
  - Context
  - Problem
  - Related works
  - Motivation
- Contribution
  - Cooperative target tasks design
  - Heterogeneity support in MPC
- Evaluation
  - State of art comparison
  - LULESH MPI+OpenMP(tasks+target)
- Conclusion

Target with asynchronous tasks

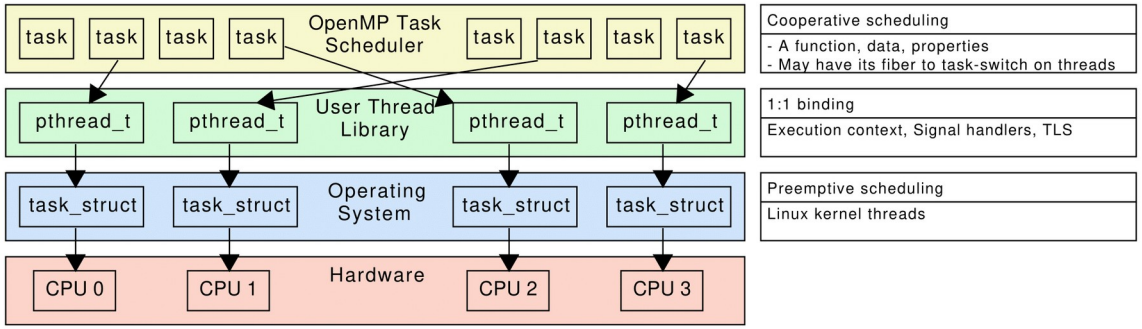
- **Cooperative target task design**
  - After asynchronous calls tasks are explicitly preempt before synchronization points without the operating system scheduler
  - Polling functions ensure asynchronous events progression



Target with asynchronous tasks

- **Cooperative target task design**
  - After asynchronous calls tasks are explicitly preempt before synchronization points without the operating system scheduler
  - Polling functions ensure asynchronous events progression

- **Fibers**
  - Lightweight user-space execution **context**
  - Tasks can pause itself at any time and may resume on any threads (support of untied tasks)

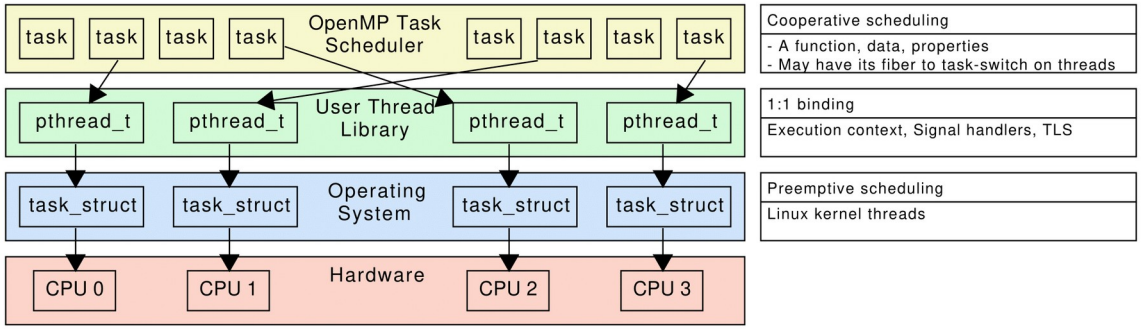


Target with asynchronous tasks

- **Cooperative target task design**
  - After asynchronous calls tasks are explicitly preempt before synchronization points without the operating system scheduler
  - Polling functions ensure asynchronous events progression

- **Fibers**
  - Lightweight user-space execution **context**
  - Tasks can pause itself at any time and may resume on any threads (support of untied tasks)

- **Overlapping of GPU operations with CPU computation**
  - Fix loss of core



Heterogeneity support in MPC

- **LLVM libomptarget**
  - The libomptartget is built as a module easy to export
  - MPC already implements the LLVM Application Binary Interface (ABI) except target ABI

Heterogeneity support in MPC

- **LLVM libomptarget**
  - The libomptartget is built as a module easy to export
  - MPC already implements the LLVM Application Binary Interface (ABI) except target ABI

