P-SOCRATES
Parallel Software Framework for Time-Critical many-core Systems

A Lightweight OpenMP4 Run-time for Embedded Systems

Roberto E. Vargas, Sara Royuela, **Maria A. Serrano**, Xavi Martorell and Eduardo Quiñones

ASP-DAC 2016
January 25-28, 2016
Macao

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Parallel programming models in Real-Time Embedded Systems

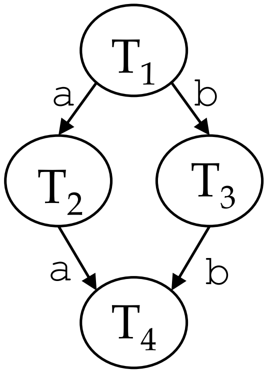
- **Why parallel programming models?**
  - Provide a high level abstraction of parallel architectures
  - Reduce the complexity of parallel programming
  - Allows exploiting the performance of many-core processors
- **Real-Time systems require more performance**
  - Composed of complex applications
  - Parallel programming models enable current many-core embedded processors to provide it
- **OpenMP, a well known parallel programming model**
  - Widely used in HPC
  - Increasingly adopted in embedded systems

OpenMP and Real-Time Systems

- **OpenMP4 tasking model allows expressing fine-grained and irregular parallelism**
  - Task
    Independent parallel unit of work and its data environment
  - Data dependencies

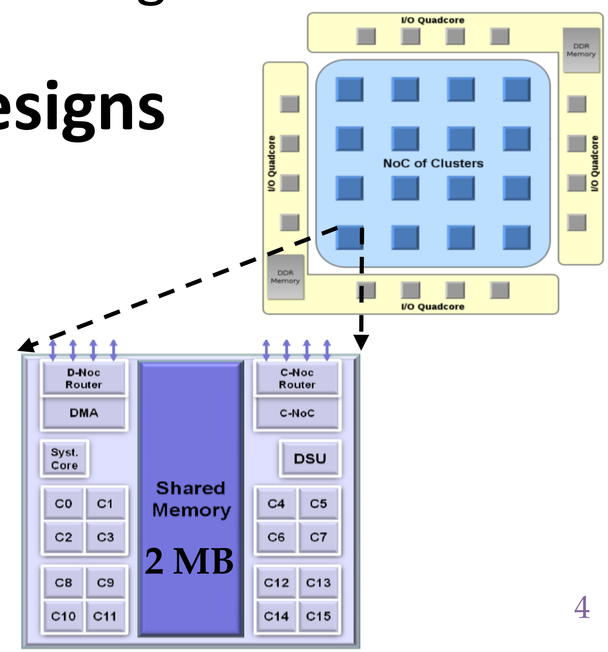
```
#pragma omp task depend(out:a,b)     // T1
{ … }
#pragma omp task depend(inout:a)     // T2
{ … }
#pragma omp task depend(inout:b)     // T3
{ … }
#pragma omp task depend(in:a,b)      // T4
{ … }
```

- **OpenMP4 tasking model resembles the way Real-Time applications are modeled**
  - Task Dependency Graph (TDG) or Direct Acyclic Graph (DAG)



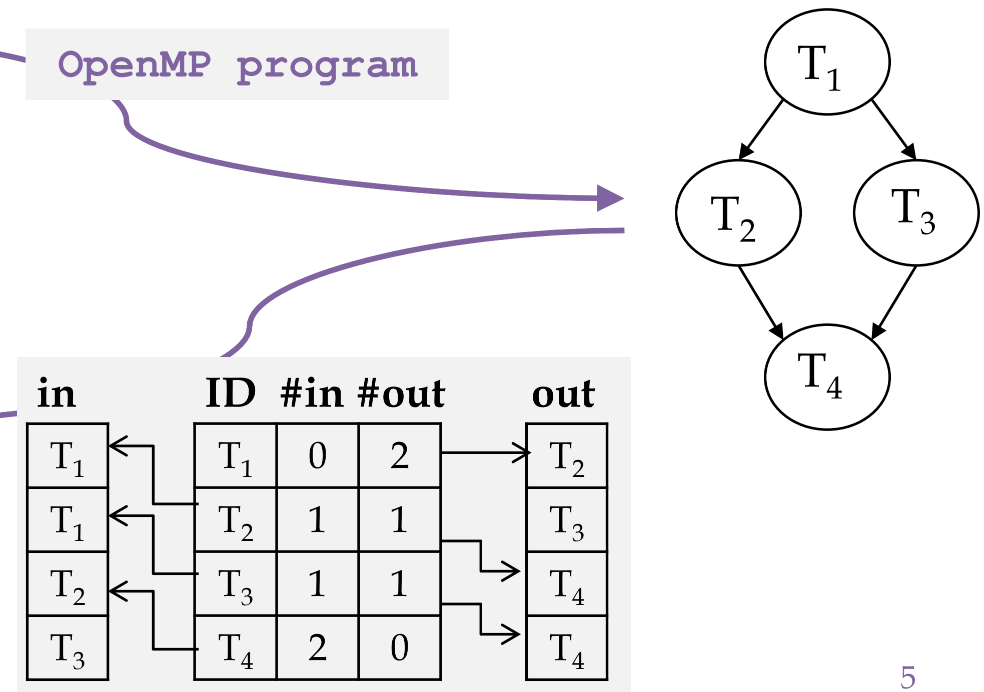
OpenMP Run-time

- **Current OpenMP implementations are designed for general purpose architectures**
  - e.g. libgomp (GCC), nanos++ (OmpSs)
  - Require <u>large data structures in memory</u> (hash table) to track at run-time the dependencies among tasks
- **Modern many-core embedded designs**
  - e.g. Kalray MPPA
  - Rely on computing fabrics with <u>small on-chip memories</u>



