# A static scheduling approach to enable safety-critical OpenMP applications

Alessandra Melani, **Maria A. Serrano**, Marko Bertogna,
Isabella Cerutti, Eduardo Quiñones, Giorgio Buttazzo

ASP-DAC 2017
January 16-19, 2017
Chiba/Tokyo, Japan

SCUOLA SUPERIORE SANT'ANNA PISA
Barcelona Supercomputing Center
Centro Nacional de Supercomputación
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

# Motivation

- **There is an increasing demand of new safety-critical real-time applications providing high performance**
  - Timing guarantees are fundamental to be fulfilled

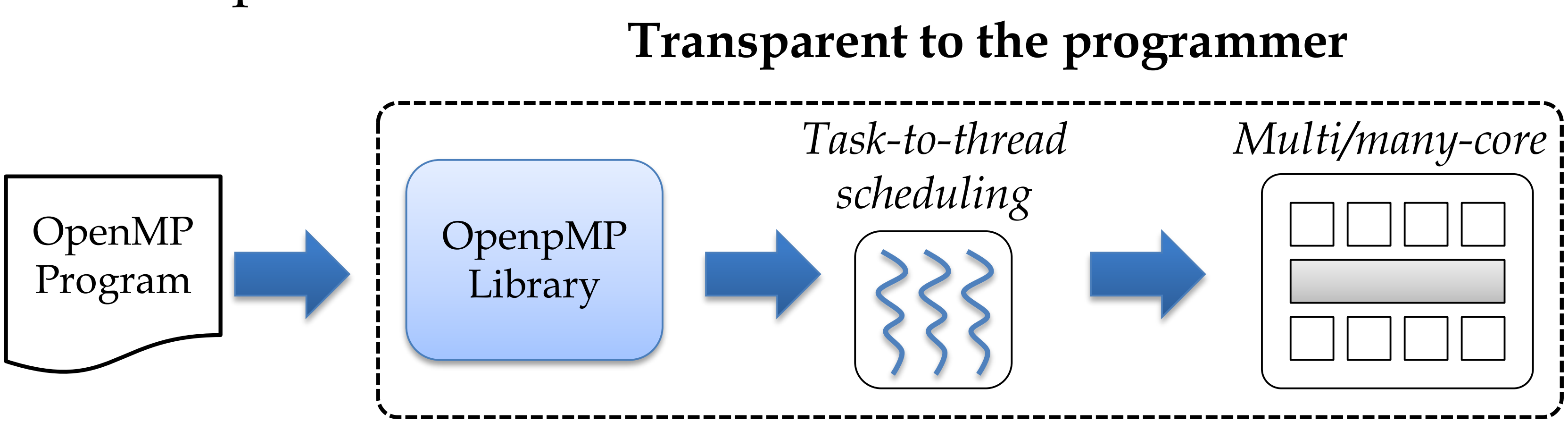
- **Performance demands can be satisfied by using advanced parallel architectures (multi/many-core)**

# Parallel programing models

- **Fundamental for exploiting the performance of multi- and many-cores**
  - Provide the level of abstraction to express parallel applications, while hiding processor complexities
  - Mandatory to exploit the massively parallel computation capabilities
- **OpenMP is one of the most used in HPC**
  - Increasingly adopted in embedded systems

# OpenMP

- **Supported by most of current many-core architectures**
- **Allows expressing fine-grained and unstructured parallelism**
  - Tasks
  - Dependencies

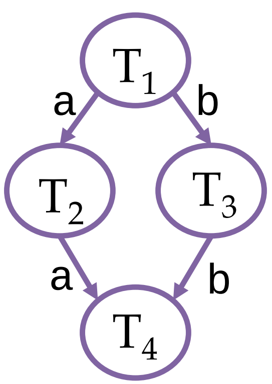


# Time predictable OpenMP

- **OpenMP tasking model**
  - Task Dependency Graph (TDG)

```
#pragma omp task depend(out:a,b)     // T1
{ … }
#pragma omp task depend(inout:a)     // T2
{ … }
#pragma omp task depend(inout:b)     // T3
{ … }
#pragma omp task depend(in:a,b)      // T4
{ … }
```

- **It resembles the Direct Acyclic Graph (DAG) real-time scheduling model**
  - Addresses the time predictability of real-time parallel computation



