OpenMP tasking:
Extensions and optimizations for performance, predictability and resilience

Sara Royuela
sara.royuela@bsc.es

OpenMP User’s Monthly Telecon
October 28, 2022
Predictable Parallel Computing in OpenMP

Dr. Eduardo Quiñones
Team leader

Chenle Yu
PhD candidate

Adrian Munera
PhD candidate

Dr. Sara Royuela
Senior Researcher

www.ampere-euproject.eu
PPC in the scope of OpenMP

Specification, compiler and runtime support in OpenMP targeting performance, predictability and resilience in multiple domains

- NVIDIA Jetson
- Kalray MPPA
- NVIDIA A100
- AMD EPYC

OpenMP

GCC

Embedded Computing

High-Performance Computing
Today

• The overhead of tasking

• The Task Dependency Graph
  – Performance
  – Memory consumption
  – Interoperability with CUDA graphs

• The RISING Stars and the AMPERE project
**Motivation for tasking:** focus on the exposing parallelism rather than figuring out how to fit in a specific machine.

**Real limitations:** fine granularities and deep cut-offs introduce too much overhead, reducing potential speed up.

Podobas, A., Brorsson, M., and Faxén, K. F.

In *3rd workshop on programmability issues for multi-core computers.*

*A comparison of some recent task-based parallel programming models.* 2010.

---

**OpenMP 3.1**

<table>
<thead>
<tr>
<th>Multisort</th>
<th>Strassen</th>
</tr>
</thead>
<tbody>
<tr>
<td>coarse</td>
<td>coarse</td>
</tr>
<tr>
<td>fine</td>
<td>fine</td>
</tr>
</tbody>
</table>

![Graphs showing speedup vs. number of threads for Multisort and Strassen algorithms using OpenMP 3.1.](image-url)

Motivation for tasking: implement offloading capabilities to exploit accelerator devices.

Real limitations: code transformation might not be optimized, e.g., memory allocations, synchronizations.
Podobas, A., and Karlsson, S.  
In *International Workshop on OpenMP.*  
Towards unifying OpenMP under the task-parallel paradigm. 2016.

**Motivation for tasking:** taskloop could eliminate the need for thread-parallelism.  

**Real limitations:** the implementation is crucial and determining granularity becomes a challenge.
Motivation for tasking: many different extensions that have changed the internals of the implementations.

Real limitations: fine granularity provides poor efficiency 17% - 77%; acceptable granularity from tens of ms.
Where are we

- Tasking is convenient to expose parallelism
- Implementation overheads limit its use for:
  - Fine grained parallelism
  - Loop parallelism
  - Accelerator devices

Let’s capitalize on the Task Dependency Graph!
A region of code that can be fully represented as a TDG:

1. **Taskified** region:
   a) All computations are enclosed in tasks.
   b) Non-taskified code has no side effects on the tasks (e.g., induction variables in loops).

2. TDG shape:
   a) Shape does not change across TDG executions.
   b) Provide information for recomputing the TDG (e.g., a clause with the variables shaping the TDG).

Pedestrian Detector

```c
#pragma omp parallel
#pragma omp single
{
    for (i=0; i<N_ITER; ++i) {
        for (by=0; by < BY; by+=BS) {
            for (bx=0; bx < BX; bx+=BS) {
                if (bx==0 && by==0) {
                    #pragma omp task depend(…)
                } else if (by==0) {
                    #pragma omp task depend(…)
                } else if (bx==0) {
                    #pragma omp task depend(…)
                } else {
                    #pragma omp task depend(…)
                }
            }
        }
    }
}
```
TDG-driven framework

- **Goals:** Reduce overhead due to task orchestration and dependency resolution
- **Methodology:** Eliminate the execution of user code to instantiate tasks

**Source code**

```c
for (int it=0; it<IT; ++it)
#pragma omp taskgraph
for (int i=0; i<N; ++i){
    #pragma omp task
    {...}
    ...
}
```

**Compiler**

- **data known**
- **data unknown**

**Runtime**

- **Iteration 1**
- **Iteration N**

**How to define it:**
- User-defined: `taskgraph` directive
- Automatically detected by the compiler:
  - Via analysis
  - `taskloop`

**When to generate it:**
- At compile-time (CRTES)
- At run-time (HPC)

Compiler transformations

1. Meet conditions / taskgraph region
   - 1. Meet conditions / taskgraph region
   - No
     - No taskgraph
     - TDG generated at run-time
       - struct kmp_record_info
         kmp_tdg_1[N_inst] = {...}
       - for (int it=0; it<IT; ++it)
         execute_TDG();
       - if (first_time)
         record_TDG();
         else
         execute_TDG();
       - for (int it=0; it<IT; ++it)
         instantiate_task();
         ...
       - struct kmp_record_info
         kmp_tdg_1[N_inst] = {...}
   - Yes
     - TDG generated at compile-time
       - struct kmp_record_info
         kmp_tdg_1[N_inst] = {...}
       - for (int it=0; it<IT; ++it)
         execute_TDG();
       - }
One thread pushes to a single queue, from which all threads pull.

