Pacific
Northwest
NATIONAL LABORATORY

Enabling the High-Level Synthesis of Data Analytics Accelerators

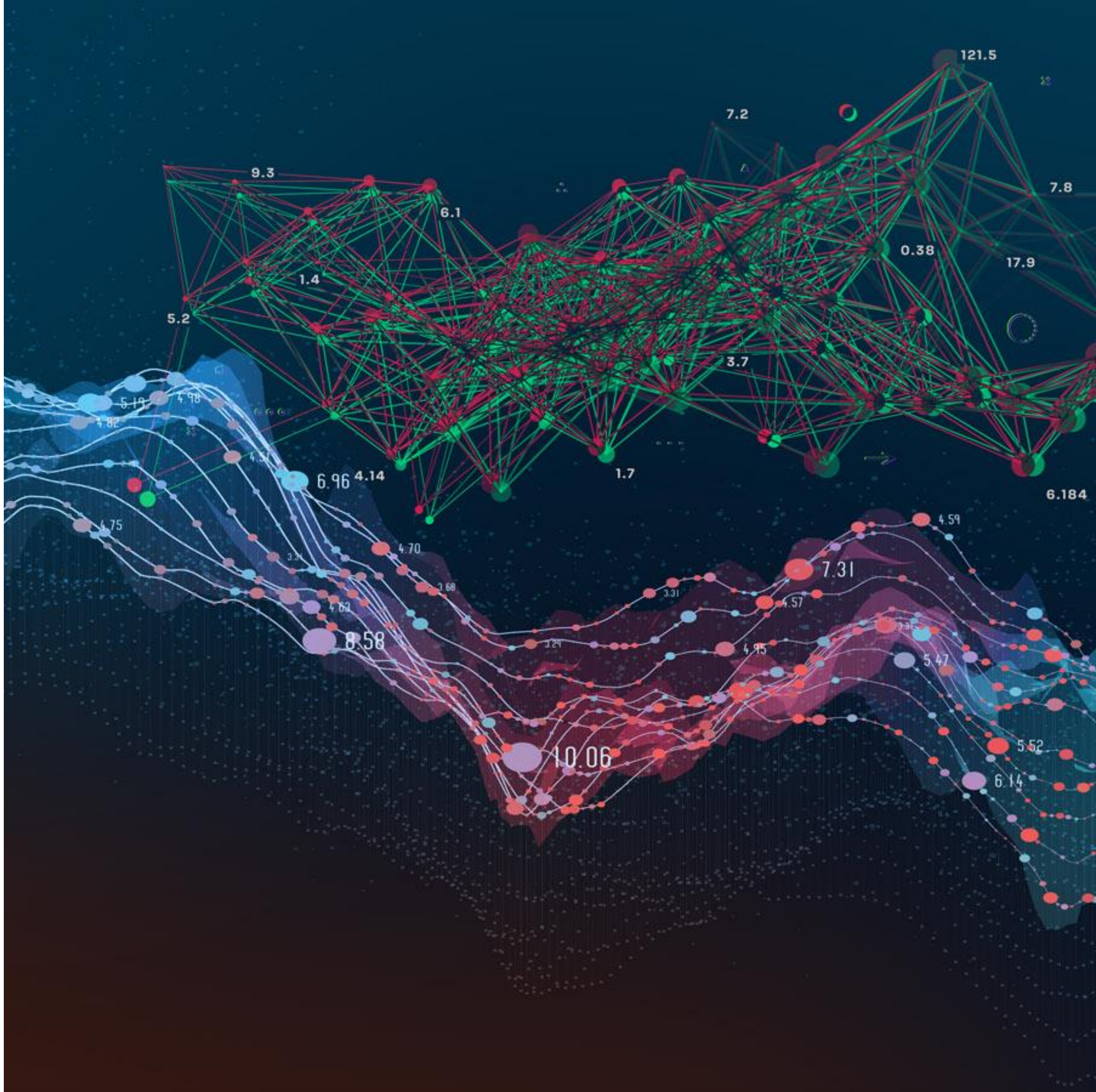
June 14, 2021

Marco Minutoli, Vito Giovanni Castellana,
Antonino Tumeo
PNNL, Richland, WA, USA

Serena Curzel, Michele Fiorito,
Fabrizio Ferrandi
Politecnico di Milano, Milano, Italy