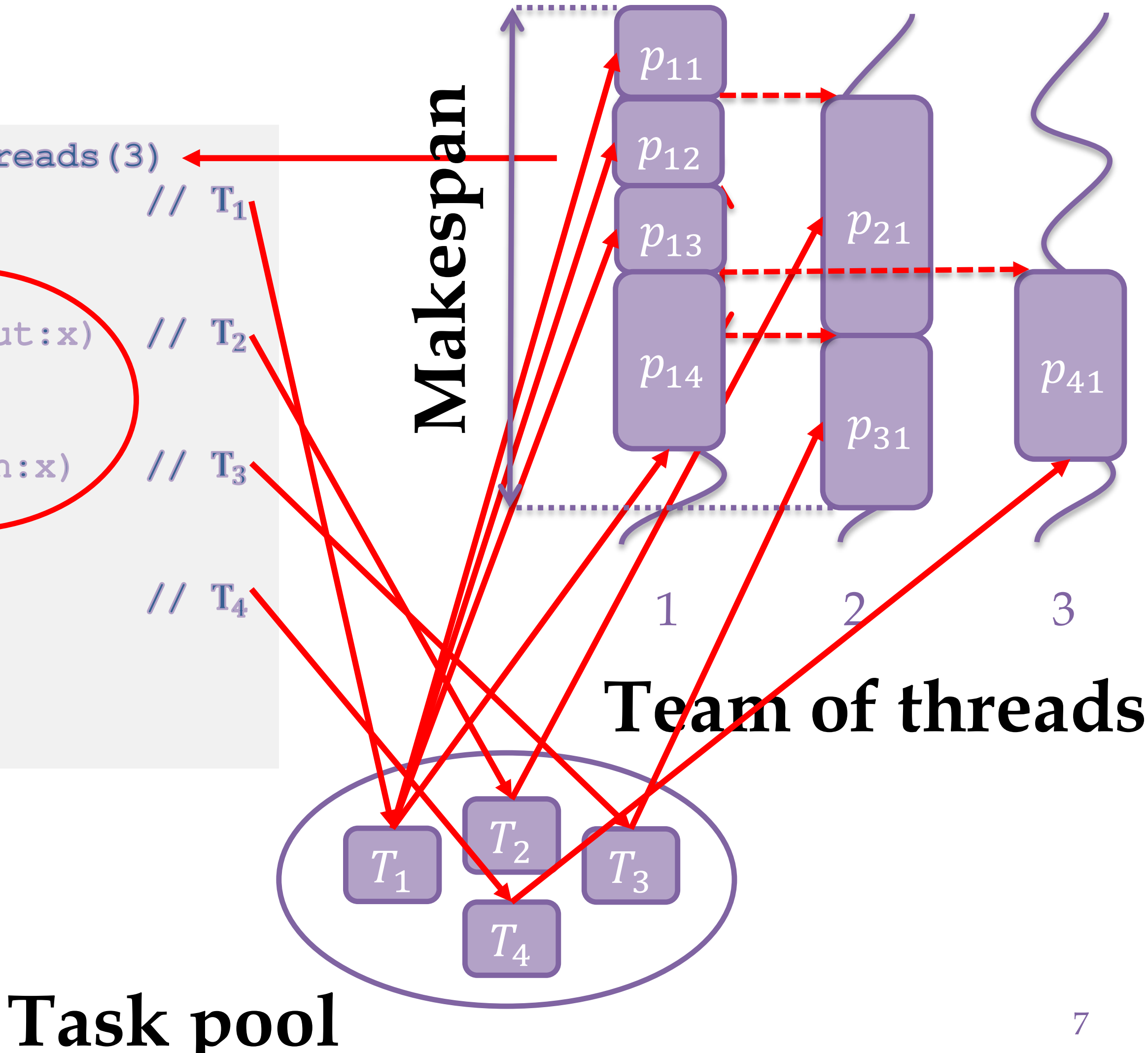
# OpenMP for safety-critical systems?

- **Current OpenMP implementations rely on dynamic scheduling approaches**
  - Allow schedulability analysis exploiting the work-conserving nature of scheduling [1]
  - Less suitable to safety-critical systems timing analysis
- **This work provides OpenMP-compliant static allocation strategies**
  - Allow a tighter timing analysis as it knows where each task executes
  - More suitable to safety-critical systems timing analysis

[1] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna and E. Quiñones, *"Timing characterization of OpenMP4 tasking model"*, in CASES, 2015.

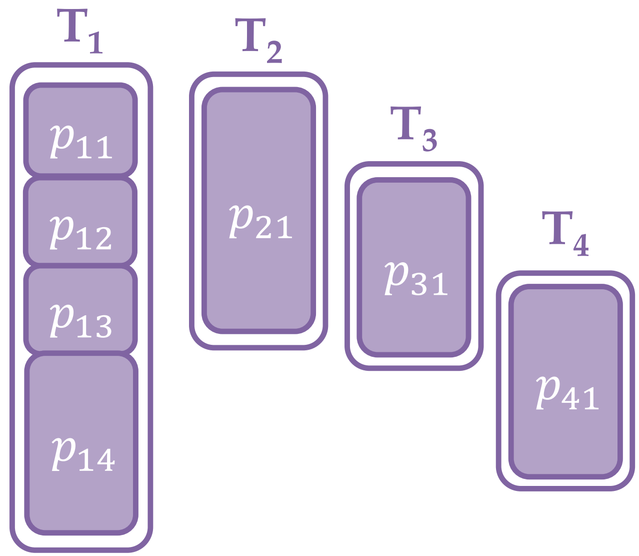
# OpenMP tasking model

```
#pragma omp parallel num_threads(3)
#pragma omp single                          // T1
{
  p11
  #pragma omp task depend(out:x)            // T2
     p21
  p12
  #pragma omp task depend(in:x)             // T3
     p31
  p13
  #pragma omp task                          // T4
     p41
  p14
}
```



# OpenMP tasking model

```
#pragma omp parallel num_threads(N)
#pragma omp single                        // T1
{
  p11
  #pragma omp task depend(out:x)          // T2
     p21
  p12
  #pragma omp task depend(in:x)           // T3
     p31
  p13
  #pragma omp task                        // T4
     p41
  p14
}
```




**Task-parts $p_{i,j}$**

- Represented by their WCET $C_{i,j}$

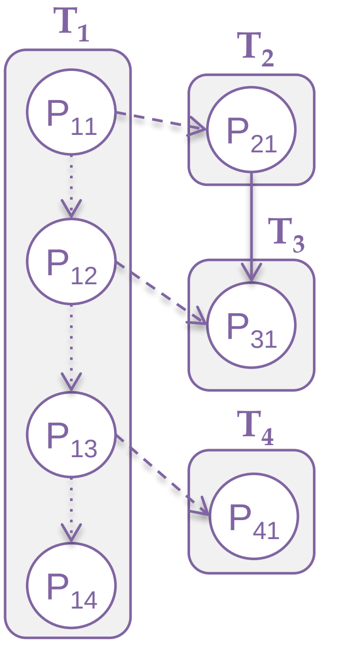
**Task Scheduling Points (TSPs)**

- Task may be suspended

# OpenMP tasking model

```
#pragma omp parallel num_threads(N)
#pragma omp single                      // T1
{
  p11
  #pragma omp task depend(out:x)        // T2
     p21
  p12
  #pragma omp task depend(in:x)         // T3
     p31
  p13
  #pragma omp task                      // T4
     p41
  p14
}
```

