# A Dynamically Scheduled Architecture for the Synthesis of Graph Methods

MARCO MINUTOLI*, VITO GIOVANNI CASTELLANA*, ANTONINO TUMEO*, MARCO LATTUADA+, FABRIZIO FERRANDI+

*High Performance Computing, Pacific Northwest National Laboratory, 99352 Richland WA, USA
{marco.minutoli, vitoGiovanni.Castellana, antonino.tumeo}@pnnl.gov

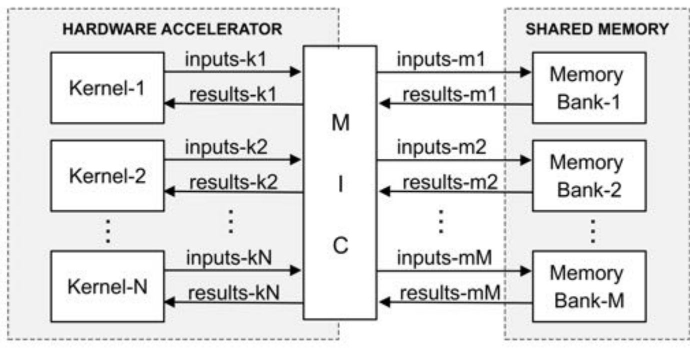
+DEIB, Politecnico di Milano, 20132 Milano, Italy
{marco.lattuada, fabrizio.ferrandi}@polimi.it

# Graph Methods and Data Analytics

- Emergence of new large-scale Data Analytics applications
  - Example: graph databases
- These applications employ graph as a convenient way to store data, and require graph methods to perform explorations (i.e., queries)
- They exhibit "irregular" behaviors:
  - Large datasets, not easily partitionable in balanced ways
  - Many fine-grained and unpredictable data accesses
  - High Synchronization intensity
  - Large amounts of fine-grained, dynamic parallelism (task based)
- Conventional general purpose processors or commodity accelerators (e.g., GPUs) are not well suited for these workloads
  - We can exploit custom accelerators on FPGAs
  - Hand-designing accelerators on FPGA is hard and time-consuming
  - High-Level Synthesis (HLS) enable generation of Register-Transfer Level code starting from high level specifications (e.g., C)
- Conventional HLS has been rarely applied to graph problems
  - We introduce a set of architectural templates to better support synthesis of graph methods

# First Architecture Template (Parallel Controller + MIC)

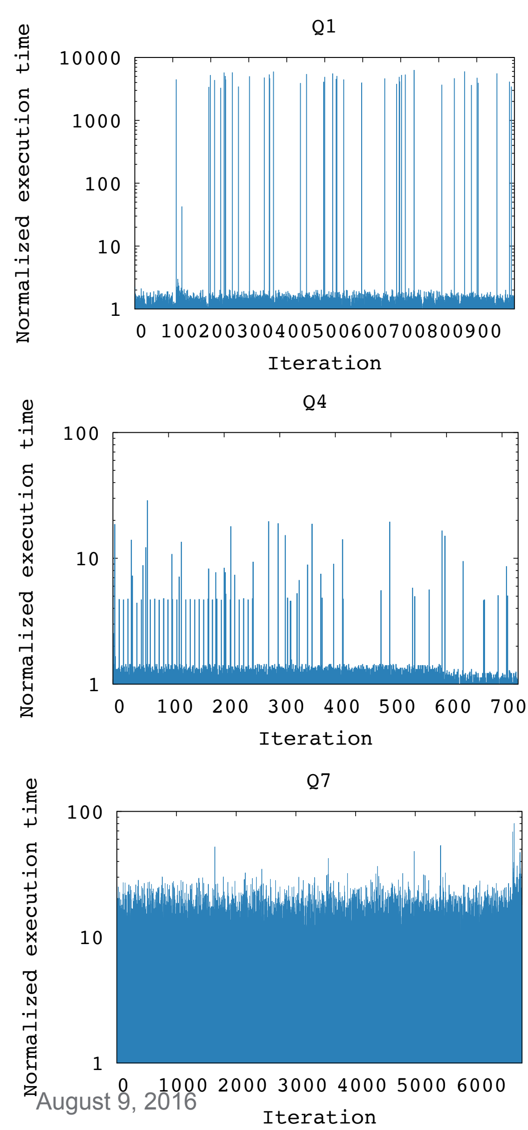
- Conventional High Level Synthesis (targeted to Digital Signal Processing):
  - Instruction Level Parallelism, uses the Finite State Machine with Datapath Model (FSMD) with a centralized controller
  - Simple memory abstraction (one port/one memory space), regular memory accesses
- Accelerating irregular applications (and RDF queries using graph methods)
  - Support for task parallelism
  - Advanced memory subsystem: support for large, multi-ported (parallel), shared memories, fine grained accesses, and synchronization
- Solutions:
  - Parallel distributed Controller (PC) – allows controlling an array of parallel accelerators (i.e., “hardware tasks”) with token passing mechanisms
  - Memory Interface Controller (MIC) - allows supporting multi-ported shared memory with dynamic address resolution and atomic memory operations