U.S. DEPARTMENT OF
ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy



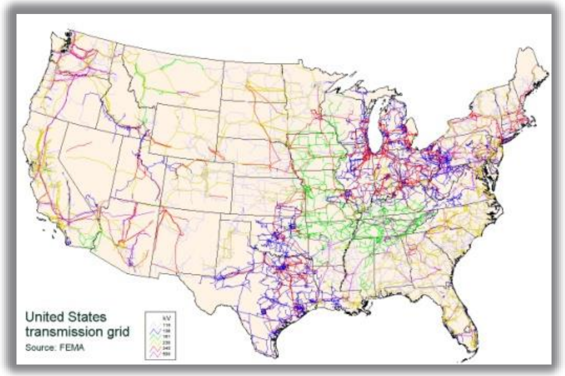
New generation of *irregular* HPC applications


Big Science


Bioinformatics


Community Detection

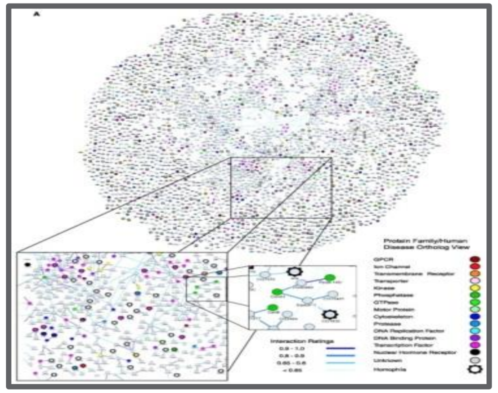

Graph Databases



Complex Networks

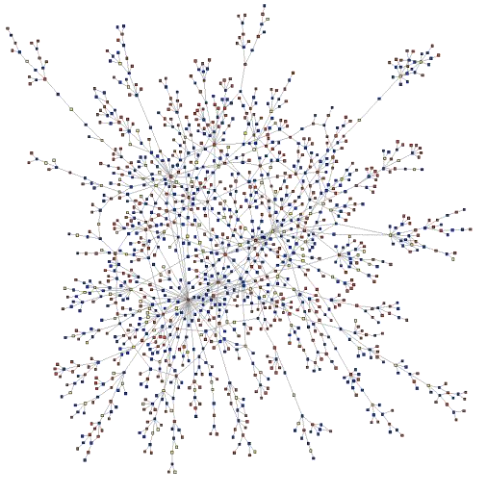
Knowledge Discovery


Language Understanding


Pattern Recognition

Pacific Northwest
NATIONAL LABORATORY

Massive Social Networks



facebook surpassed 1 billion active users

- Statistics
  - More than 1 billion active users, even more objects
  - Average user has 130 friends and connected to 80 community pages, groups, and events
- Graph characteristics
  - **Topology**: Interaction graph is low-diameter and has no good separators
  - **Irregularity**: Communities are not uniform in size
  - **Overlap**: individuals are members of one or more communities
- Sample queries:
  - **Allegiance switching**: identify entities that switch communities.
  - **Community structure**: identify the genesis and dissipation of communities
  - **Phase change**: identify significant change in the network structure