Memory efficient OpenMP Run-time

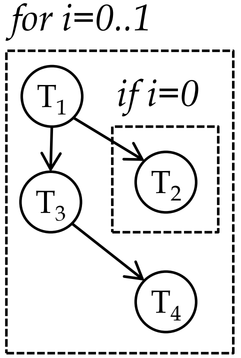
- **SOLUTION: Derive the complete TDG at compile time and maintain it in memory at run-time**
  – Although counter-intuitive, the memory consumption is reduced using more memory efficient structures
- **New compiler pass**
  – Derives the TDG of a OpenMP program
- **New Run-Time**
  – Efficiently stores and manages the TDG



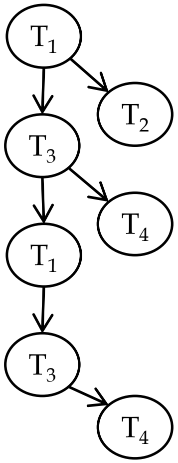
Compiler pass: Static construction of TDG

```
for (i = 0; i < 2; i++ ){
    #pragma omp task … {              // T1
    if (i==0) {
        #pragma omp task { … }}       // T2
    #pragma omp task … {              // T3
        #pragma omp task { … }}       // T4
}
```

1. **The Control/Data Flow Analysis Stage generates the augmented static TDG (*asTDG)***
   i. Parallel Control Flow Analysis
   ii. Induction-Variable Analysis
   iii. Range Analysis



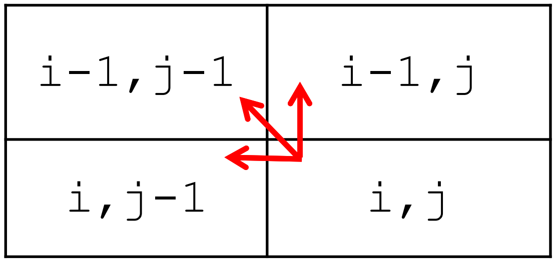
2. **The Task Expansion Stage generates the expanded static TDG (*esTDG*)**
   i. Expand the control flow structures
   ii. Resolve the dependency



Case Study: Matrix Processing

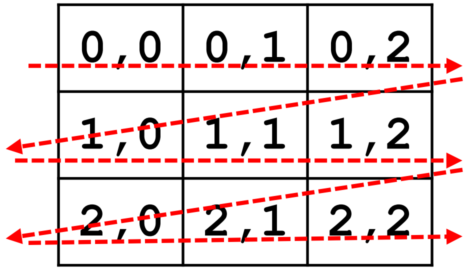
- **`compute_block(i, j)`**

```
m[i][j] = func(m[i-1][j-1],
               m[i-1][j],
               m[i][j-1])
```

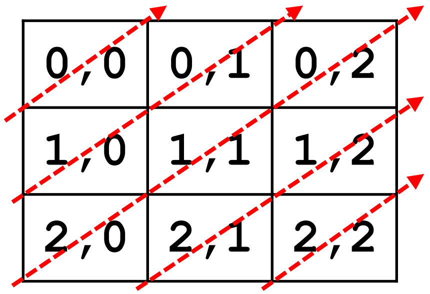


  – Sequential version

```
for (int i=0; i<=2; i++) {
    for (int j=0; j<=2; j++) {
        compute_block(i,j);
    }
}
```



  – Parallel version (Wave-front strategy)
    - Each task computes one block



Case Study: Matrix Processing

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

```
for (int i=0; i<=2; i++) {
      for (int j=0; j<=2; j++) {
         if (i==0 && j==0) {            // Initial block
            #pragma omp task depend (out: m[i][j])
            compute_block(i,j);         // Task region T1
         } elseif (i==0) {              // Blocks in upper edge
            #pragma omp task depend (in: m[i][j-1], out: m[i][j])
            compute_block(i,j);         // Task region T2
         } elseif (j==0) {              // Blocks in left edge
            #pragma omp task depend (in: m[i-1][j], out: m[i][j])
            compute_block(i,j);         // Task region T3
         } else {                       // Internal blocks
            #pragma omp task depend (in: m[i-1][j], in: m[i][j-1],
                                     in: m[i-1][j-1], out: m[i][j])
            compute_block(i,j);         // Task region T4
         }
}}
```