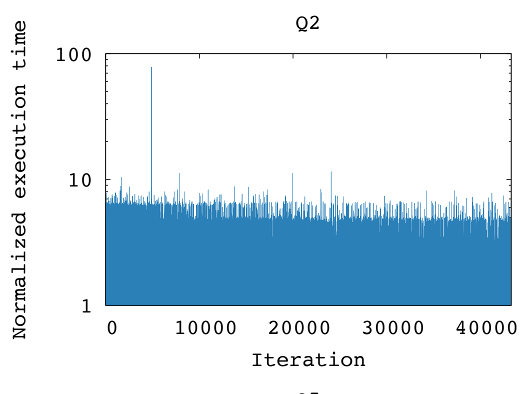


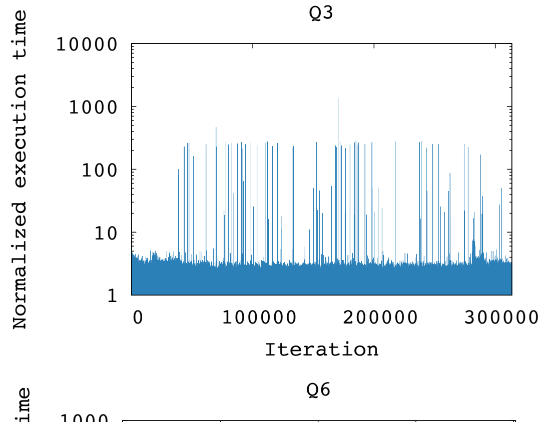

[V. G. Castellana, M. Minutoli, A. Morari, A. Tumeo, M. Lattuada, F. Ferrandi: High Level Synthesis of RDF Queries for Graph Analytics. ICCAD 2015]

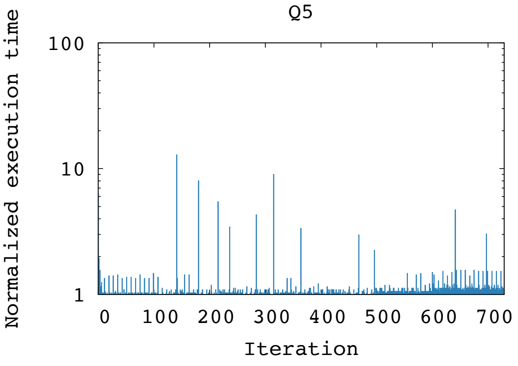
[M. Minutoli, V. G. Castellana, A. Tumeo: High-Level Synthesis of SPARQL queries. SC15 poster]