Definition of Irregular Applications

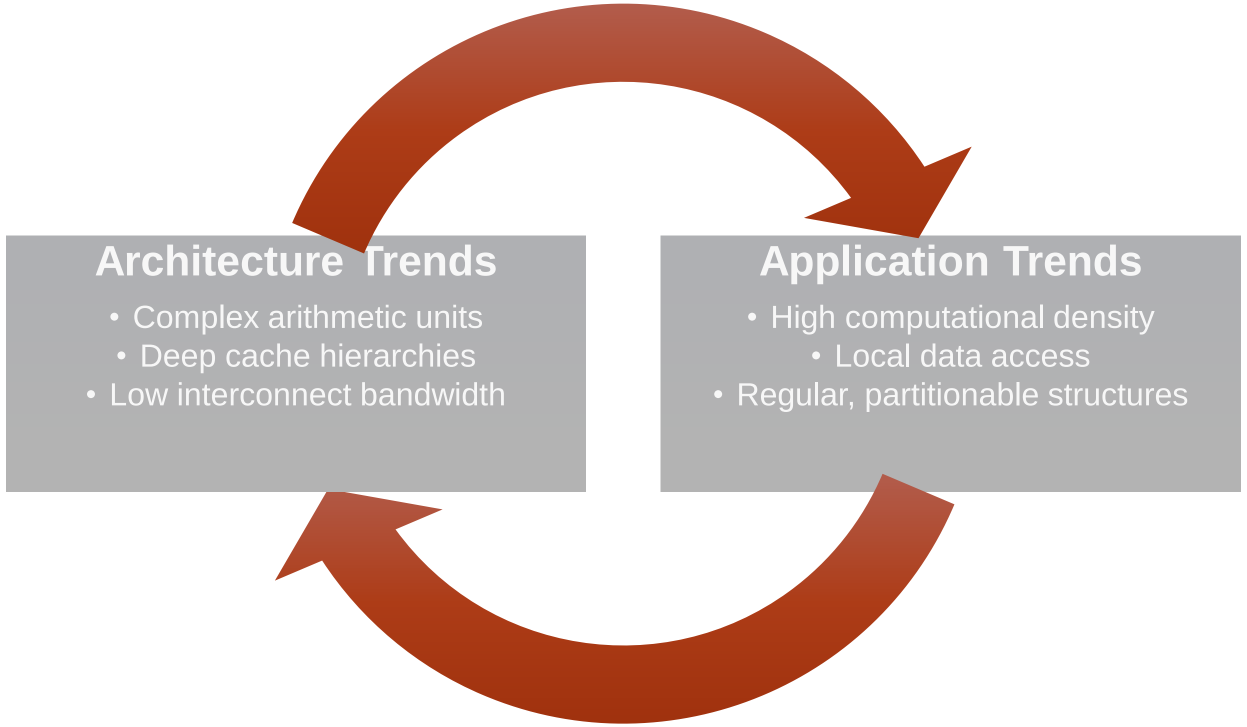
- Irregularity in data structures
    - Pointer- or linked-list based data structures such as graphs, unbalanced trees, unstructured grids
    - Very poor spatial and temporal locality
        - ✓ Unpredictable data accesses
        - ✓ Fine grained data accesses
- Irregularity in control
    - Divergent branches
        - ✓ If (vertex==x) z; else k
- Irregularity in communication patterns
    - Unpredictable and fine grained communication
    - A consequence of irregularity in data structures and in control

Additional Characteristics

- Very large datasets
    - Way more than what is currently available for single cluster nodes
    - Very difficult to partition in a balanced way
- Large amounts of parallelism (e.g., each vertex, each edge in the graph)
- Usually, high synchronization intensity
    - Concurrent activities accessing the same elements of the data structures
- Datasets may be dynamically updated

Self-reinforcing Trend of FLOP-computing

- The HPC community builds systems for scientific simulations.



Architecture Trends

- Complex arithmetic units
- Deep cache hierarchies
- Low interconnect bandwidth

Application Trends

- High computational density
- Local data access
- Regular, partitionable structures