From an OpenMP program, an OpenMP-DAG can be derived [2]



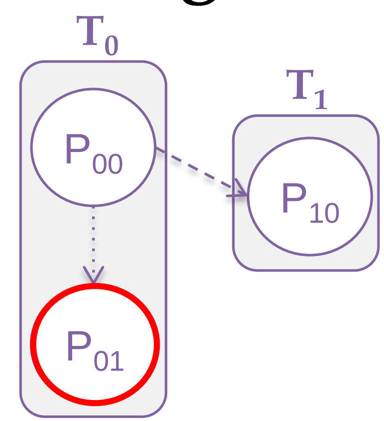


[2] R. Vargas, E. Quiñones and A. Marongiu. *"OpenMP and Timing Predictability: A Possible Union?"* In 18th Design, Automation and Test in Europe Conference (DATE), 2015.

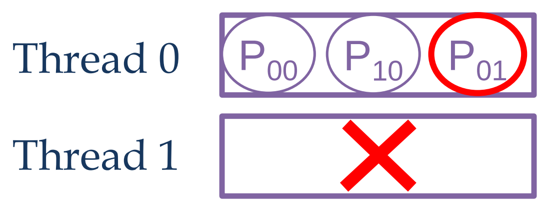

| OpenMP4 | DAG-based |
|---|---|
| Task parts | Nodes |
| Dependencies and TSPs | Edges |

# OpenMP tasking model

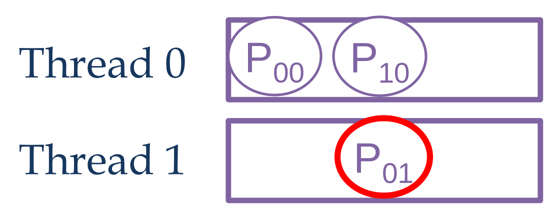
## Task classification that affects the scheduling




- **Tied tasks**
  - Must only be executed by the thread that started it




- **Untied tasks**
  - Can be resumed by any thread after being suspended



# OpenMP scheduling

- **Dynamic scheduling [1]**
  - Valid only for untied tasks

$$R^{ub} = len(G) + \frac{1}{m}(vol(G) - len(G)) \leq D$$

- **Our proposal: Static scheduling**
  - Valid for tied and untied tasks
  - Two approaches:
    - Optimal ILP based
    - Sub-optimal Heuristics-based

[1] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna and E. Quiñones, *"Timing characterization of OpenMP4 tasking model"*, in CASES, 2015.

# Strategy 1: Optimal static allocation

- **Problem definition: Optimally allocate OpenMP task-parts to threads**
  – Determine the minimum time interval needed to execute an OpenMP application on m threads
- **Solution**
  – ILP formulation for tied tasks
  – ILP formulation for untied tasks
- **Complexity**
  – NP-hard
  – Number of variables and constrains: $O(N^2p^2m)$

# Strategy 2: Sub-optimal static allocation

- **Heuristics (priority rules) to solve the makespan minimization problem [3,4]:**
  - Longest Processing Time (LPT)
  - Shortest Processing Time (SPT)
  - Largest Number of Successors in the Next Level (LNSNL)
  - Largest Number of Successors (LNS)
  - Largest Remaining Workload (LRW)

[3] M. L. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.
[4] K. E. Raheb, C. T. Kiranoudis, P. P. Repoussis, and C. D. Tarantilis, *"Production scheduling with complex precedence constraints in parallel machines"* Computing and Informatics, vol. 24, no. 3, 2012.

# Strategy 2: Sub-optimal static allocation

- **Tied tasks**
  - Input
    - `G`: OpenMP DAG
    - `m`: Num. threads
  - Output
    - μ: Makespan
    - Ψ: Task-parts starting times
    - θ: Task-to-thread mapping
  - `A`: Allocated task-parts
  - `R`: Ready task-parts
  - `L[1..m]`: Last idle time of each thread
  - `S[1..m]`: Tasks suspended on each thread

**procedure** $\textsc{HeurTied}(G, m)$
    $A \leftarrow \emptyset$; $R \leftarrow p_{1,1}$
    $L \leftarrow \textsc{Array}(m, 0)$; $S \leftarrow \textsc{Array}(m, \emptyset)$
    **while** $\textsc{Size}(A) \; != \sum_{i=1}^{N} n_i$ **do**
        $k \leftarrow \textsc{FirstIdleThread}(L)$
        $P_{i,j} \leftarrow \textsc{NextReadyJob}(k, R, S_k, G)$
        **if** $j == 1$ **then**
            $\theta_i \leftarrow k$
            **if** $j \; != n_i$ **then**
                $S_k \leftarrow \textsc{Append}(i, S_k)$
            **end if**
        **else if** $j == n_i$ **then**
            $S_k \leftarrow \textsc{Remove}(i, S_k)$
        **end if**
        $\psi_{i,j} = \max(L_{\theta_i}, \psi_{i,j})$; $L_{\theta_i} \leftarrow \psi_{i,j} + C_{i,j}$
        $A \leftarrow \textsc{Append}(P_{i,j}, A)$; $R \leftarrow \textsc{Remove}(P_{i,j}, R)$
        **for** $P_{k,z} \mid (P_{i,j}, P_{k,z}) \in E$ **do**
            **if** $\psi_{k,z} < \psi_{i,j} + C_{i,j}$ **then**
                $\psi_{k,z} \leftarrow \psi_{i,j} + C_{i,j}$;
            **end if**
            $F_{k,z} \leftarrow F_{k,z} + 1$
            **if** $F_{k,z} == \textsc{Size}(\textsc{InEdges}_{k,z})$ **then**
                $R \leftarrow \textsc{Append}(P_{k,z}, R)$
            **end if**
        **end for**
    **end while**
    $\mu = \max_{i=1}^{m} L_i$
    **return** $(\mu, \psi, \theta)$
**end procedure**