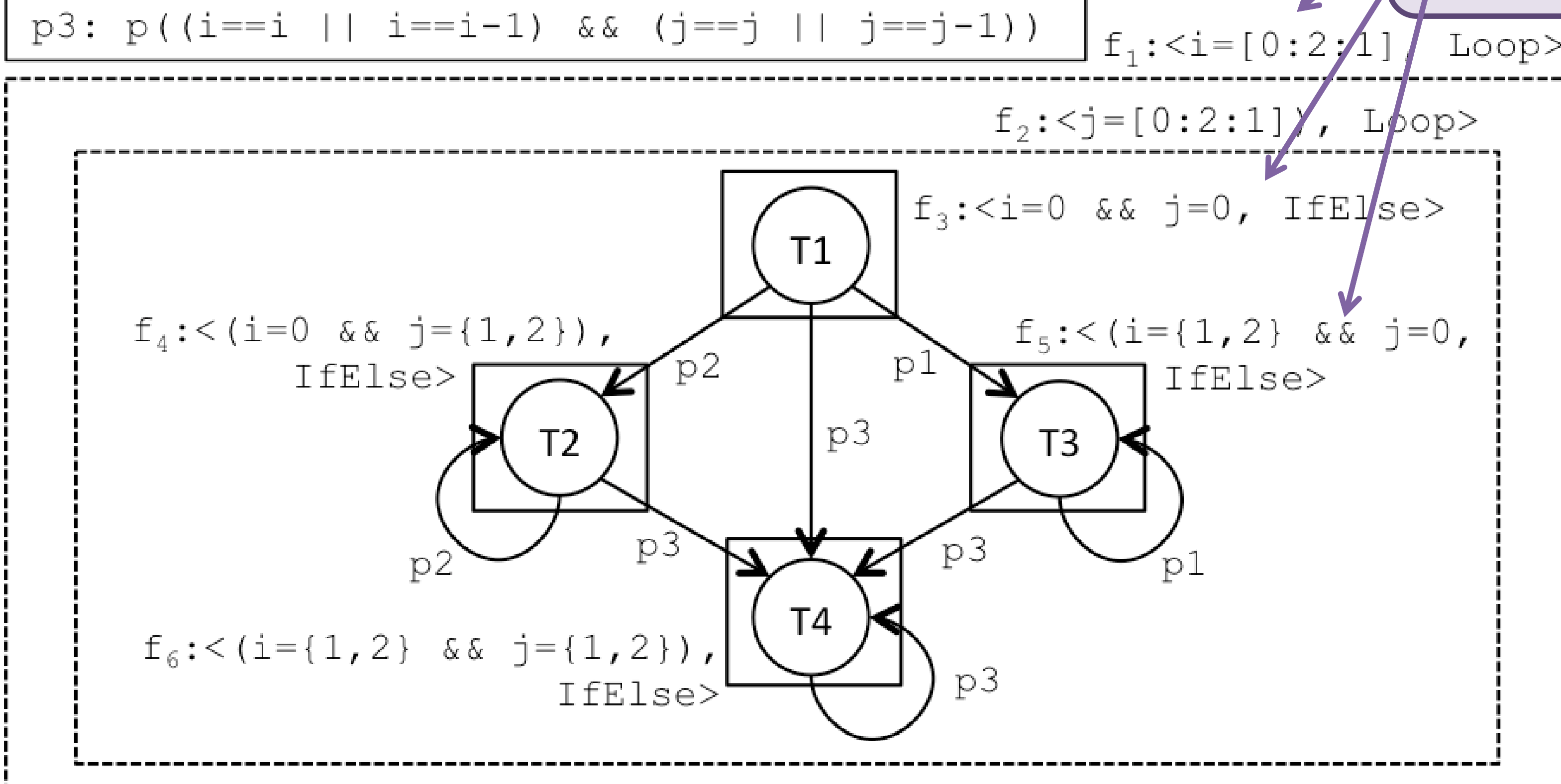
Case study: Control/Data Flow Analysis

- ***augmented static TDG*** **(*asTDG*)**

Control flow statements

C… dependencies

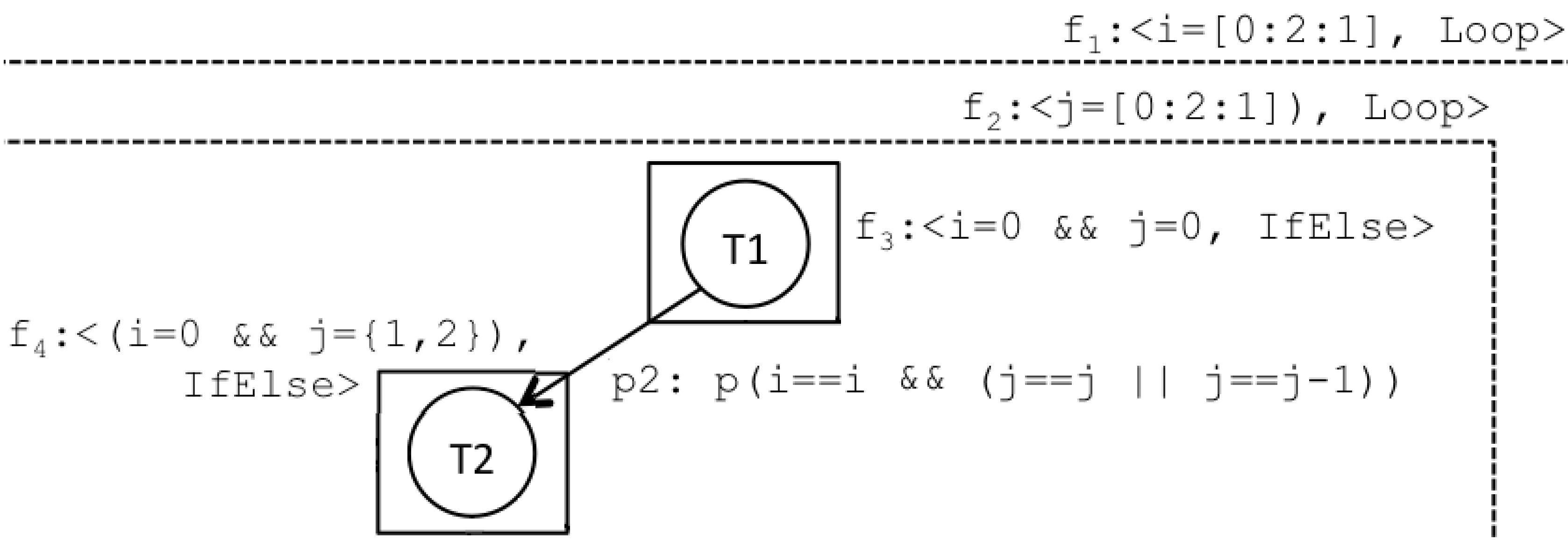
```
p1: p((i==i || i==i-1) && j==j)
p2: p(i==i && (j==j || j==j-1))
p3: p((i==i || i==i-1) && (j==j || j==j-1))
```