- We need systems for *data analysis, discovery, and inferencing.*

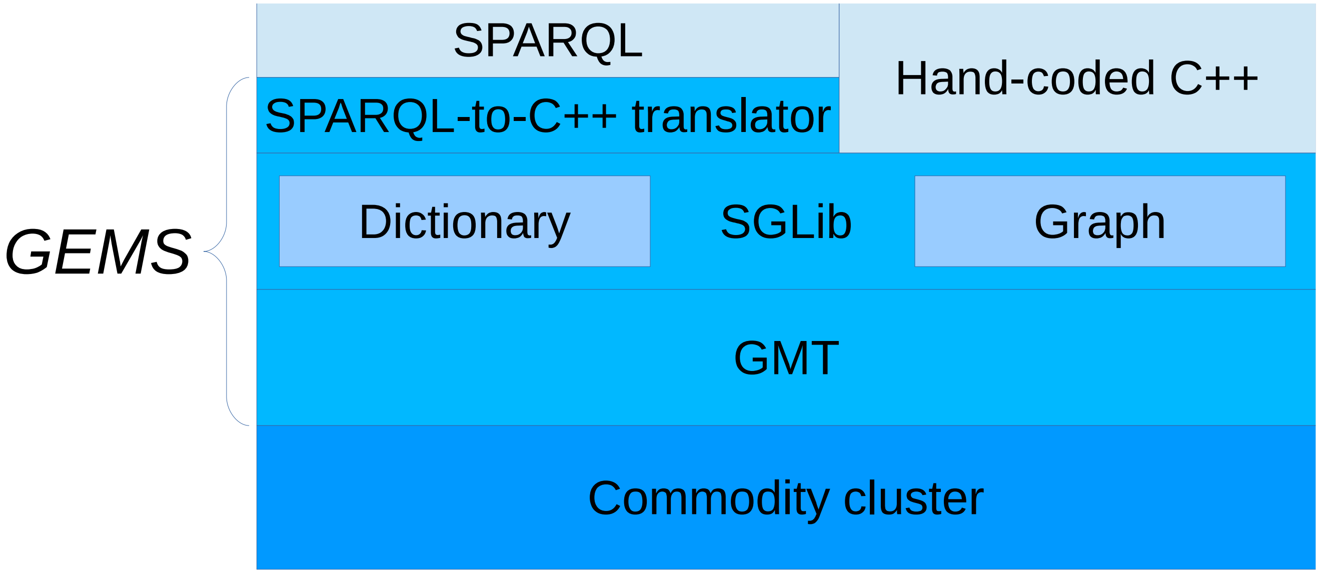
Desirable System Features for Irregular Applications

- Multithreading
  - Tolerate, rather than reduce (with caches/locality) data access latencies
- Global Address Space
  - No necessity to explicitly partition the dataset
- Fine-grained synchronization
  - May need to lock a single memory word
- Optimization: aggregation of fine-grained data accesses

Exemplar Irregular & Data Analytics Application: Graph Databases

- Promising solution to store large and heterogeneous datasets of these application fields
- Organize data in form of triples
  - Subject-predicate-object
  - Following the Resource Description Framework (RDF)
  - Set of triples represent a labeled, directed multigraph
- Queried through languages such as SPARQL
  - Fundamental operation is graph matching

Graph Engine for Multithreaded Systems (GEMS)


SPARQL

SPARQL-to-C++ translator

Hand-coded C++

Dictionary

SGLib

Graph

GMT

Commodity cluster

GEMS

[V.G. Castellana, A. Morari, J. Weaver, A. Tumeo, D. Haglin, O. Villa, J. Feo:In-Memory Graph Databases for Web-Scale Data. IEEE Computer 48(3): 24-35 (2015)]