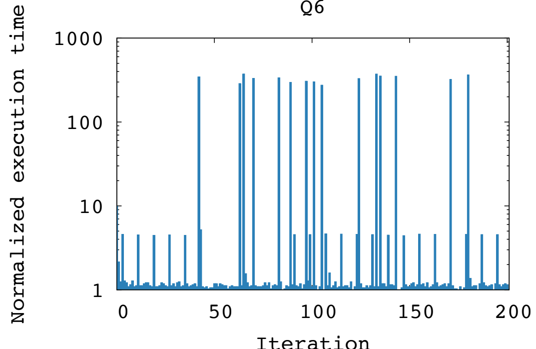
# Load Unbalancing in Queries



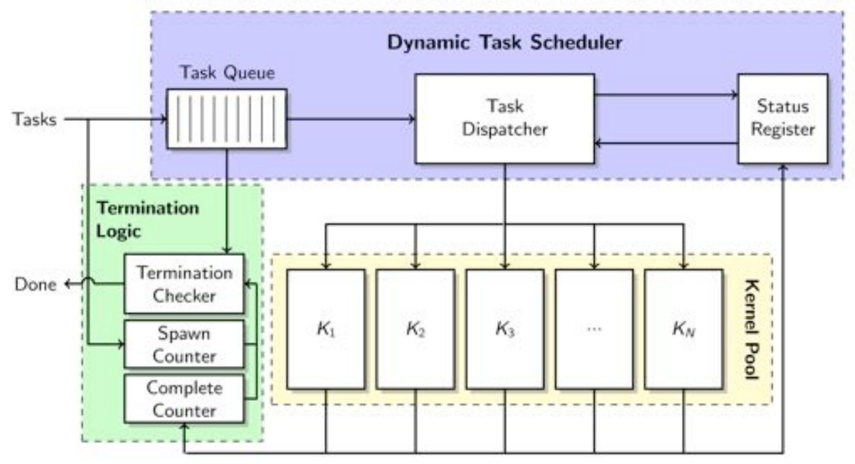




- Profiled the Lehigh University Benchmark (LUBM) with 5,309,056 triples (LUBM-40)
- Nested loops performing the search for the graph patterns of Queries Q1-Q7
- Some iterations last order of magnitude more than others

# Dynamic Task Scheduler

- The PC supports a block based fork-join parallel model
  - Each group of task executing on the kernel pool must terminate before a new one is launched
- The Dynamic Task scheduler launches a new task as soon as an accelerator (kernel) is available




[Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo, Marco Lattuada, Fabrizio Ferrandi: Efficient Synthesis of Graph Methods: A Dynamically Scheduled Architecture. ICCAD 2016]

- The Task Queue stores tasks ready for execution
- The Status Register keeps track of resource availability
- The Task Dispatcher issues the tasks
- The Termination Logic checks that all tasks have been used

# Experimental Evaluation

LUBM-1 (100k triples)

| | Single Acc. # Cycles | Parallel Controller # Cycles | Dynamic Scheduler # Cycles | Speedup | |
|---|---|---|---|---|---|
| | | | | Single Acc. | Parallel Controller |
| Q1 | 5,339,286 | 5,176,116 | 5,129,902 | 1.04 | 1.01 |
| Q2 | 141,022 | 54,281 | 50,997 | 2.77 | 1.06 |
| Q3 | 5,824,354 | 1,862,683 | 1,805,731 | 3.23 | 1.03 |
| Q4 | 63,825 | 42,851 | 19,928 | 3.20 | 2.15 |
| Q5 | 33,322 | 13,442 | 9,016 | 3.70 | 1.49 |
| Q6 | 674,951 | 340,634 | 197,894 | 3.41 | 1.72 |
| Q7 | 1,700,170 | 694,225 | 492,280 | 3.45 | 1.41 |

LUBM-40 (5M triples)

| | Single Acc. # Cycles | Parallel Controller # Cycles | Dynamic Scheduler # Cycles | Speedup | |
|---|---|---|---|---|---|
| | | | | Single Acc. | Parallel Controller |
| Q1 | 1,082,526,974 | 1,001,581,548 | 287,527,463 | 3.76 | 3.48 |
| Q2 | 7,359,732 | 2,801,694 | 2,672,295 | 2.75 | 1.05 |
| Q3 | 308,586,247 | 98,163,298 | 95,154,310 | 3.24 | 1.03 |
| Q4 | 63,825 | 42,279 | 19,890 | 3.21 | 2.13 |
| Q5 | 33,322 | 13,400 | 8,992 | 3.71 | 1.49 |
| Q6 | 682,949 | 629,671 | 199,749 | 3.42 | 3.15 |
| Q7 | 85,341,784 | 35,511,299 | 24,430,557 | 3.49 | 1.45 |

- Dynamic Scheduling always provides higher performance
- In the majority of cases, speed ups are over 3 (with 4 accelerators)
- The design is also more area efficient: higher speed up than area overhead (also w.r.t. parallel controller)

# A Dynamically Scheduled Architecture for the Synthesis of Graph Methods

Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo, Marco Lattuada and Fabrizio Ferrandi

Pacific Northwest
NATIONAL LABORATORY
*Proudly Operated by* **Battelle** *Since 1965*