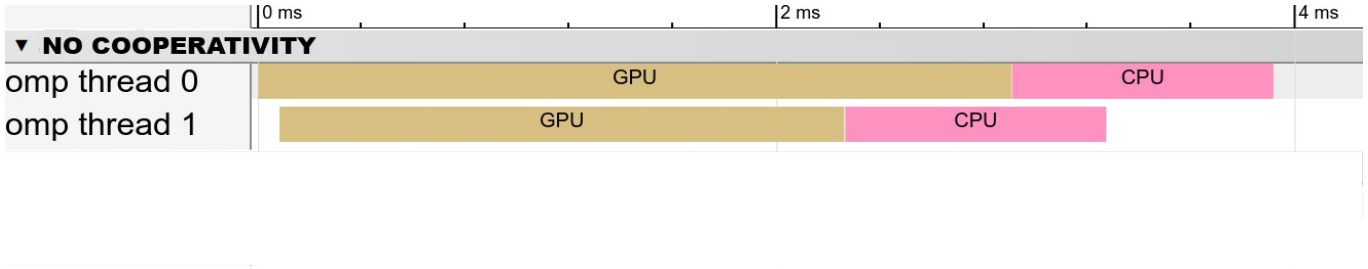
- **Integration of OpenMP Target in MPC**
  - Adding entry point in MPC-OpenMP ABI
    - __kmpc_get_target_offload
  - Implement new utilities OpenMP functions in MPC-OpenMP API
    - omp_get_default_device and omp_is_initial_device
  - Adding in the ABI target directives as tasks with dependencies
    - __kmpc_omp_target_task_alloc

Heterogeneity support in MPC

- **LLVM libomptarget**
  - The libomptartget is built as a module easy to export
  - MPC already implements the LLVM Application Binary Interface (ABI) except target ABI

- **Integration of OpenMP Target in MPC**
  - Adding entry point in MPC-OpenMP ABI
    - __kmpc_get_target_offload
  - Implement new utilities OpenMP functions in MPC-OpenMP API
    - omp_get_default_device and omp_is_initial_device
  - Adding in the ABI target directives as tasks with dependencies
    - __kmpc_omp_target_task_alloc

- **Enabling Asynchronism through Cooperativity**
  - In practice physical core is retain by a CUDA stream synchronization

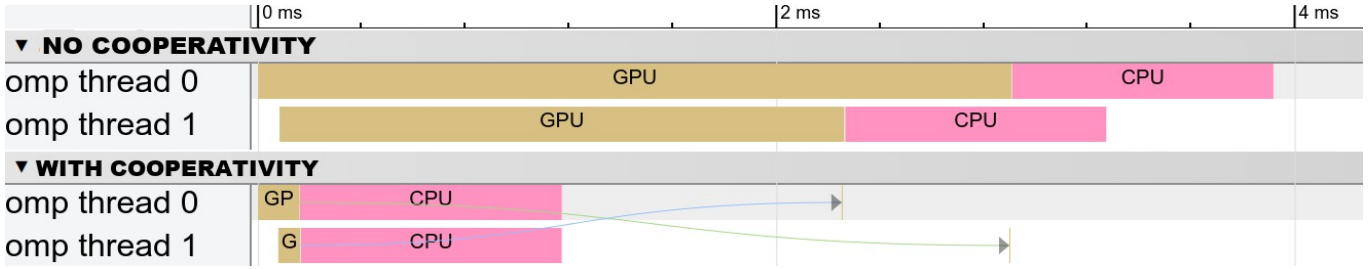


Heterogeneity support in MPC

- **LLVM libomptarget**
  - The libomptartget is built as a module easy to export
  - MPC already implements the LLVM Application Binary Interface (ABI) except target ABI

- **Integration of OpenMP Target in MPC**
  - Adding entry point in MPC-OpenMP ABI
    - __kmpc_get_target_offload
  - Implement new utilities OpenMP functions in MPC-OpenMP API
    - omp_get_default_device and omp_is_initial_device
  - Adding in the ABI target directives as tasks with dependencies
    - __kmpc_omp_target_task_alloc

- **Enabling Asynchronism through Cooperativity**
  - In practice physical core is retain by a CUDA stream synchronization
  - Adding fibers to MPC-OpenMP target task
  - Patch LLVM libomptarget CUDA RTL
    - Call a polling into MPC relying on cudaStreamQuery



Execution environment

- **Runtime environment**
  - Inti compute node hosted at CEA
    - 2 x AMD EPYC 7H12 64-core processors (4 NUMA nodes) @2.6GHz
    - 4 x NVIDIA A100 GPUs 40GB
  - The process run on a NUMA domain with 16 cores, 32GB of memory and one GPU