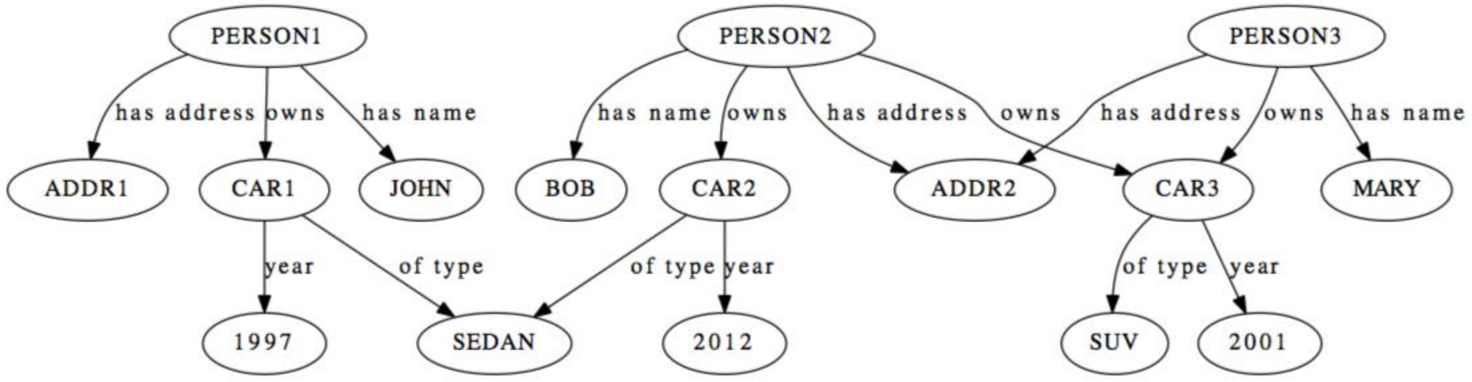
- A software stack that implements a RDF (graph) database on a homogeneous commodity cluster
- Uses graph methods
- Converts SPARQL to graph pattern matching routines in C++
- Employs a custom Runtime (GMT – Global Memory and Threading) which provides:
  - Lightweight software multithreading
  - Message aggregation
  - Global address space

Query example

- Return the names of all persons owning at least two cars, of which at least one is a SUV

```
PERSON1 has_name    JOHN  .
PERSON1 has_address ADDR1 .
PERSON1 owns        CAR1  .
CAR1    of_type     SEDAN .
CAR1    year        1997  .
PERSON2 has_name    BOB   .
PERSON2 has_address ADDR2 .
PERSON2 owns        CAR2  .
CAR2    of_type     SEDAN .
CAR2    year        2012  .
PERSON2 owns        CAR3  .
CAR3    of_type     SUV   .
CAR3    year        2001  .
PERSON3 has_name    MARY  .
PERSON3 has_address ADDR2 .
PERSON3 owns        CAR3  .
```

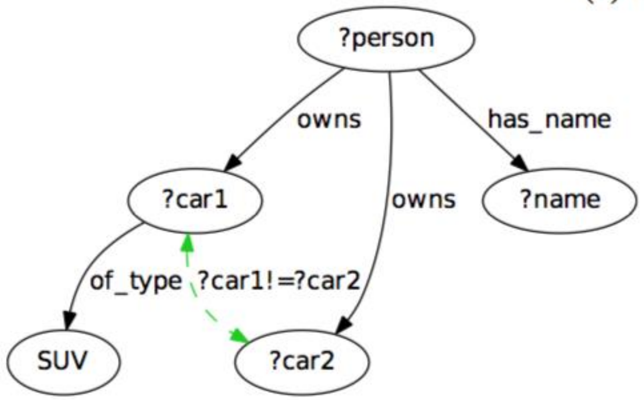
(a) Dataset in simplified N-Triples format



(b) RDF Graph

```
SELECT DISTINCT ?name
WHERE {
    ?person owns     ?car1 .
    ?person owns     ?car2 .
    ?person has_name ?name .
    ?car1   of_type  SUV   .
    FILTER(?car1 != ?car2)
}
```