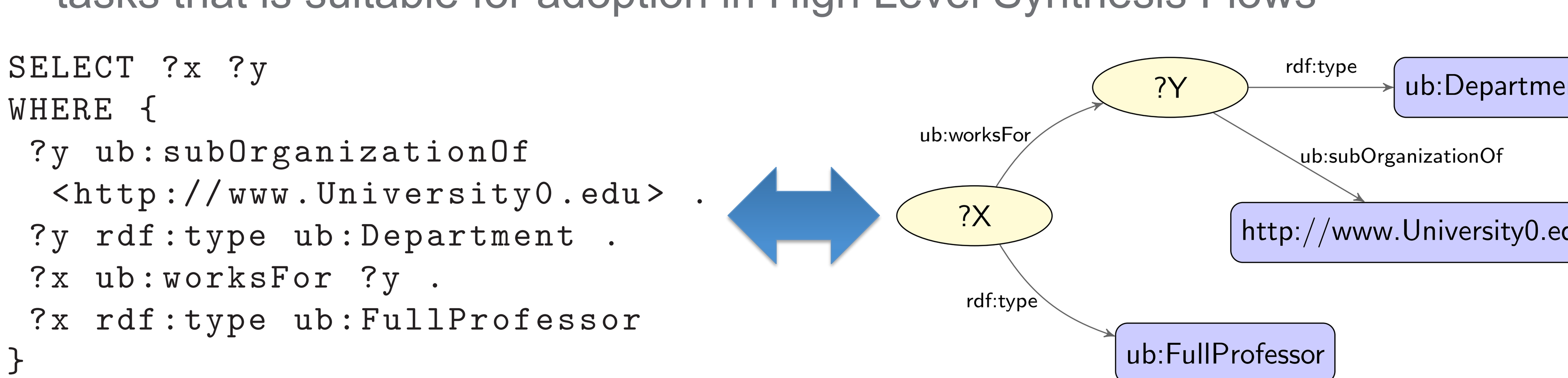
## Introduction

The Resource Description Framework (RDF) is a standard data model that represent data as triples (subject-predicate-object).

**RDF databases:**
- directly map to directed labeled graphs
- can be queried using SPARQL

SPARQL queries can be translated into Graph Pattern Matching methods that are intrinsically irregular in their behavior
- The execution is highly **data-dependent**
- At the same time they are characterized by high level of **data-parallelism**

**Our Contributions:**
- We analyze the behavior of LUBM queries to understand how the load unbalance between tasks affects the execution
- We propose an architecture template to tolerate the unbalancing between tasks that is suitable for adoption in High Level Synthesis Flows