Case study: Control/Data Flow Analysis

```
... for (int i=0; i<=2; i++) {                                  ← f1
        for (int j=0; j<=2; j++) {                              ← f2
            if (i==0 && j==0) {                                 ← f3
                #pragma omp task depend (inout: m[i][j])
                compute_block(i,j);      // Task region T1
            } elseif (i==0) {                                   ← f4
                #pragma omp task depend (in: m[i][j-1], inout: m[i][j])
                computeblock(i,j);       // Task region T2
            }
...
```

p2

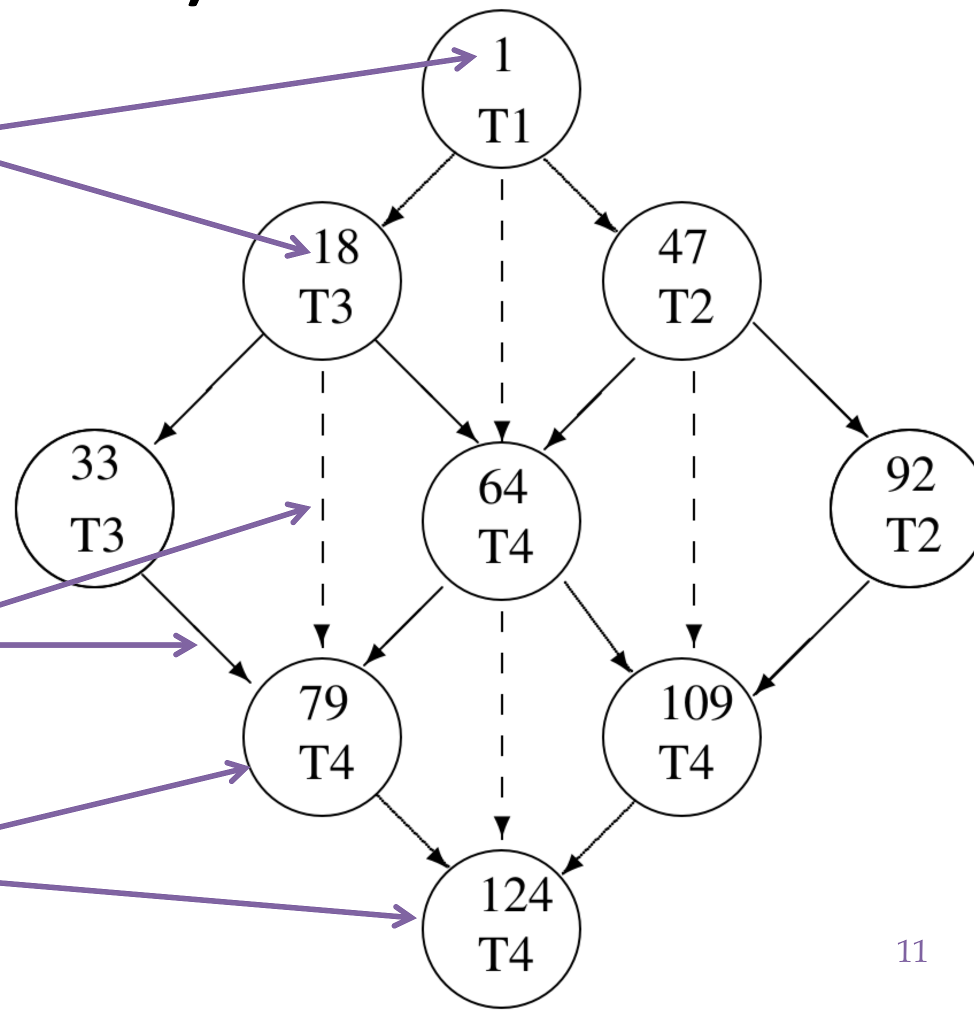


Case study: Task Expansion Stage

- ***expanded static TDG* (*esTDG*)**

Task ID: Allows the run-time to identify task instances and the corresponding task construct