Execution environment

Runtime environment

- Inti compute node hosted at CEA
  - 2 x AMD EPYC 7H12 64-core processors (4 NUMA nodes) @2.6GHz
  - 4 x NVIDIA A100 GPUs 40GB
- The process run on a NUMA domain with 16 cores, 32GB of memory and one GPU

Software environment

- LLVM 14.x
  - number of threads = number HHT
- NVIDIA/PGI 22.2
- MPC-OpenMP with target support
  - Patched LLVM 14.x libomptarget
  - Open MPI 4.0.5
- Optimization enable

State-of-the-Art comparison

- **Microbenchmark**
  - DAXPY OpenMP (tasks+target)
    - Vector size T*n
      - T = tasks number (64)
      - n = size of a task
    - T triplets of tasks
      - data upload, daxpy, data download

```
void B5(double a, double * x, double * y, int T, int n) {
    # pragma omp parallel
    # pragma omp single
    for (int t = 0 ; t < T ; ++t) {
        # pragma omp target update to(y[t*n:n]) nowait depend(inout: y[t*n])

        # pragma omp target teams distribute parallel for nowait
            depend(inout: y[t*n])
        daxpy(a, x, y, t*n, n);

        # pragma omp target update from(y[t*n:n]) nowait depend(inout:
            y[t*n])
    }
}
```

State-of-the-Art comparison

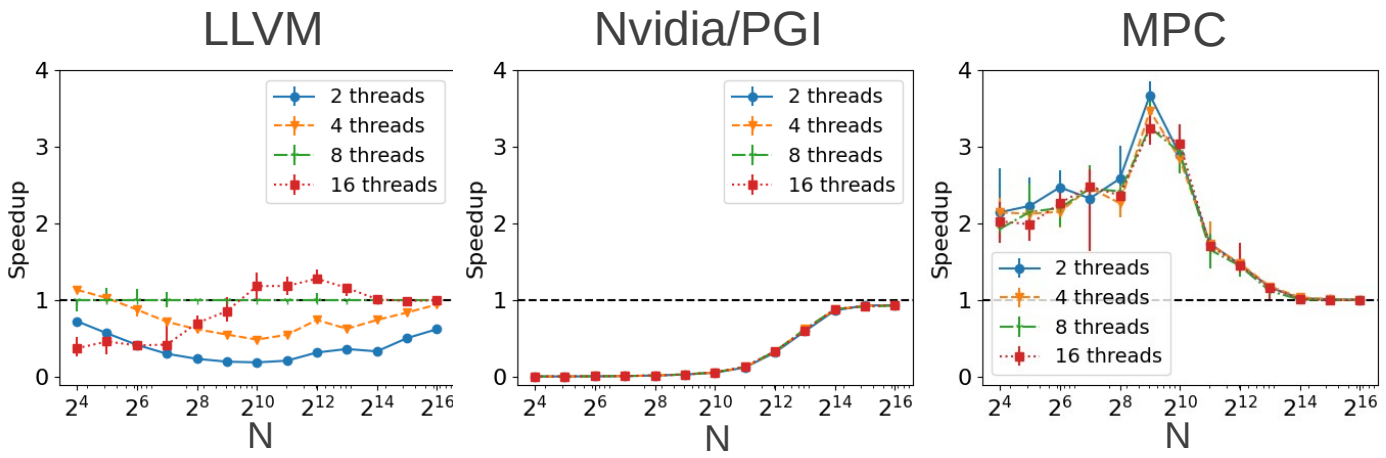
- **Microbenchmark**
  - DAXPY OpenMP (tasks+target)
    - Vector size T*n
      - T = tasks number (64)
      - n = size of a task
    - T triplets of tasks
      - data upload, daxpy, data download

```
void B5(double a, double * x, double * y, int T, int n) {
    # pragma omp parallel
    # pragma omp single
    for (int t = 0 ; t < T ; ++t) {
        # pragma omp target update to(y[t*n:n]) nowait depend(inout: y[t*n])

        # pragma omp target teams distribute parallel for nowait
            depend(inout: y[t*n])
        daxpy(a, x, y, t*n, n);

        # pragma omp target update from(y[t*n:n]) nowait depend(inout:
            y[t*n])
    }
}
```