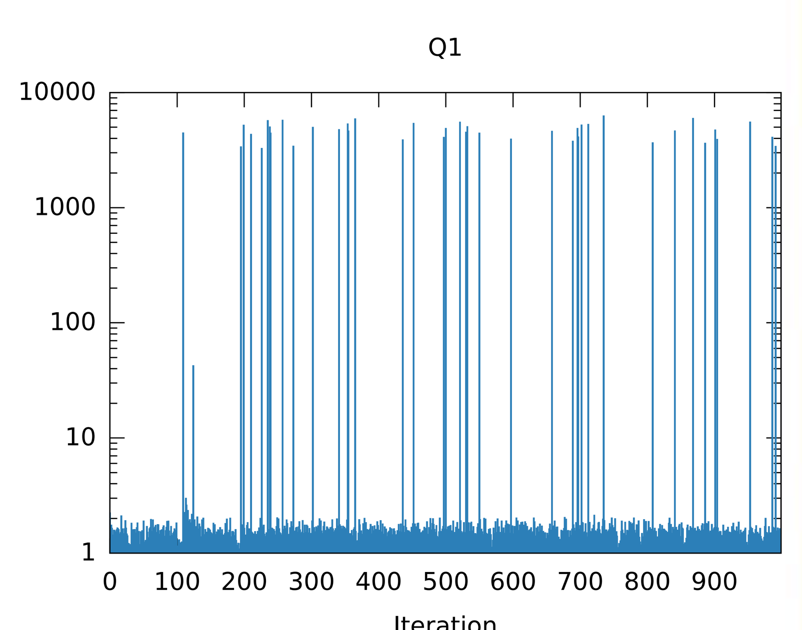


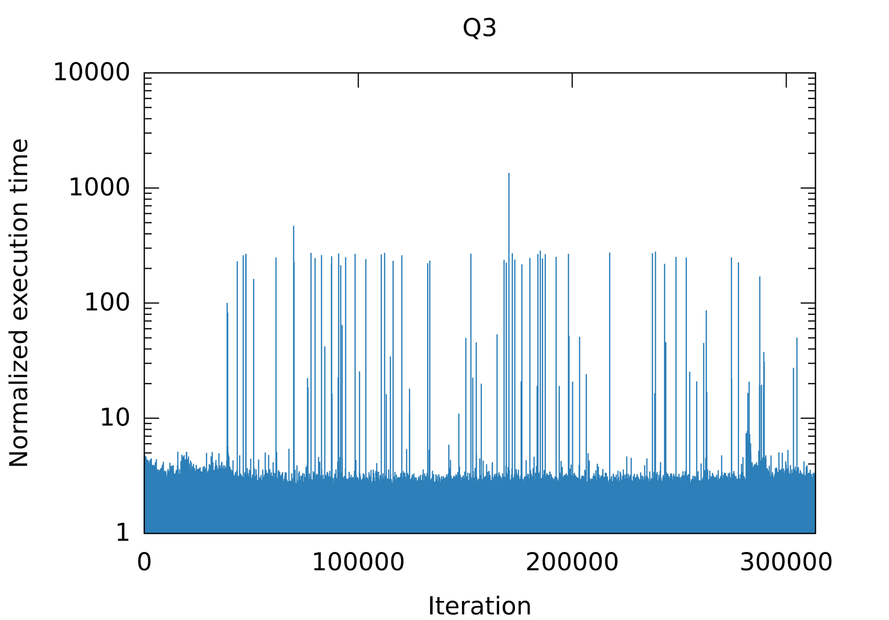

## Query Execution Analysis

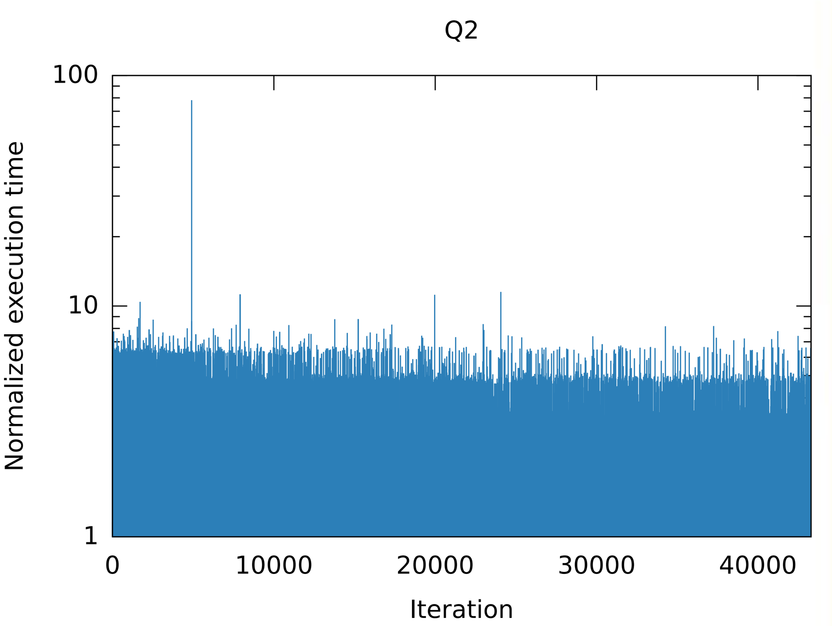
We consider 3 queries from LUBM (Q1-Q3):
- Input graph: LUBM-40 (5,309,056 triples)

The execution time of each outer loop iteration can vary of few order of magnitude.

Forking and joining tasks in groups can lead to resource under utilization when the workload between task is highly unbalanced (e.g., the group needs to wait for the slowest task: Q1 and Q3).