$$t_{id} = sid_t + T \times \sum_{i=1}^{L_t} l_i \cdot M^i$$

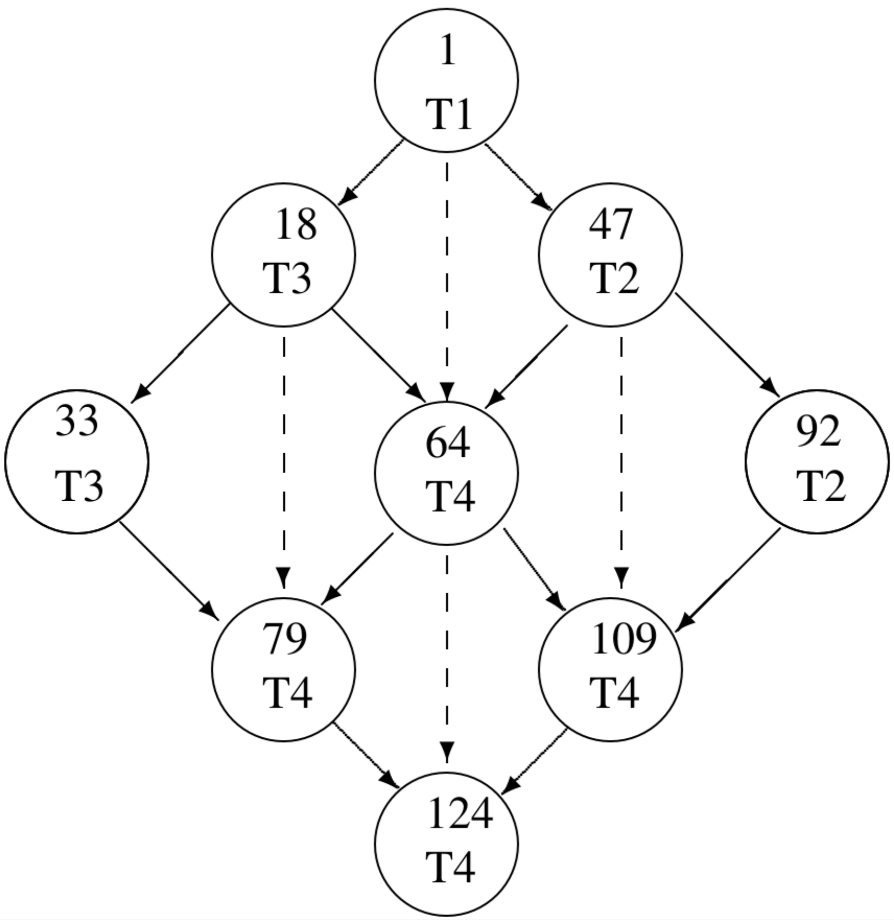


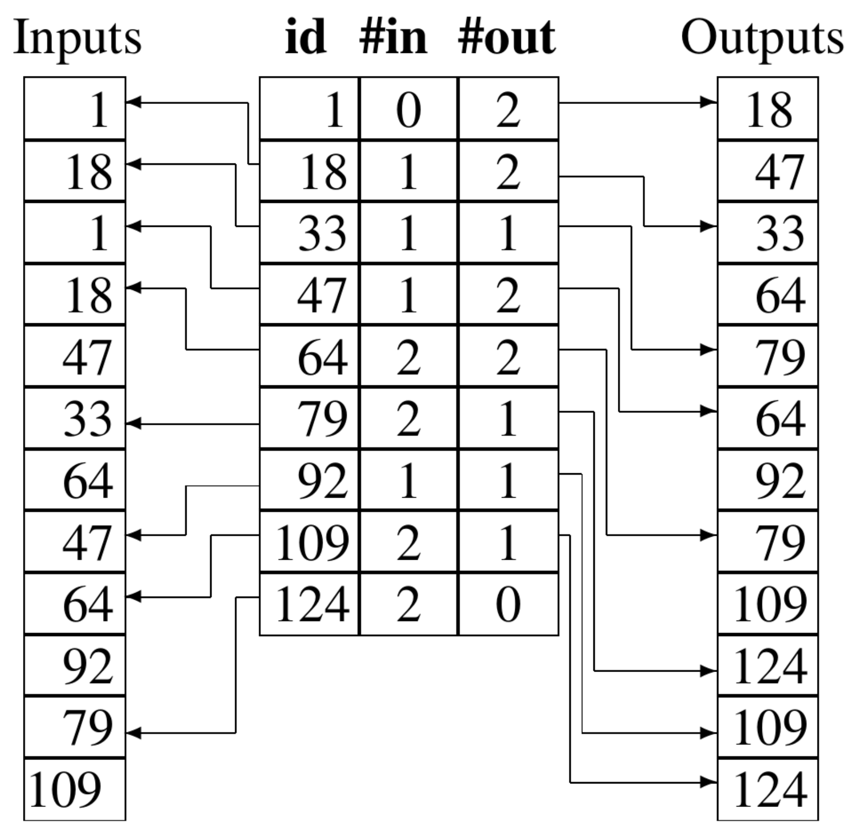
Dependencies among task instances

Task instances executed at run-time

OpenMP Run-Time

- **Representation of the *esTDG*: sparse matrix**

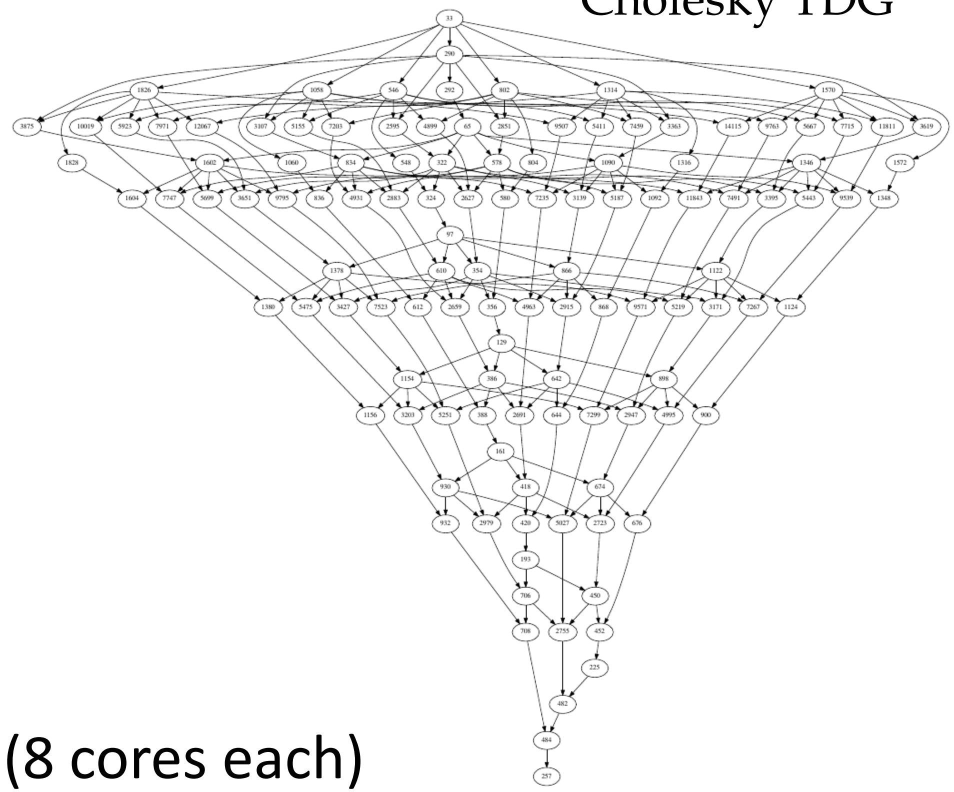




Evaluation: Experimental Setup