(c) Simplified SPARQL query

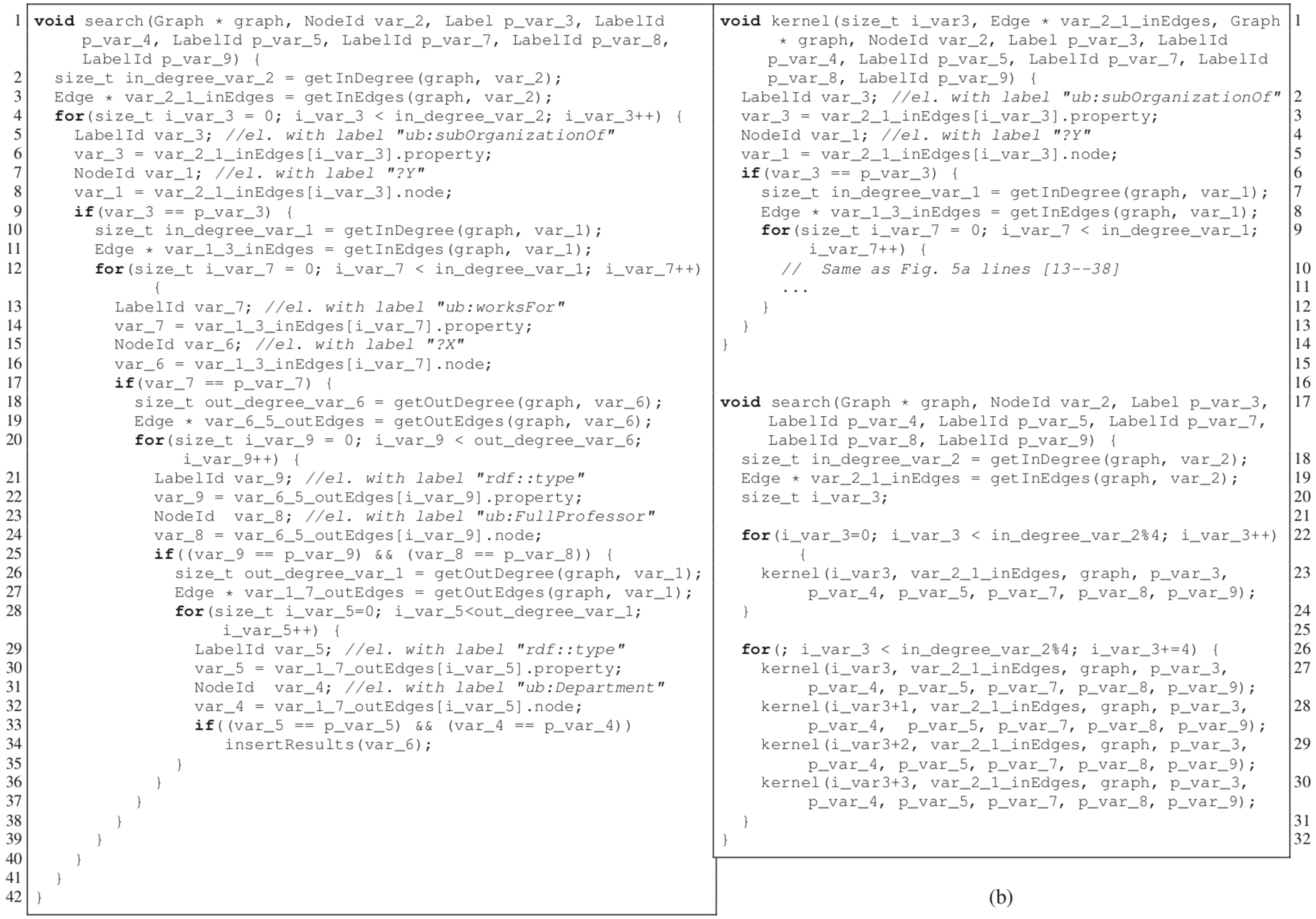


(d) Pattern graph

```
has_name = get_label("has_name")
of_type  = get_label("of_type")
owns     = get_label("owns")
suv      = get_label("SUV")
forall e1 in edges(*, of_type, suv)
  ?car1 = source_node(e1)
  forall e2 in edges(*, owns, ?car1)
    ?person = source_node(e2)
    forall e3 in edges(?person, owns, *)
      ?car2 = target_node(e3)
      if (?car1 != ?car2)
        forall e4 in edges(?person,has_name,*)
          ?name = target_node(e4)
          tuples.add(<?name>)
distinct(tuples)
```