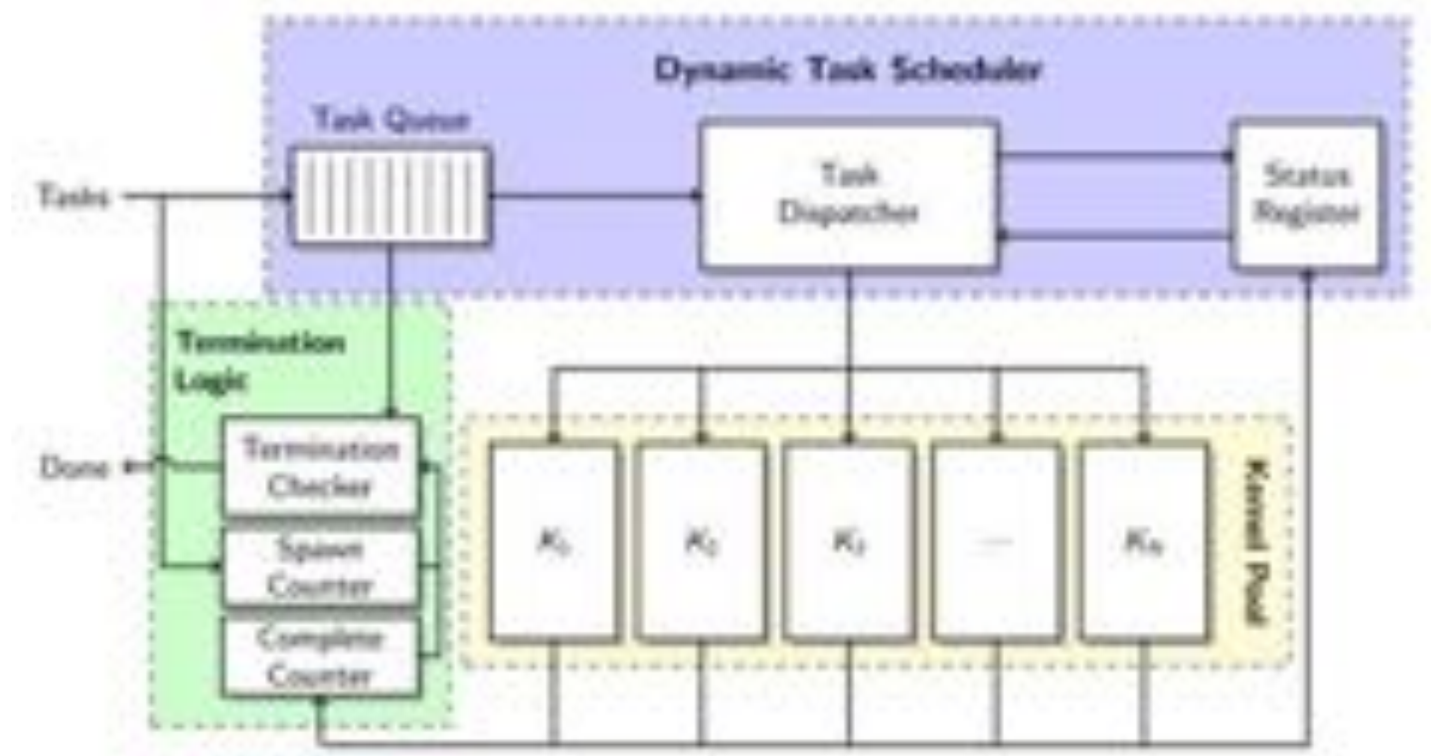




## The Dynamic Task Scheduling Architecture

**Dynamic Task Scheduler:**
- New tasks are inserted in the **Task Queue**
- The **Status Register** keeps track of resource availability
- When there are available resources, the **Task Dispatcher** pops a task from the queue and start its execution

**Kernels in the Pool:**
- are interfaced to the memory using a Hierarchical Memory Controller Interface supporting atomic memory operations.
- notify the **Status Register** when they are ready to accept another task.




**Termination Logic:**
- The **Spawn Counter** records the number of spawned tasks (pushed into the queue)
- The **Complete Counter** registers the number of tasks consumed by the kernels
- The **Termination Checker** monitors the status of the Task Queue and the two counters to verify the Termination Condition and to assert the done signal accordingly

**Termination Condition:**
When the two counters are equal and the Task Queue is empty all the Tasks that have entered the queue have been consumed.

## Experimental Evaluation

The architecture implementing the **Dynamic Task Scheduling** shows a **speed up** over the Serial implementation between **2.75** and **3.76**.

The **area overhead** is between **1.72-1.94 (LUTs)** and **1.62-1.95 (Slices)**.

The memory profiling shows that with 8 kernels the architecture is able to use 3 (out of 4) memory ports for the 80% of the computation time.