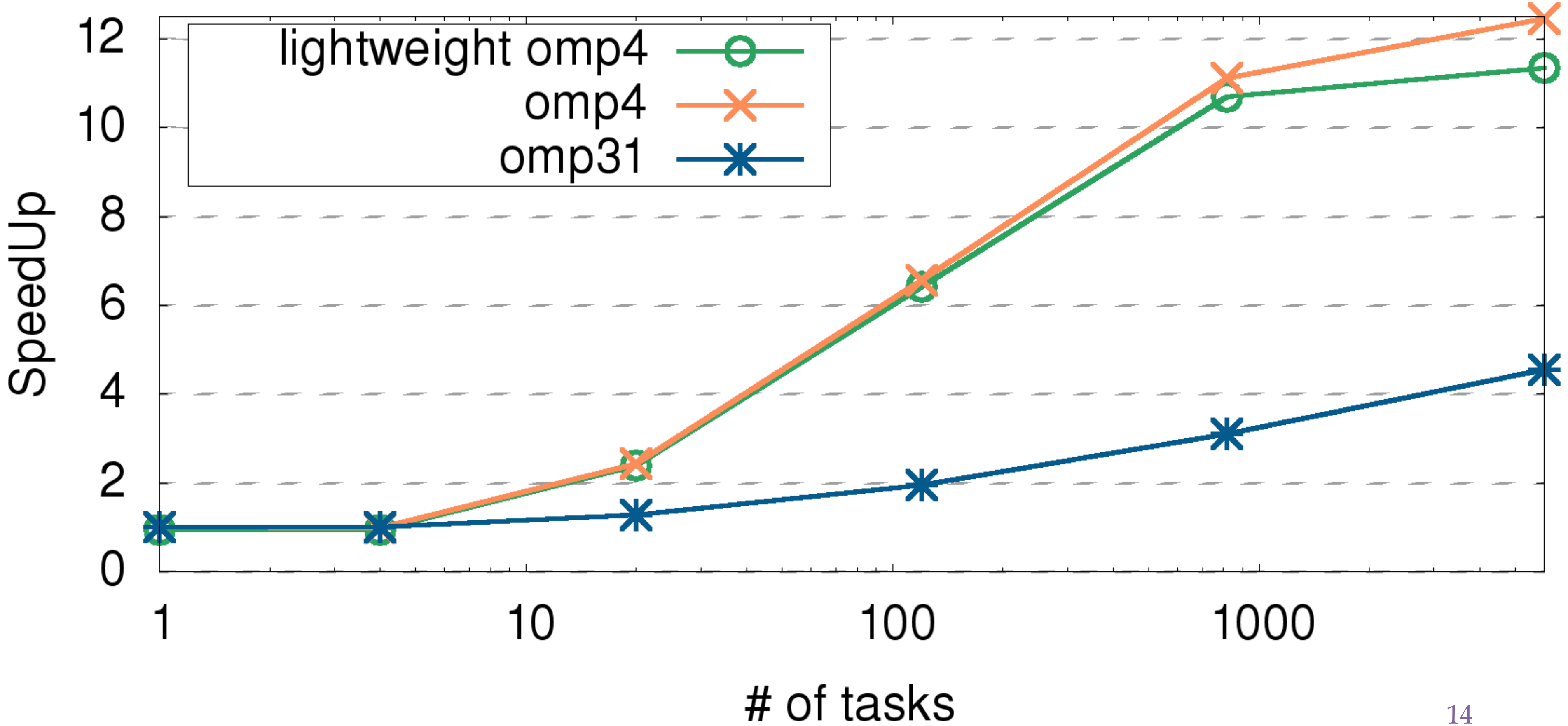
- **OpenMP framework**
  - New compiler pass
    - Mercurium
  - Lightweight Run-Time
    - GNU libgomp (GCC 4.7.2)
- **Application**
  - Cholesky Factorization
- **Platforms**
  - 2 Intel Xeon CPUs E5-2670 (8 cores each)
  - MPPA processor (256 cores: 16 clusters x16 cores; 2 MB per cluster)