- **Results**
  - Median of 5 runs
  - Comparison to LLVM default (8 HHT)
  - Similar target operation and execution time ⇒ difference came from task scheduling



OpenMP speedup compared to LLVM using 8 Hidden Helper Threads 64 tasks

State-of-the-Art comparison

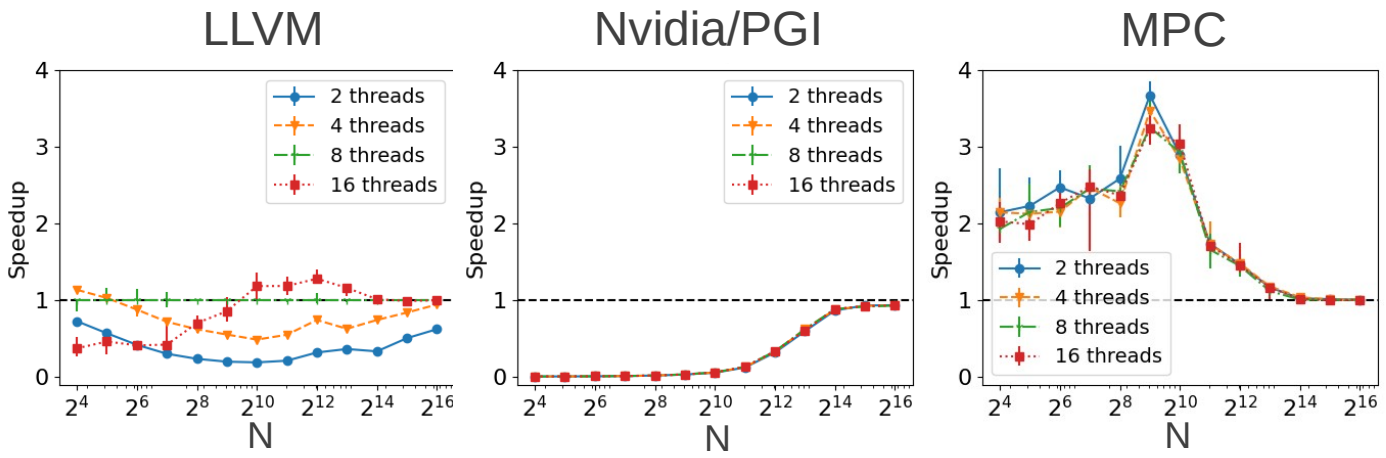
- **Microbenchmark**
  - DAXPY OpenMP (tasks+target)
    - Vector size T*n
      - T = tasks number (64)
      - n = size of a task
    - T triplets of tasks
      - data upload, daxpy, data download

```
void B5(double a, double * x, double * y, int T, int n) {
    # pragma omp parallel
    # pragma omp single
    for (int t = 0 ; t < T ; ++t) {
        # pragma omp target update to(y[t*n:n]) nowait depend(inout: y[t*n])

        # pragma omp target teams distribute parallel for nowait
            depend(inout: y[t*n])
        daxpy(a, x, y, t*n, n);

        # pragma omp target update from(y[t*n:n]) nowait depend(inout:
            y[t*n])
    }
}
```

- **Results**
  - Median of 5 runs
  - Comparison to LLVM default (8 HHT)
  - Similar target operation and execution time difference came from task scheduling
  - LLVM 14.x
    - Default setting (8 HHT) reasonable average performance