Vanilla
One thread pushes to its queue and the rest steal work from it.

TDG
Each thread pushes and pulls from its own queue. Work-stealing is allowed.
Can we reduce overhead?

Synthetic

```c
#pragma omp parallel
#pragma omp single
{
    for (int i=0; i<N_Tasks; ++i) {
        int index = I % N_Cores;
        #pragma omp task depend(out:deps[index])
        fn();
    }
}
```

**Task orchestration overhead**

\[
\text{Computation} = \frac{\text{serial_time}}{\#\text{threads}}
\]

\[
\text{Overhead} = \text{Total_time} - \text{Computation}
\]

A = 1 task of $10^9$ inst., B = 10 tasks of $10^8$ inst., C = $10^7$ tasks of $10^7$ inst., D = $10^3$ tasks of $10^6$ inst., E = $10^4$ tasks of $10^5$ inst., F = $10^5$ tasks of $10^4$ inst.

 ✓ Reduce #instructions needed to orchestrate tasks
 ✓ Alleviate contention

<table>
<thead>
<tr>
<th>#tasks</th>
<th>$10^0$</th>
<th>$10^1$</th>
<th>$10^2$</th>
<th>$10^3$</th>
<th>$10^4$</th>
<th>$10^5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>1.6</td>
<td>0.6</td>
<td>0.6</td>
<td>1.5</td>
<td>36.7</td>
<td>466.2</td>
</tr>
<tr>
<td>Taskgraph</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>9.9</td>
<td>132.2</td>
</tr>
</tbody>
</table>
Scalability

Speed-up of *TDG-driven* execution compared to vanilla *task* and *taskloop* implementations using different number of threads and different task granularities

→ Negligible negative impact
→ Considerable positive impact for fine granularities and high thread contention

Speed-up of *TDG-driven* execution compared to vanilla GCC and LLVM implementations using different task granularities

→ Coarse grained tasks provide comparable results
→ Fine grained tasks show certain stability
→ Benefits are portable across compilers/RTLs
Memory management

- **Goals:**
  1. Avoid dynamic allocation of task structures
  2. Reduce and bound the memory requirements of the OpenMP RTL

- **Methodology:**
  1. Compiler: static generation of the required task structures
  2. Runtime: *lazy task creation* (task created when dependencies fulfilled)

## Space-Time Adaptive Processing

**Performance speedup**

**Use of dynamic memory**

**Memory consumption**

---

Applicability

- Pedestrian detector (automotive)
- Infra-red sensor processing (space)
- 3D Path Planning (avionics)
- Cholesky Factorization (HPC)

Many applications can be represented as a TDG!
The RISING stars project

- Enable a versatile and efficient **data acquisition** providing interoperability between different programming models (OpenMP, CUDA)
- Expose **data acquisition/transfer** in the **programming model**
- Introduce **real-time** oriented features in the **programming model** to define periodicity, preemption, migration, and allocation.
- Use cases: Adaptive optics, adaptive beamforming, the Square Kilometer Array and Space Situational Awareness.
for (k=0; k<NB; ++k) {
    #pragma omp target depend(inout: Ah[k][k])
    potrf(Ah[k][k], ts, ts);
    for (i=k+1; i<NB; ++i) {
        #pragma omp task depend (in: Ah[k][k]) \ 
            depend(inout: Ah[k][i])
        trsm(Ah[k][k], Ah[k][i], ts, ts);
    }
    for (l=k+1; l<NB; ++l) {
        for (j=k+1; j<l; ++j) {
            #pragma omp task depend(in: Ah[k][l]) \ 
            depend(in: Ah[k][j]) \ 
            depend(inout: Ah[j][l])
            gemm(Ah[k][l], Ah[k][j], Ah[j][l], ts, ts);
        }
        #pragma omp task depend(in: Ah[k][l]) \ 
            depend(inout: Ah[l][l])
        syrk(Ah[k][l], Ah[l][l], ts, ts);
    }
}
## CUDA memory management strategies