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**

Iterates until all task-parts have been allocated

**procedure** $\text{HeurTied}(G, m)$
$A \leftarrow \emptyset$; $R \leftarrow p_{1,1}$
$L \leftarrow \text{Array}(m, 0)$; $S \leftarrow \text{Array}(m, \emptyset)$
**while** $\text{Size}(A) \mathrel{!}= \sum_{i=1}^{N} n_i$ **do**
$k \leftarrow \text{FirstIdleThread}(L)$
$P_{i,j} \leftarrow \text{NextReadyJob}(k, R, S_k, G)$
**if** $j == 1$ **then**
$\theta_i \leftarrow k$
**if** $j \mathrel{!}= n_i$ **then**
$S_k \leftarrow \text{Append}(i, S_k)$
**end if**
**else if** $j == n_i$ **then**
$S_k \leftarrow \text{Remove}(i, S_k)$
**end if**
$\psi_{i,j} = \max(L_{\theta_i}, \psi_{i,j})$; $L_{\theta_i} \leftarrow \psi_{i,j} + C_{i,j}$
$A \leftarrow \text{Append}(P_{i,j}, A)$; $R \leftarrow \text{Remove}(P_{i,j}, R)$
**for** $P_{k,z} \mid (P_{i,j}, P_{k,z}) \in E$ **do**
**if** $\psi_{k,z} < \psi_{i,j} + C_{i,j}$ **then**
$\psi_{k,z} \leftarrow \psi_{i,j} + C_{i,j}$;
**end if**
$F_{k,z} \leftarrow F_{k,z} + 1$
**if** $F_{k,z} == \text{Size}(\text{InEdges}_{k,z})$ **then**
$R \leftarrow \text{Append}(P_{k,z}, R)$
**end if**
**end for**
**end while**
$\mu = \max_{i=1}^{m} L_i$
**return** $(\mu, \psi, \theta)$
**end procedure**

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**

Find the earliest available thread

**procedure** $\textsc{HeurTied}(G, m)$
$A \leftarrow \emptyset$; $R \leftarrow p_{1,1}$
$L \leftarrow \textsc{Array}(m, 0)$; $S \leftarrow \textsc{Array}(m, \emptyset)$
**while** $\textsc{Size}(A) \mathrel{!}= \sum_{i=1}^{N} n_i$ **do**
$k \leftarrow \textsc{FirstIdleThread}(L)$
$P_{i,j} \leftarrow \textsc{NextReadyJob}(k, R, S_k, G)$
**if** $j == 1$ **then**
$\theta_i \leftarrow k$
**if** $j \mathrel{!}= n_i$ **then**
$S_k \leftarrow \textsc{Append}(i, S_k)$
**end if**
**else if** $j == n_i$ **then**
$S_k \leftarrow \textsc{Remove}(i, S_k)$
**end if**
$\psi_{i,j} = \max(L_{\theta_i}, \psi_{i,j})$; $L_{\theta_i} \leftarrow \psi_{i,j} + C_{i,j}$
$A \leftarrow \textsc{Append}(P_{i,j}, A)$; $R \leftarrow \textsc{Remove}(P_{i,j}, R)$
**for** $P_{k,z} \mid (P_{i,j}, P_{k,z}) \in E$ **do**
**if** $\psi_{k,z} < \psi_{i,j} + C_{i,j}$ **then**
$\psi_{k,z} \leftarrow \psi_{i,j} + C_{i,j}$;
**end if**
$F_{k,z} \leftarrow F_{k,z} + 1$
**if** $F_{k,z} == \textsc{Size}(\textsc{InEdges}_{k,z})$ **then**
$R \leftarrow \textsc{Append}(P_{k,z}, R)$
**end if**
**end for**
**end while**
$\mu = \max_{i=1}^{m} L_i$
**return** $(\mu, \psi, \theta)$
**end procedure**

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**

  Find the next ready task-part according to previous heuristics
  - Checks tied tasks scheduling restrictions

**procedure** $\textsc{HeurTied}(G, m)$
  $A \leftarrow \emptyset; R \leftarrow p_{1,1}$
  $L \leftarrow \textsc{Array}(m, 0); S \leftarrow \textsc{Array}(m, \emptyset)$
  **while** $\textsc{Size}(A) \mathrel{!}= \sum_{i=1}^{N} n_i$ **do**
    $k \leftarrow \textsc{FirstIdleThread}(L)$
    $P_{i,j} \leftarrow \textsc{NextReadyJob}(k, R, S_k, G)$
    **if** $j == 1$ **then**
      $\theta_i \leftarrow k$
      **if** $j \mathrel{!}= n_i$ **then**
        $S_k \leftarrow \textsc{Append}(i, S_k)$
      **end if**
    **else if** $j == n_i$ **then**
      $S_k \leftarrow \textsc{Remove}(i, S_k)$
    **end if**
    $\psi_{i,j} = \max(L_{\theta_i}, \psi_{i,j}); L_{\theta_i} \leftarrow \psi_{i,j} + C_{i,j}$
    $A \leftarrow \textsc{Append}(P_{i,j}, A); R \leftarrow \textsc{Remove}(P_{i,j}, R)$
    **for** $P_{k,z} \mid (P_{i,j}, P_{k,z}) \in E$ **do**
      **if** $\psi_{k,z} < \psi_{i,j} + C_{i,j}$ **then**
        $\psi_{k,z} \leftarrow \psi_{i,j} + C_{i,j};$
      **end if**
      $F_{k,z} \leftarrow F_{k,z} + 1$
      **if** $F_{k,z} == \textsc{Size}(\textsc{InEdges}_{k,z})$ **then**
        $R \leftarrow \textsc{Append}(P_{k,z}, R)$
      **end if**
    **end for**
  **end while**
  $\mu = \max_{i=1}^{m} L_i$
  **return** $(\mu, \psi, \theta)$
**end procedure**