OpenMP speedup compared to LLVM using 8 Hidden Helper Threads 64 tasks

State-of-the-Art comparison

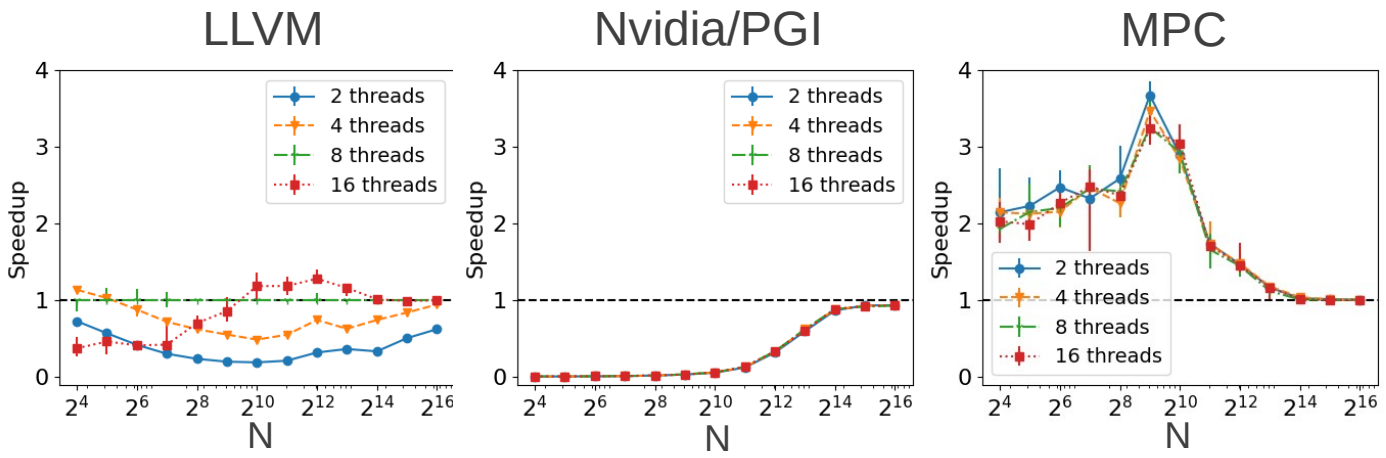
- **Microbenchmark**
  - DAXPY OpenMP (tasks+target)
    - Vector size T*n
      - T = tasks number (64)
      - n = size of a task
    - T triplets of tasks
      - data upload, daxpy, data download

```
void B5(double a, double * x, double * y, int T, int n) {
    # pragma omp parallel
    # pragma omp single
    for (int t = 0 ; t < T ; ++t) {
        # pragma omp target update to(y[t*n:n]) nowait depend(inout: y[t*n])

        # pragma omp target teams distribute parallel for nowait
            depend(inout: y[t*n])
        daxpy(a, x, y, t*n, n);

        # pragma omp target update from(y[t*n:n]) nowait depend(inout:
            y[t*n])
    }
}
```

- **Results**
  - Median of 5 runs
  - Comparison to LLVM default (8 HHT)
  - Similar target operation and execution time difference came from task scheduling
  - LLVM 14.x
    - Default setting (8 HHT) reasonable average performance
  - NVIDIA/PGI 22.2
    - Low performance
    - Number of threads does not impact performances



OpenMP speedup compared to LLVM using 8 Hidden Helper Threads 64 tasks

State-of-the-Art comparison

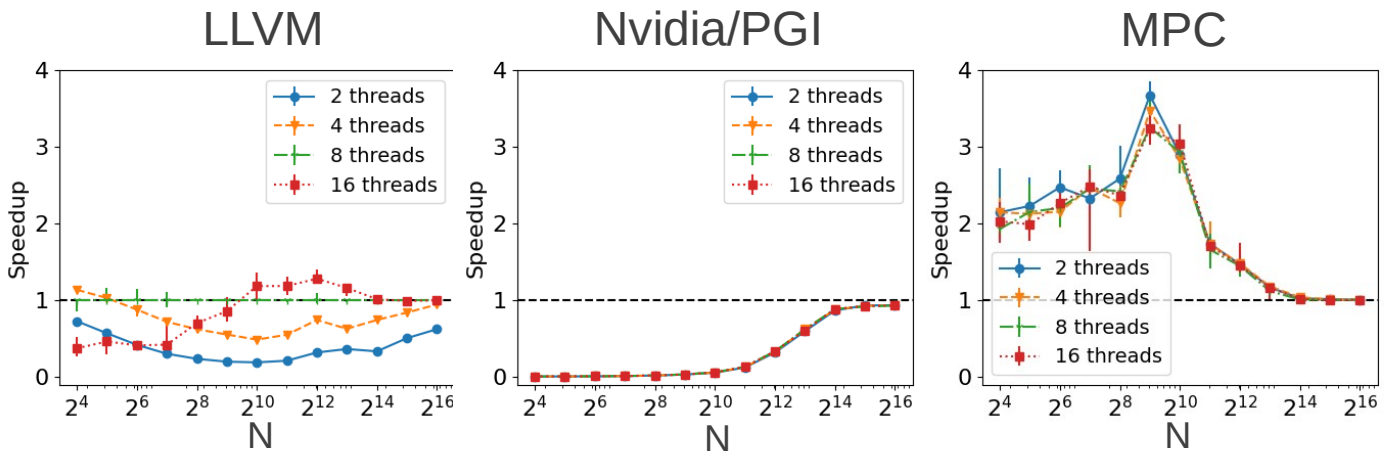
- **Microbenchmark**
  - DAXPY OpenMP (tasks+target)
    - Vector size T*n
      - T = tasks number (64)
      - n = size of a task
    - T triplets of tasks
      - data upload, daxpy, data download

```
void B5(double a, double * x, double * y, int T, int n) {
    # pragma omp parallel
    # pragma omp single
    for (int t = 0 ; t < T ; ++t) {
        # pragma omp target update to(y[t*n:n]) nowait depend(inout: y[t*n])

        # pragma omp target teams distribute parallel for nowait
            depend(inout: y[t*n])
        daxpy(a, x, y, t*n, n);

        # pragma omp target update from(y[t*n:n]) nowait depend(inout:
            y[t*n])
    }
}
```