(e) Pseudocode

Source Code Example



```
1  void search(Graph * graph, NodeId var_2, Label p_var_3, LabelId
        p_var_4, LabelId p_var_5, LabelId p_var_7, LabelId p_var_8,
        LabelId p_var_9) {
2    size_t in_degree_var_2 = getInDegree(graph, var_2);
3    Edge * var_2_1_inEdges = getInEdges(graph, var_2);
4    for(size_t i_var_3 = 0; i_var_3 < in_degree_var_2; i_var_3++) {
5      LabelId var_3; //el. with label "ub:subOrganizationOf"
6      var_3 = var_2_1_inEdges[i_var_3].property;
7      NodeId var_1; //el. with label "?Y"
8      var_1 = var_2_1_inEdges[i_var_3].node;
9      if(var_3 == p_var_3) {
10       size_t in_degree_var_1 = getInDegree(graph, var_1);
11       Edge * var_1_3_inEdges = getInEdges(graph, var_1);
12       for(size_t i_var_7 = 0; i_var_7 < in_degree_var_1; i_var_7++)
             {
13         LabelId var_7; //el. with label "ub:worksFor"
14         var_7 = var_1_3_inEdges[i_var_7].property;
15         NodeId var_6; //el. with label "?X"
16         var_6 = var_1_3_inEdges[i_var_7].node;
17         if(var_7 == p_var_7) {
18           size_t out_degree_var_6 = getOutDegree(graph, var_6);
19           Edge * var_6_5_outEdges = getOutEdges(graph, var_6);
20           for(size_t i_var_9 = 0; i_var_9 < out_degree_var_6;
                  i_var_9++) {
21             LabelId var_9; //el. with label "rdf::type"
22             var_9 = var_6_5_outEdges[i_var_9].property;
23             NodeId  var_8; //el. with label "ub:FullProfessor"
24             var_8 = var_6_5_outEdges[i_var_9].node;
25             if((var_9 == p_var_9) && (var_8 == p_var_8)) {
26               size_t out_degree_var_1 = getOutDegree(graph, var_1);
27               Edge * var_1_7_outEdges = getOutEdges(graph, var_1);
28               for(size_t i_var_5=0; i_var_5<out_degree_var_1;
                      i_var_5++) {
29                 LabelId var_5; //el. with label "rdf::type"
30                 var_5 = var_1_7_outEdges[i_var_5].property;
31                 NodeId  var_4; //el. with label "ub:Department"
32                 var_4 = var_1_7_outEdges[i_var_5].node;
33                 if((var_5 == p_var_5) && (var_4 == p_var_4))
34                    insertResults(var_6);
35               }
36             }
37           }
38         }
39       }
40     }
41   }
42 }
```

(a)

```
void kernel(size_t i_var3, Edge * var_2_1_inEdges, Graph      1
       * graph, NodeId var_2, Label p_var_3, LabelId
      p_var_4, LabelId p_var_5, LabelId p_var_7, LabelId
      p_var_8, LabelId p_var_9) {
  LabelId var_3; //el. with label "ub:subOrganizationOf"      2
  var_3 = var_2_1_inEdges[i_var_3].property;                 3
  NodeId var_1; //el. with label "?Y"                        4
  var_1 = var_2_1_inEdges[i_var_3].node;                     5
  if(var_3 == p_var_3) {                                     6
    size_t in_degree_var_1 = getInDegree(graph, var_1);      7
    Edge * var_1_3_inEdges = getInEdges(graph, var_1);       8
    for(size_t i_var_7 = 0; i_var_7 < in_degree_var_1;      9
         i_var_7++) {
      //  Same as Fig. 5a lines [13--38]                     10
      ...                                                    11
    }                                                        12
  }                                                          13
}                                                            14
                                                             15
                                                             16
void search(Graph * graph, NodeId var_2, Label p_var_3,      17
      LabelId p_var_4, LabelId p_var_5, LabelId p_var_7,
      LabelId p_var_8, LabelId p_var_9) {
  size_t in_degree_var_2 = getInDegree(graph, var_2);        18
  Edge * var_2_1_inEdges = getInEdges(graph, var_2);         19
  size_t i_var_3;                                            20
                                                             21
  for(i_var_3=0; i_var_3 < in_degree_var_2%4; i_var_3++)     22
       {
    kernel(i_var3, var_2_1_inEdges, graph, p_var_3,          23
         p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
  }                                                          24
                                                             25
  for(; i_var_3 < in_degree_var_2%4; i_var_3+=4) {           26
    kernel(i_var3, var_2_1_inEdges, graph, p_var_3,          27
         p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
    kernel(i_var3+1, var_2_1_inEdges, graph, p_var_3,        28
         p_var_4,  p_var_5, p_var_7, p_var_8, p_var_9);
    kernel(i_var3+2, var_2_1_inEdges, graph, p_var_3,        29
         p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
    kernel(i_var3+3, var_2_1_inEdges, graph, p_var_3,        30
         p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
  }                                                          31
}                                                            32
```