**Execution time in ms.**

<table>
<thead>
<tr>
<th>App</th>
<th>Graph nodes</th>
<th>Unified memory + prefetch</th>
<th>Unified memory (non-prefetch)</th>
<th>Zero copy</th>
<th>cudaMemcpy</th>
<th>cudaMemcpyAsync</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector addition</td>
<td>1024</td>
<td>739</td>
<td>865</td>
<td>687</td>
<td>448</td>
<td>435</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>592</td>
<td>738</td>
<td>595</td>
<td>344</td>
<td>343</td>
</tr>
<tr>
<td>Saxpy</td>
<td>1024</td>
<td>658</td>
<td>916</td>
<td>538</td>
<td>441</td>
<td>441</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>459</td>
<td>572</td>
<td>452</td>
<td>286</td>
<td>255</td>
</tr>
<tr>
<td>Nbody</td>
<td>1024</td>
<td>6080</td>
<td>6058</td>
<td>6464</td>
<td>6041</td>
<td>6094</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>762</td>
<td>775</td>
<td>806</td>
<td>764</td>
<td>765</td>
</tr>
</tbody>
</table>

- **Faster when there is no need for unified memory**
- **Faster execution when reducing the number of nodes**
- **Pre-fetching reduces page faults**
The AMPERE project

- Use of **Domain Specific Modeling Languages** and **high-level synthesis methods** for building **correct-by-construction** systems.

- Use OpenMP to provide the **performance** needed to develop complex Cyber-Physical Systems:
  - Predictive Cruise Control (automotive)
  - Obstacle Detection and Avoidance System (railway)

- Provide mechanisms to guarantee **non-functional requirements**: time predictability, resilience and energy consumption.
Goals:

1. Exploit parallelism within OS tasks with OpenMP (host and target) tasks
2. Exploit heterogeneity through specializations

void PeriodicTask() {
    #pragma omp parallel
    #pragma omp single
    {
        #pragma omp task depend(out:Image)
        { read_image(); }
        #pragma omp task depend(in:Image) \
        depend(out:ResultsA) cpu_omp
        { analysisA(); }
        #pragma omp target depend(in:Image) \
        depend(out:ResultsA) \
        map(to:Image) map(from:ResultsA)\n        gpu_omp
        { analysisB(); }
        #pragma omp task depend(in:ResultsA, ResultsB)
        { read_image(); }
    }
}

void analysisA_gpu() { ... }

#pragma omp declare variant(analysisA_gpu)\nmatch(construct={target})\nimplementation={extension(gpu_omp)}
void analysisA() { ...}

https://gitlab.bsc.es/ampere-sw/wp2
AMALTHEA DSML to OpenMP: Performance

NVIDIA Jetson TX2 board with a GPU and a 4-core ARM CPU
Correctness analysis for OpenMP

**Goals:**
1. Detect/resolve race conditions
2. Detect/correct wrong synchronizations (task dependencies, memory fences)
3. Detect inconsistencies in the data-sharing attributes

**LLVM Compiler pipeline to define/correct data-sharing attributes:**

**GOAL:** Assess predictability of OpenMP to allow schedulability analysis

**Requirements:**

1. Work-conserving scheduler for non-pessimistic WCRT analysis
2. Prescriptive task priorities to support fixed priority schedulers
3. Prescriptive implementation of Task Scheduling Points to allow limited preemptive scheduling
   - Taskyield, to alleviate pessimism and enhance schedulability

Response time upper bound ($R_{k}^{ub}$):

$$R_{k}^{ub} \leftarrow len(G_k) + \frac{1}{m} \left( vol(G_k) - len(G_k) \right) + \frac{1}{m} \left( I_{hp} + I_{lp} \right)$$

$R_{k}^{ub}$ bounds the maximum observed execution time

Real-time system schedulable with at least 16 cores

---


Augmented OpenMP code

```c
#pragma omp task redundancy(
    spatial|temporal|spatial_temporal, 3, /*n replicas*/
    var1:func1,
    var2:func2)
{ /*code*/}
```

**GOAL:** Task-level replication for fault-detection

Spatial: Each replica in a different core

Temporal: OpenMP mutexinoutset between replicas

Spatial: Each replica in a different core

**TDG with replication**

**Engine control management system**

- TDG1 (T10, T20, T50)
- TDG2 (T10, T50)
- TDG3 (T10, T20)
- TDG4 (T10)

Optimizations

Interoperability

Functional safety

Predictability and CRTES
Projects and collaborations

- AMPERE
  - www.ampere-euproject.eu
- Rising STARS
  - www.risingstars-project.eu
- P-SOCRATES
- ESA
- AIRBUS DEFENCE & SPACE
- DENSO
- BOSCH
- THALES
High-performance OpenMP tasking

Sara Royuela
sara.royuela@bsc.es

OpenMP User's Monthly Telecon
October 28, 2022