- **Results**
  - Median of 5 runs
  - Comparison to LLVM default (8 HHT)
  - Similar target operation and execution time difference came from task scheduling
  - LLVM 14.x
    - Default setting (8 HHT) reasonable average performance
  - NVIDIA/PGI 22.2
    - Low performance
    - Number of threads does not impact performances
  - MPC-OpenMP with target support
    - Improve performances on fine-grain offloading
    - Small performance variation with the numbe



OpenMP speedup compared to LLVM using 8 Hidden Helper Threads 64 tasks

LULESH

- Simulation of hydrodynamic problems, describes the motion of materials subjected to forces

LULESH

- Simulation of hydrodynamic problems, describes the motion of materials subjected to forces

Original Parallel for application to MPI+OpenMP tasks

- Loops were transformed to tasks generating loops with dependencies
- Single-producer/multi-consumer scheme
- MPI communications finely nested within OpenMP tasks

LULESH

- Simulation of hydrodynamic problems, describes the motion of materials subjected to forces

Original Parallel for application to MPI+OpenMP tasks

- Loops were transformed to tasks generating loops with dependencies
- Single-producer/multi-consumer scheme
- MPI communications finely nested within OpenMP tasks

Porting MPI + OpenMP Task version to MPI + OpenMP (Task + Target)

- GPU tasks “super tasks” are composed of several CPU tasks
- Offload parts
  - First loop of IntegrateStressForElems and CalcKinematicsForElems
  - Overlap with CalcAccelerationForNodes
- Use of **cudaMallocHost** to allocate memory
  - Pinned memory for asynchronous execution

LULESH

- Simulation of hydrodynamic problems, describes the motion of materials subjected to forces

Original Parallel for application to MPI+OpenMP tasks

- Loops were transformed to tasks generating loops with dependencies
- Single-producer/multi-consumer scheme
- MPI communications finely nested within OpenMP tasks

Porting MPI + OpenMP Task version to MPI + OpenMP (Task + Target)

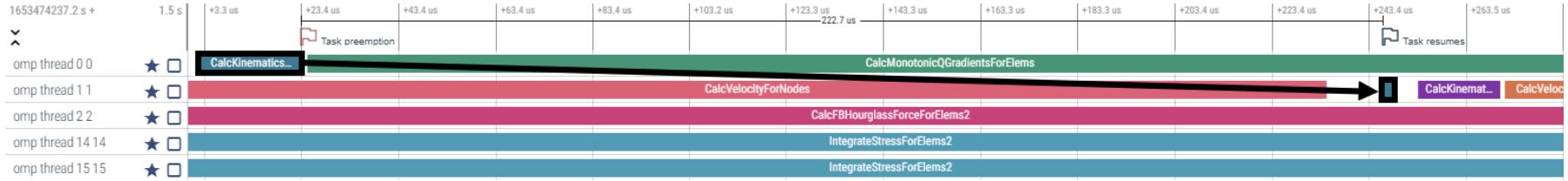
- GPU tasks “super tasks” are composed of several CPU tasks
- Offload parts
  - First loop of IntegrateStressForElems and CalcKinematicsForElems
  - Overlap with CalcAccelerationForNodes
- Use of **cudaMallocHost** to allocate memory
  - Pinned memory for asynchronous execution

Problem size

- 264^3 elements
  - 80% of the NUMA domain memory capacity
  - 20% of the GPU memory capacity
- 128 iterations