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**

Update task-part mapping

**procedure** HEURTIED$(G, m)$
    $A \leftarrow \emptyset$; $R \leftarrow p_{1,1}$
    $L \leftarrow$ ARRAY$(m, 0)$; $S \leftarrow$ ARRAY$(m, \emptyset)$
    **while** SIZE$(A)$ $!= \sum_{i=1}^{N} n_i$ **do**
        $k \leftarrow$ FIRSTIDLETHREAD$(L)$
        $P_{i,j} \leftarrow$ NEXTREADYJOB$(k, R, S_k, G)$
        **if** $j == 1$ **then**
            $\theta_i \leftarrow k$
            **if** $j$ $!= n_i$ **then**
                $S_k \leftarrow$ APPEND$(i, S_k)$
            **end if**
        **else if** $j == n_i$ **then**
            $S_k \leftarrow$ REMOVE$(i, S_k)$
        **end if**
        $\psi_{i,j} = \max(L_{\theta_i}, \psi_{i,j})$; $L_{\theta_i} \leftarrow \psi_{i,j} + C_{i,j}$
        $A \leftarrow$ APPEND$(P_{i,j}, A)$; $R \leftarrow$ REMOVE$(P_{i,j}, R)$
        **for** $P_{k,z} \mid (P_{i,j}, P_{k,z}) \in E$ **do**
            **if** $\psi_{k,z} < \psi_{i,j} + C_{i,j}$ **then**
                $\psi_{k,z} \leftarrow \psi_{i,j} + C_{i,j}$;
            **end if**
            $F_{k,z} \leftarrow F_{k,z} + 1$
            **if** $F_{k,z} ==$ SIZE(INEDGES$_{k,z}$) **then**
                $R \leftarrow$ APPEND$(P_{k,z}, R)$
            **end if**
        **end for**
    **end while**
    $\mu = \max_{i=1}^{m} L_i$
    **return** $(\mu, \psi, \theta)$
**end procedure**

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**

Update
- task-part starting time
- thread next idle time

```
procedure HeurTied(G, m)
    A ← ∅; R ← p_{1,1}
    L ← Array(m, 0); S ← Array(m, ∅)
    while Size(A) != \sum_{i=1}^{N} n_i do
        k ← FirstIdleThread(L)
        P_{i,j} ← NextReadyJob(k, R, S_k, G)
        if j == 1 then
            θ_i ← k
            if j != n_i then
                S_k ← Append(i, S_k)
            end if
        else if j == n_i then
            S_k ← Remove(i, S_k)
        end if
        ψ_{i,j} = max(L_{θ_i}, ψ_{i,j}); L_{θ_i} ← ψ_{i,j} + C_{i,j}
        A ← Append(P_{i,j}, A); R ← Remove(P_{i,j}, R)
        for P_{k,z} | (P_{i,j}, P_{k,z}) ∈ E do
            if ψ_{k,z} < ψ_{i,j} + C_{i,j} then
                ψ_{k,z} ← ψ_{i,j} + C_{i,j};
            end if
            F_{k,z} ← F_{k,z} + 1
            if F_{k,z} == Size(InEdges_{k,z}) then
                R ← Append(P_{k,z}, R)
            end if
        end for
    end while
    μ = max_{i=1}^{m} L_i
    return (μ, ψ, θ)
end procedure
```

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**

Check next ready jobs

**procedure** $\textsc{HeurTied}(G, m)$
  $A \leftarrow \emptyset$; $R \leftarrow p_{1,1}$
  $L \leftarrow \textsc{Array}(m, 0)$; $S \leftarrow \textsc{Array}(m, \emptyset)$
  **while** $\textsc{Size}(A) \mathrel{!}= \sum_{i=1}^{N} n_i$ **do**
    $k \leftarrow \textsc{FirstIdleThread}(L)$
    $P_{i,j} \leftarrow \textsc{NextReadyJob}(k, R, S_k, G)$
    **if** $j == 1$ **then**
      $\theta_i \leftarrow k$
      **if** $j \mathrel{!}= n_i$ **then**
        $S_k \leftarrow \textsc{Append}(i, S_k)$
      **end if**
    **else if** $j == n_i$ **then**
      $S_k \leftarrow \textsc{Remove}(i, S_k)$
    **end if**
    $\psi_{i,j} = \max(L_{\theta_i}, \psi_{i,j})$; $L_{\theta_i} \leftarrow \psi_{i,j} + C_{i,j}$
    $A \leftarrow \textsc{Append}(P_{i,j}, A)$; $R \leftarrow \textsc{Remove}(P_{i,j}, R)$
    **for** $P_{k,z} \mid (P_{i,j}, P_{k,z}) \in E$ **do**
      **if** $\psi_{k,z} < \psi_{i,j} + C_{i,j}$ **then**
        $\psi_{k,z} \leftarrow \psi_{i,j} + C_{i,j}$;
      **end if**
      $F_{k,z} \leftarrow F_{k,z} + 1$
      **if** $F_{k,z} == \textsc{Size}(\textsc{InEdges}_{k,z})$ **then**
        $R \leftarrow \textsc{Append}(P_{k,z}, R)$
      **end if**
    **end for**
  **end while**
  $\mu = \max_{i=1}^{m} L_i$
  **return** $(\mu, \psi, \theta)$
**end procedure**