Cholesky TDG



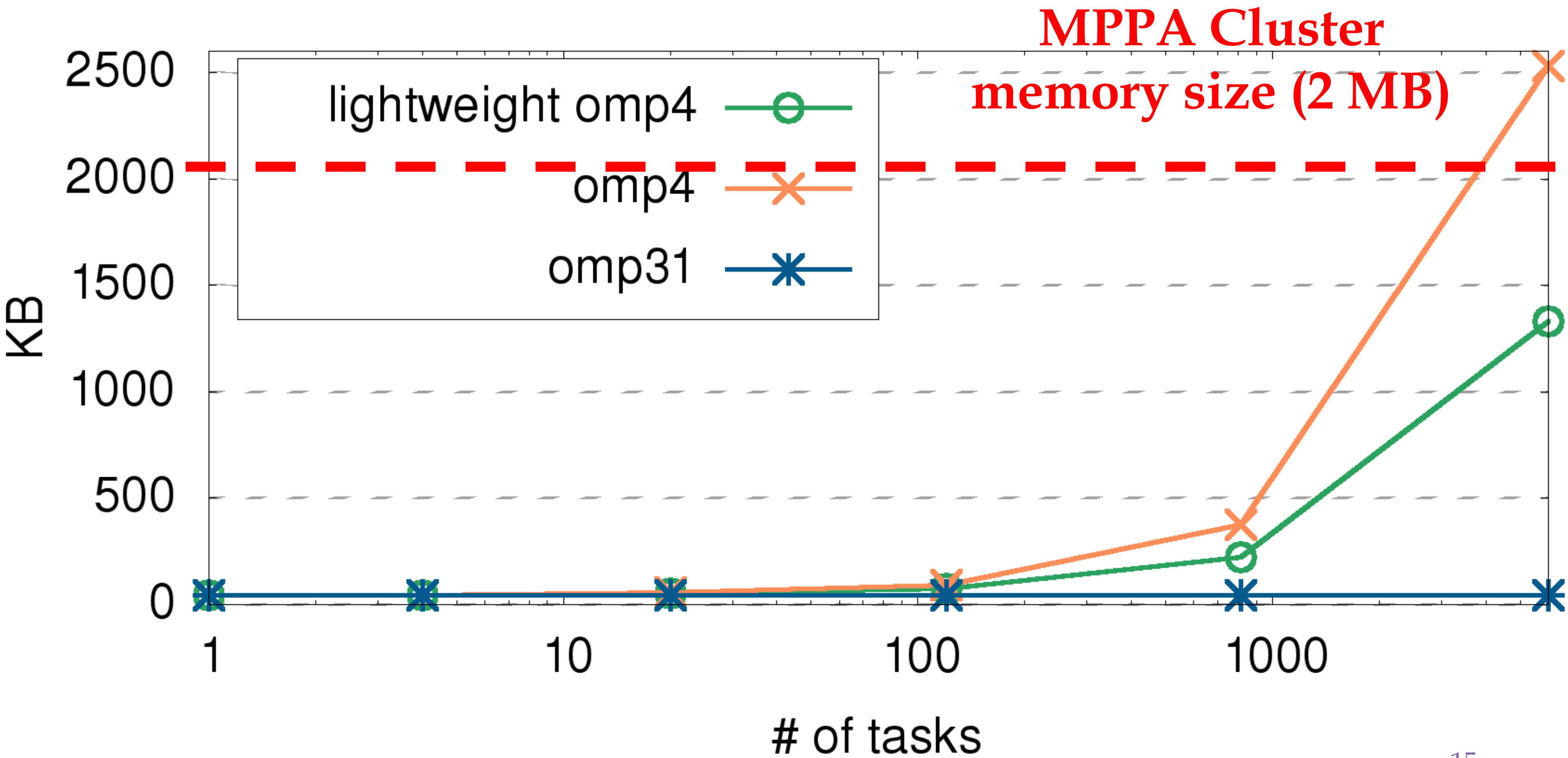
Evaluation: Performance speed-up

- **2 Intel Xeon CPUs (16 cores)**



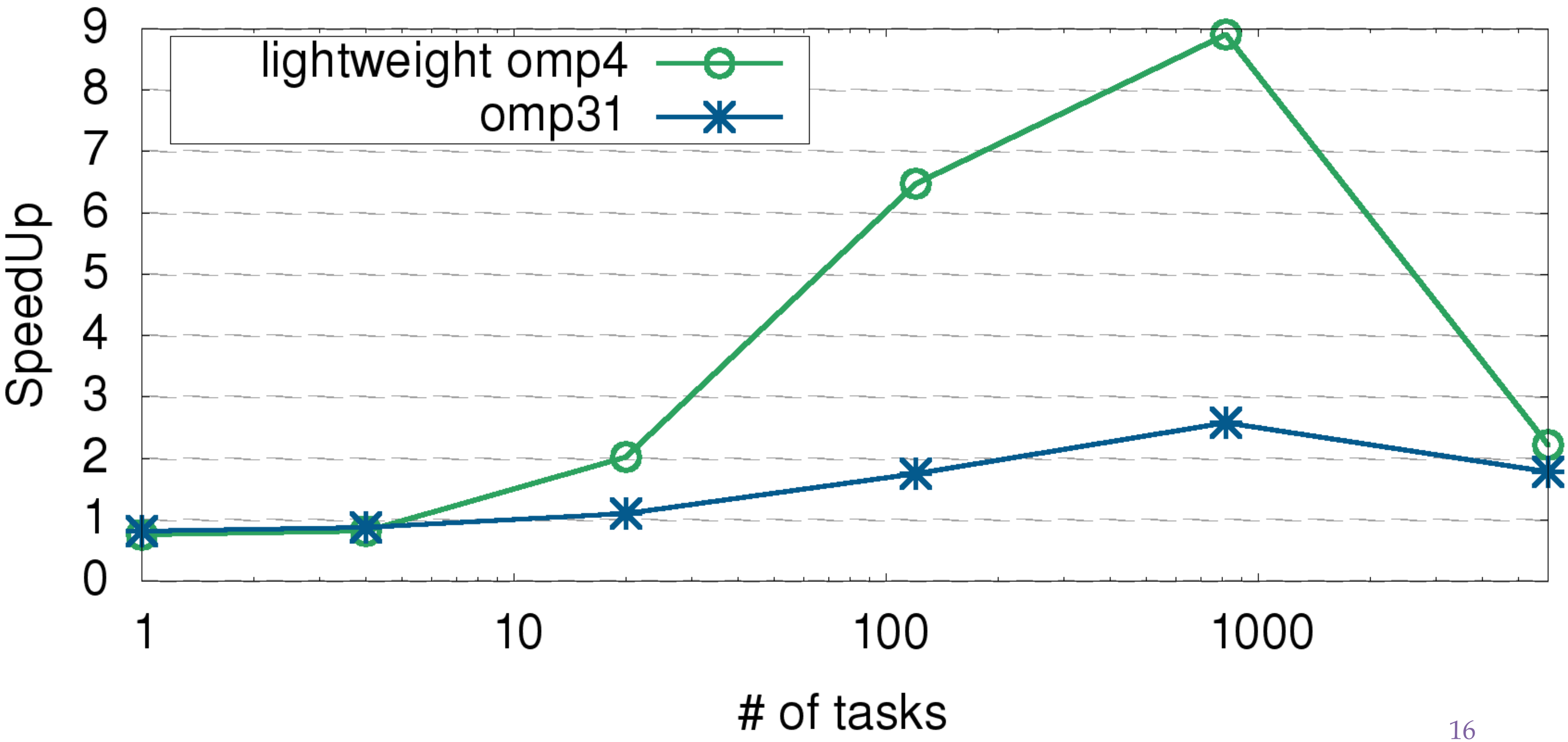
Evaluation: Memory usage

- **2 Intel Xeon CPUs (16 cores)**



Evaluation: Performance speed-up

- **MPPA (1 cluster, 16 cores)**



Conclusions

- **Parallel programming models are vital to exploit the parallel capabilities of many-core processors**
  - OpenMP is supported by most processors
- **Current OpenMP run-time implementations use large data structures**
  - Many-core processors rely on small on-chip memories
- **Our run-time handles the *static TDG***
  - New compiler pass derives it
  - Memory efficient data structure maintains it
- **Our approach provides similar speed-up of current run-times while reducing the memory consumption**

Acknowledgments

This work was supported by

- EU project P-SOCRATES (FP7- ICT-2013-10)
- Spanish Ministry of Science and Innovation grant TIN2012-34557.

P-SOCRATES

Parallel Software Framework for Time-Critical many-core Systems

A Lightweight OpenMP4 Run-time for Embedded Systems

Roberto E. Vargas, Sara Royuela, **Maria A. Serrano**, Xavi Martorell and Eduardo Quiñones

ASP-DAC 2016
January 25-28, 2016
Macao

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

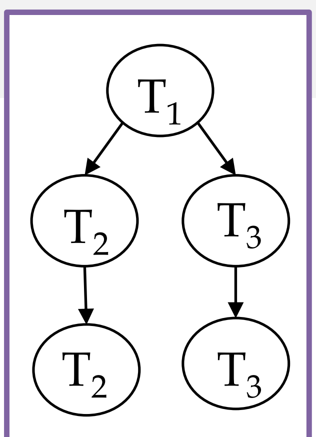
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Missing Information

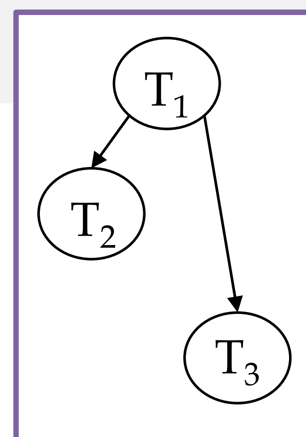
- **When an if-else statement cannot be evaluated**