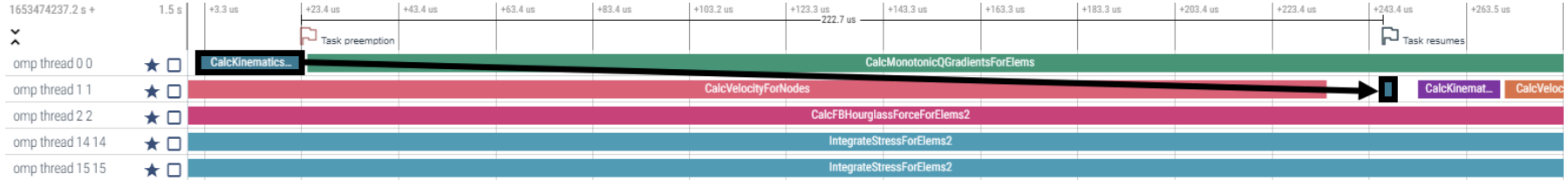
LULESH

- **Fully functional Hybrid and Heterogenous task-based version**
  - Automatic overlapping MPI communication and GPU operation

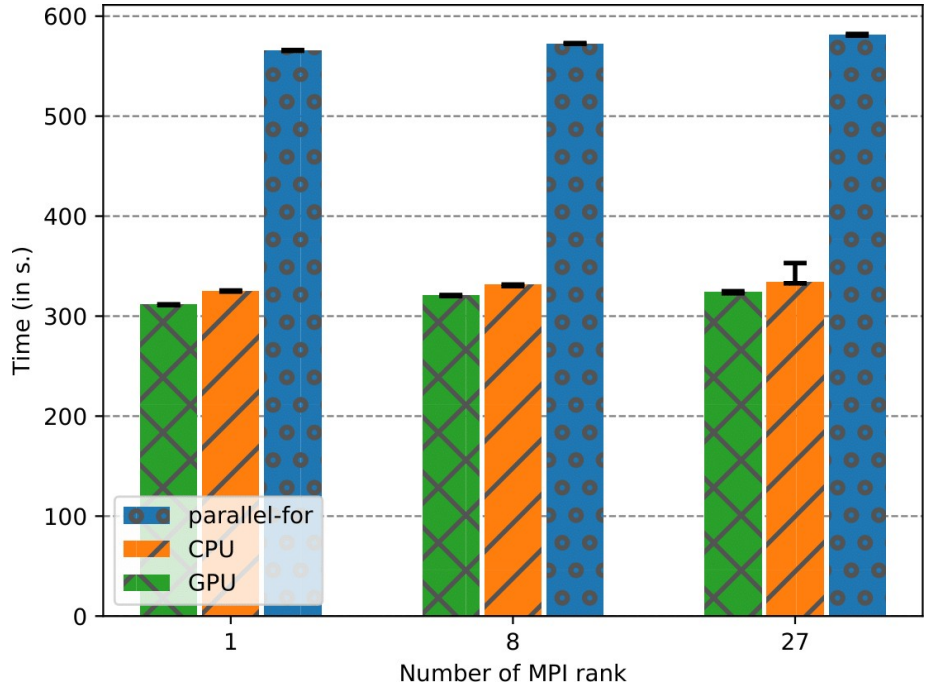


LULESH

- **Fully functional Hybrid and Heterogenous task-based version**
  - Automatic overlapping MPI communication and GPU operation

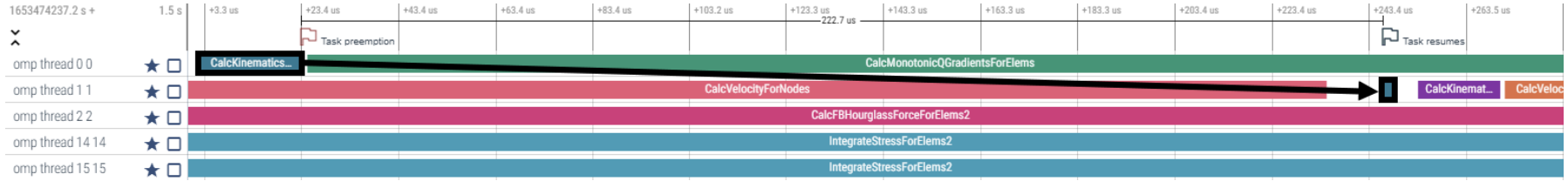


- **Weak scaling**
  - Median runtime of 10 runs
  - Each version scale very well