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**

Computes makespan

**procedure** $\textsc{HeurTied}(G, m)$
$A \leftarrow \emptyset;\ R \leftarrow p_{1,1}$
$L \leftarrow \textsc{Array}(m, 0);\ S \leftarrow \textsc{Array}(m, \emptyset)$
**while** $\textsc{Size}(A)\ != \sum_{i=1}^{N} n_i$ **do**
$k \leftarrow \textsc{FirstIdleThread}(L)$
$P_{i,j} \leftarrow \textsc{NextReadyJob}(k, R, S_k, G)$
**if** $j == 1$ **then**
$\theta_i \leftarrow k$
**if** $j\ != n_i$ **then**
$S_k \leftarrow \textsc{Append}(i, S_k)$
**end if**
**else if** $j == n_i$ **then**
$S_k \leftarrow \textsc{Remove}(i, S_k)$
**end if**
$\psi_{i,j} = \max(L_{\theta_i}, \psi_{i,j});\ L_{\theta_i} \leftarrow \psi_{i,j} + C_{i,j}$
$A \leftarrow \textsc{Append}(P_{i,j}, A);\ R \leftarrow \textsc{Remove}(P_{i,j}, R)$
**for** $P_{k,z} \mid (P_{i,j}, P_{k,z}) \in E$ **do**
**if** $\psi_{k,z} < \psi_{i,j} + C_{i,j}$ **then**
$\psi_{k,z} \leftarrow \psi_{i,j} + C_{i,j};$
**end if**
$F_{k,z} \leftarrow F_{k,z} + 1$
**if** $F_{k,z} == \textsc{Size}(\textsc{InEdges}_{k,z})$ **then**
$R \leftarrow \textsc{Append}(P_{k,z}, R)$
**end if**
**end for**
**end while**
$\mu = \max_{i=1}^{m} L_i$
**return** $(\mu, \psi, \theta)$
**end procedure**

# Strategy 2: Sub-optimal static allocation

- **Tied Tasks**
- **Untied task**
  - Slightly simpler algorithm
- **Complexity:** $O(N^2p^2)$

```
procedure HeurTied(G, m)
    A ← ∅; R ← p_{1,1}
    L ← Array(m, 0); S ← Array(m, ∅)
    while Size(A) != ∑_{i=1}^{N} n_i do
        k ← FirstIdleThread(L)
        P_{i,j} ← NextReadyJob(k, R, S_k, G)
        if j == 1 then
            θ_i ← k
            if j != n_i then
                S_k ← Append(i, S_k)
            end if
        else if j == n_i then
            S_k ← Remove(i, S_k)
        end if
        ψ_{i,j} = max(L_{θ_i}, ψ_{i,j}); L_{θ_i} ← ψ_{i,j} + C_{i,j}
        A ← Append(P_{i,j}, A); R ← Remove(P_{i,j}, R)
        for P_{k,z} | (P_{i,j}, P_{k,z}) ∈ E do
            if ψ_{k,z} < ψ_{i,j} + C_{i,j} then
                ψ_{k,z} ← ψ_{i,j} + C_{i,j};
            end if
            F_{k,z} ← F_{k,z} + 1
            if F_{k,z} == Size(InEdges_{k,z}) then
                R ← Append(P_{k,z}, R)
            end if
        end for
    end while
    μ = max_{i=1}^{m} L_i
    return (μ, ψ, θ)
end procedure
```

# Evaluation: Experimental setting

- **Static allocation strategies vs. Response-Time upper bound [1]**
- **Task sets**
  - Real OpenMP 3D path planning application
  - Synthetic DAG task-sets
- **Intel® Core™ i7-4770K CPU 3.50 GHz**
  - 16GB RAM
  - ILP solver: IBM ILOG CPLEX Optimization Studio v.12.61

[1] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna and E. Quiñones, *"Timing characterization of OpenMP4 tasking model"*, in CASES, 2015.

# Evaluation: 3D Path Planning application

- **Real case study: Airborne collision avoidance**
- **Application set ups: 3DPP1 and 3DPP2**
  - DAGs composed of 129 and 257 nodes, respectively

| | 3DPP1 (m=8) | 3DPP2 (m=2) | 3DPP2 (m=4) | 3DPP2 (m=8) |
|---|---|---|---|---|

# Evaluation: 3D Path Planning application

- **Real case study: Airborne collision avoidance**
- **Application set ups: 3DPP1 and 3DPP2**
  - DAGs composed of 129 and 257 nodes, respectively

| | 3DPP1 (m=8) | 3DPP2 (m=2) | 3DPP2 (m=4) | 3DPP2 (m=8) |
|---|---|---|---|---|
| ILP | 254 | 506 | 506 | 506 |

**ILP:** Converged in ~10 sec. to the best found solution

# Evaluation: 3D Path Planning application

- **Real case study: Airborne collision avoidance**
- **Application set ups: 3DPP1 and 3DPP2**
  - DAGs composed of 129 and 257 nodes, respectively

| | 3DPP1 (m=8) | 3DPP2 (m=2) | 3DPP2 (m=4) | 3DPP2 (m=8) |
|---|---|---|---|---|
| ILP | 254 | 506 | 506 | 506 |
| SPT | 317 | 824 | 660 | 571 |
| LPT | 254 | 659 | 577 | 530 |
| LNS | 254 | 715 | 506 | 506 |
| LNSNL | 300 | 748 | 619 | 549 |
| LRW | 254 | 717 | 506 | 506 |