  - All related tasks are instantiated
- **When the loop boundaries are unknown**
  - Disable parallelism across iterations by inserting a `barrier` at the end of the loop
- **When the dependency cannot be evaluated**
  - Dependency is always kept, forcing the involved tasks to be serialized.
- **The situations described above will result in a bigger esTDG or in a performance loss, although guarantee a correct esTDG.**

Missing Information

```
#pragma omp task depend(out:b,c)
// T1
for (i= 0; i < 2; i++){
  if(i == unknownCondition()) {
    #pragma omp task
                depend(inout:b)
    // T2
  } else {
    #pragma omp task
                depend(inout:c)
    // T3
  }
}
```




```
#pragma omp task depend(out:b,c)
// T1
for (i=0; i < 2; i++){
  if(i == 0) {
    #pragma omp task
                depend(inout:b)
    // T2
  } else {
    #pragma omp task
                depend(inout:c)
    // T3
  }
}
```



Compiler complexity

- **Control/Data flow analysis stage**
  i. Control Flow Analysis → Cyclomatic Complexity [1]
  ii. Induction-Variable Analysis
  iii. Range Analysis
  (ii–iii) → Asymptotic Linear Complexity [2]

- **Task expansion stage** → Quadratic on the number of instantiated tasks.

[1] T. McCabe. A complexity measure. *IEEE Transactions of Software Engineering*, 1976.
[2] R. E. Rodrigues, V. H. Sperle Campos, F. M. Quinto Pereira. A fast and low-overhead technique to secure programs against integer overflows. In *CGO*, 2013.