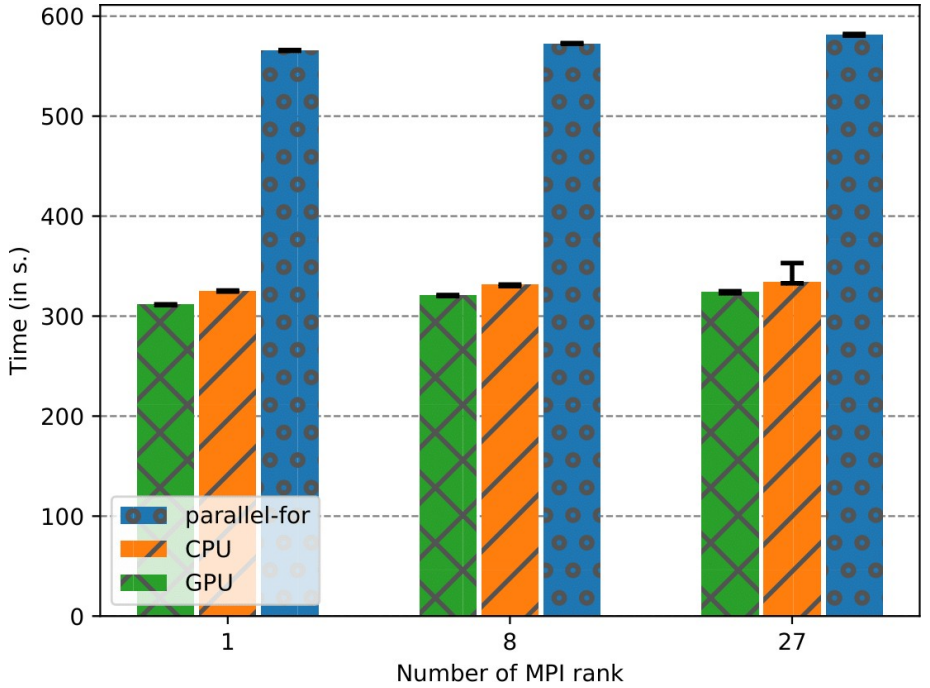


LULESH

- **Fully functional Hybrid and Heterogenous task-based version**
  - Automatic overlapping MPI communication and GPU operation



- **Weak scaling**
  - Median runtime of 10 runs
  - Each version scale very well
  - OpenMP tasks version significatively improve performances due to cache reuse
  - Slight performance gain with GPU offloading



Conclusion

- Unification of the CPU/GPU task-based programming model on a distributed machine

Conclusion

- Unification of the CPU/GPU task-based programming model on a distributed machine

- Asynchronism of OpenMP Target Task is not guaranteed by the standard

Conclusion

- **Unification of the CPU/GPU task-based programming model on a distributed machine**
- **Asynchronism of OpenMP Target Task is not guaranteed by the standard**
- **User-space cooperative target task design**
  - Overlapping of GPU operations with CPU computation

Conclusion

- **Unification of the CPU/GPU task-based programming model on a distributed machine**
- **Asynchronism of OpenMP Target Task is not guaranteed by the standard**
- **User-space cooperative target task design**
  - Overlapping of GPU operations with CPU computation
- **Add heterogeneity support into the MPC framework**
  - Integration of the LLVM libomptarget in MPC
  - Only NVIDIA GPUs are supported

Conclusion

- **Unification of the CPU/GPU task-based programming model on a distributed machine**
- **Asynchronism of OpenMP Target Task is not guaranteed by the standard**
- **User-space cooperative target task design**
  - Overlapping of GPU operations with CPU computation
- **Add heterogeneity support into the MPC framework**
  - Integration of the LLVM libomptarget in MPC
  - Only NVIDIA GPUs are supported
- **Future works**
  - Increase the computing load on GPU
    - Offload more loops
  - Develop more applications MPI+OpenMP (Task+Target)
  - Support GPUs from other vendors

Thanks for your attention

DE LA RECHERCHE À L'INDUSTRIE

Enhancing MPI + OpenMP Task based Applications for Heterogenous Architectures with GPU support

Laboratoire d'informatique en calcul intensif et image pour la simulation
Commissariat à l'énergie atomique et aux énergies alternatives – www.cea.fr
Institut national de recherche en informatique et en automatique – www.inria.fr