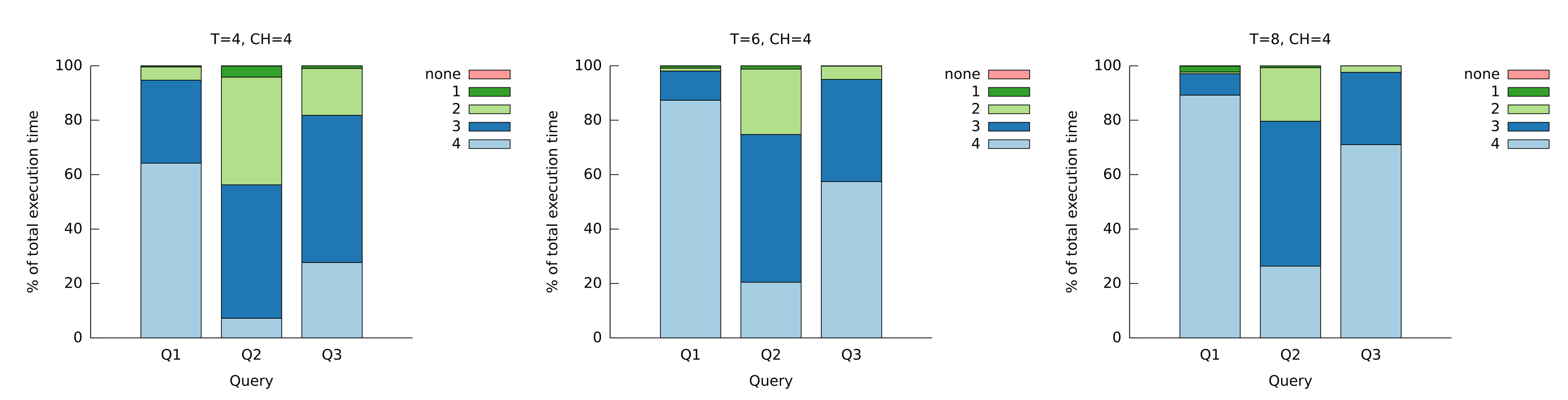
Increasing the number of kernels over 8 shows diminishing returns when the number of memory channels is fixed to 4.

| | Serial | | | | Dynamic Scheduler | | | | Area Overhead over Serial | | SpeedUp |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | LUTs | Slices | Latency (#) | Max. Freq. | LUTs | Slices | Latency (#) | Max. Freq. | LUTs | Slices | |
| **Q1** | 5,600 | 1,802 | 1,082,526,974 | 130.34MHz | 10,844 | 3,503 | 287,527,463 | 113.60MHz | 1.94 | 1.94 | 3.76 |
| **Q2** | 2,690 | 824 | 7,359,732 | 143.66MHz | 4,636 | 1,335 | 2,672,295 | 132.87MHz | 1.72 | 1.62 | 2.75 |
| **Q3** | 5,525 | 1,775 | 308,586,247 | 121.27MHz | 10,664 | 3,467 | 95,154,310 | 116.92MHz | 1.93 | 1.95 | 3.24 |

Comparison between a Serial Architecture and one implementing Dynamic Scheduling (T=4)

| | T=6, CH=4 | | | | T=8, CH=4 | | | |
|---|---|---|---|---|---|---|---|---|
| | LUTs | Slices | Latency | Max. Freq | LUTs | Slices | Latency | Max. Freq |
| **Q1** | 15,305 | 4,822 | 268,093,088 | 111.58MHz | 20,286 | 6,469 | 268,491,462 | 104.08MHz |
| **Q2** | 6,507 | 1,942 | 2,355,699 | 113.45MHz | 8,429 | 2,381 | 2,268,763 | 112.47MHz |
| **Q3** | 15,259 | 4,943 | 83,327,993 | 106.19MHz | 20,078 | 6,486 | 79,649,000 | 102.67MHz |

Architecture implementing Dynamic Scheduling with T=6 and T=8




Memory profiling of the architecture implementing the Dynamic Scheduling

**Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo**
Pacific Northwest National Laboratory
P.O. Box 999, MS-IN: J4-30, Richland, WA 99352 USA
{marco.minutoli, vitoGiovanni.Castellana, antonino.tumeo}@pnnl.gov

**Marco Lattuada, Fabrizio Ferrandi**
DEIB – Politecnico di Milano
P.za Leonardo Da Vinci 32, 20132 Milano Italy
{marco.lattuada, fabrizio.ferrandi}@polimi.it