**Sub-optimal heuristics**

# Evaluation: 3D Path Planning application

- **Real case study: Airborne collision avoidance**
- **Application set ups: 3DPP1 and 3DPP2**
  - DAGs composed of 129 and 257 nodes, respectively

| | 3DPP1 (m=8) | 3DPP2 (m=2) | 3DPP2 (m=4) | 3DPP2 (m=8) |
|---|---|---|---|---|
| ILP | 254 | 506 | 506 | 506 |
| SPT | 317 | 824 | 660 | 571 |
| LPT | 254 | 659 | 577 | 530 |
| LNS | 254 | 715 | 506 | 506 |
| LNSNL | 300 | 748 | 619 | 549 |
| LRW | 254 | 717 | 506 | 506 |
| | (7s) | (11m41s) | (11m48s) | (11m53s) |

**Sub-optimal heuristics**

# Evaluation: 3D Path Planning application

- **Real case study: Airborne collision avoidance**
- **Application set ups: 3DPP1 and 3DPP2**
  - DAGs composed of 129 and 257 nodes, respectively

| | 3DPP1 (m=8) | 3DPP2 (m=2) | 3DPP2 (m=4) | 3DPP2 (m=8) |
|---|---|---|---|---|
| ILP | 254 | 506 | 506 | 506 |
| SPT | 317 | 824 | 660 | 571 |
| LPT | 254 | 659 | 577 | 530 |
| LNS | 254 | 715 | 506 | 506 |
| LNSNL | 300 | 748 | 619 | 549 |
| LRW | 254 | 717 | 506 | 506 |
| BOUND-untied | 331 | 827 | 666.5 | 586.25 |

**Dynamic approach**

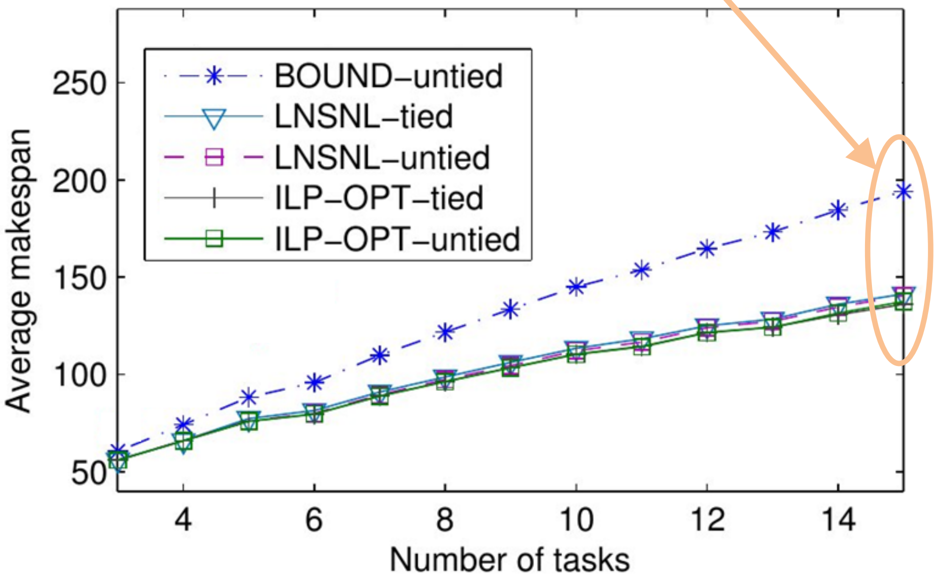
Max. over estimation: 63%

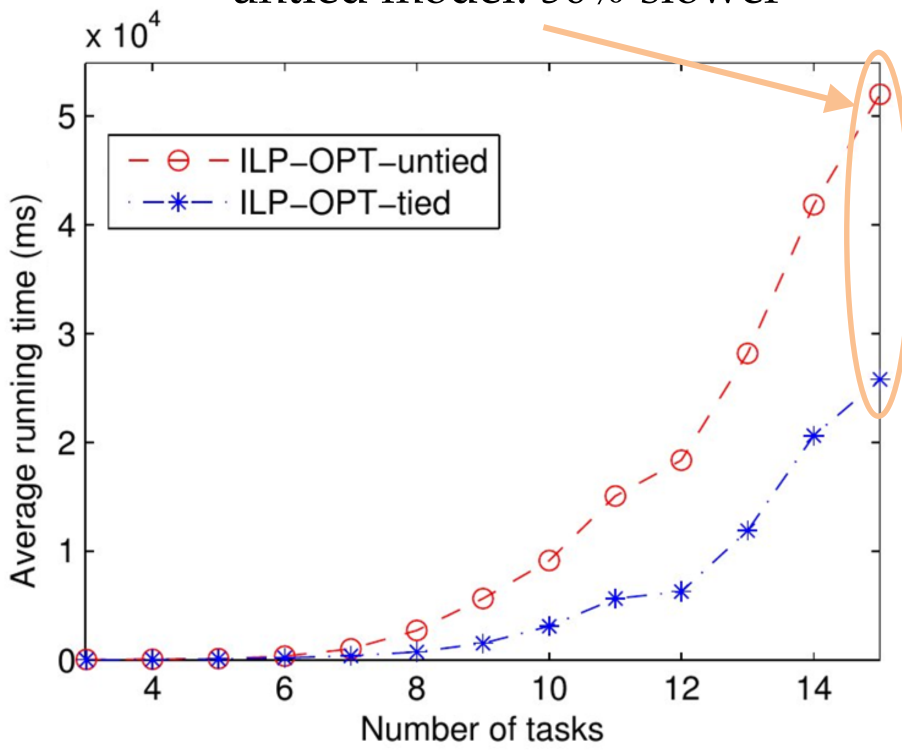
# Evaluation: Synthetic OpenMP-DAGs

- **Small task sets, 4 cores**

Max. over estimation
dynamic vs. static ILP: ~40%

Larger solution space for the
untied model: 50% slower



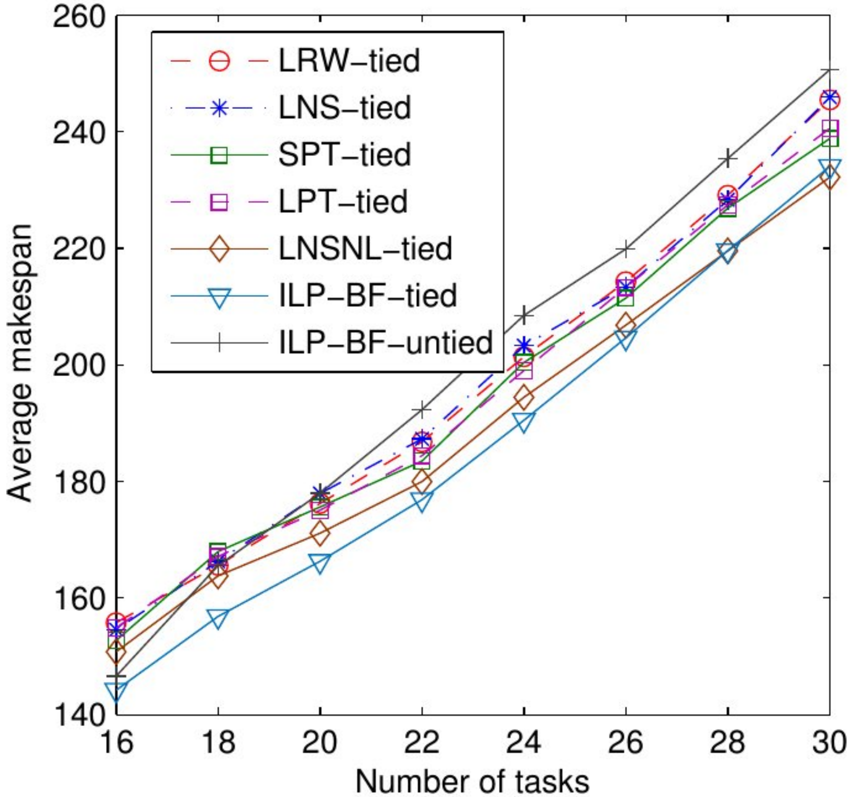
# Evaluation: Synthetic OpenMP-DAGs

- **Large task sets, 4 cores**

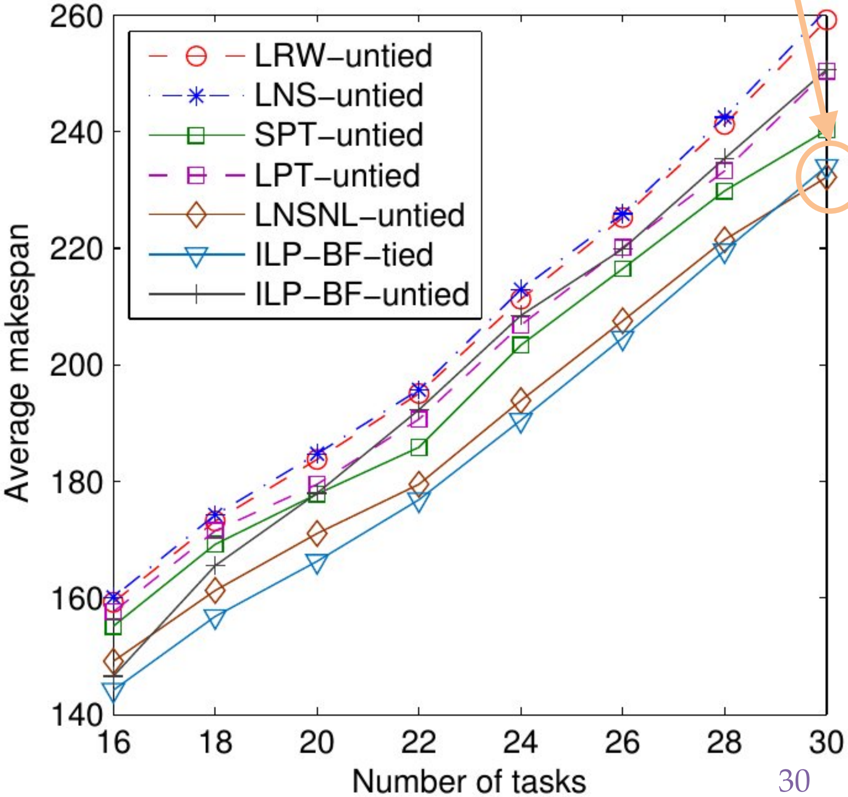
  Best feasible solution by ILP solver in 300 s

LNSNL outperforms ILP for untied

**OpenMP Tied Model**




**OpenMP Untied Model**



# Conclusions

- **Parallel programing models are fundamental to exploit the performance capabilities of parallel architectures**
  - OpenMP, one of the most advanced
- **However, relies on dynamic scheduling, not suitable in certain safety-critical domains**
- **We propose two OpenMP-complain static allocation strategies:**
  - A computationally expensive but optimal ILP solver
  - More efficient but sub-optimal heuristics

# A static scheduling approach to enable safety-critical OpenMP applications

Alessandra Melani, **Maria A. Serrano**, Marko Bertogna,
Isabella Cerutti, Eduardo Quiñones, Giorgio Buttazzo

This work was supported by the EU projects P-SOCRATES (FP7-ICT-2013-10) and HERCULES (H2020/ICT/2015/688860) and the Spanish Ministry of Science and Innovation grant TIN2015-65316-P

ASP-DAC 2017
January 16-19, 2017
Chiba/Tokyo, Japan