(b)

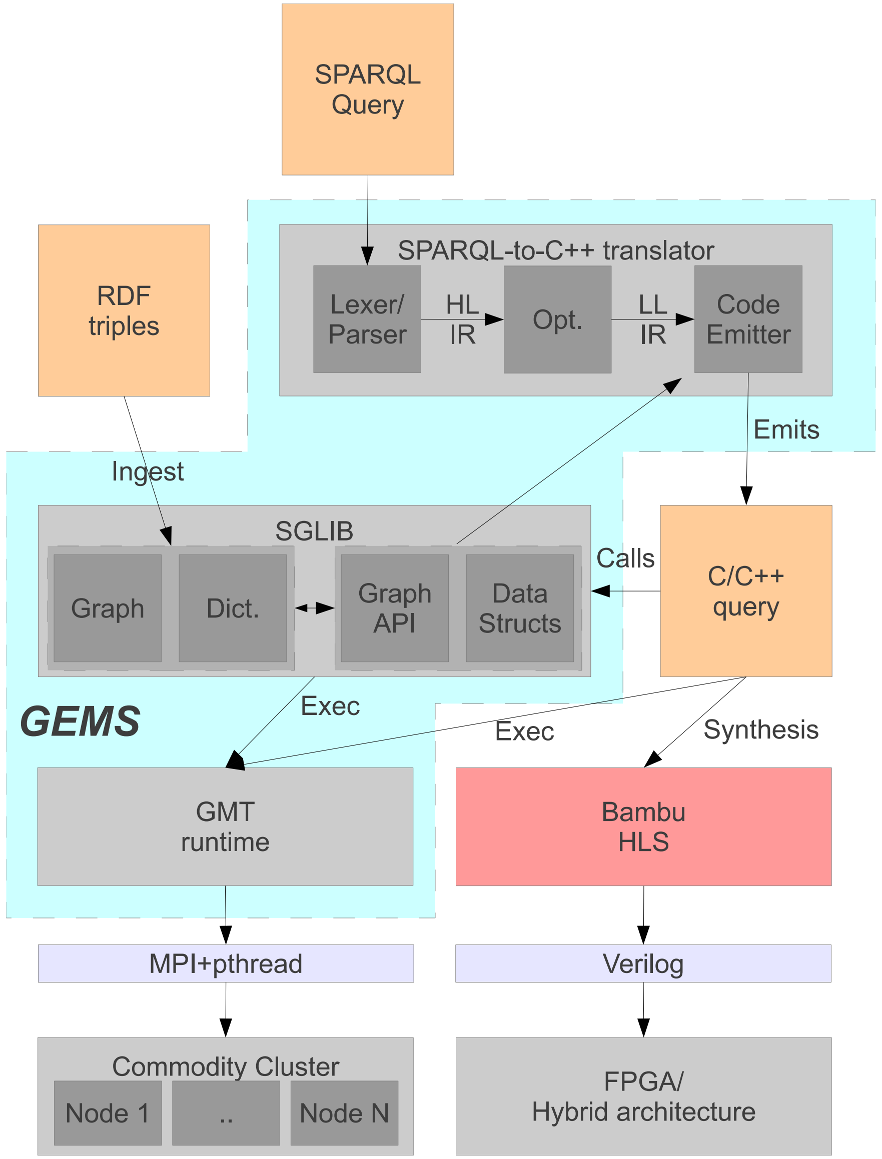
Fig. 5: Pseudo code for the pattern matching routines of example query Q6

Can we use FPGAs to accelerate Data Analytics and Irregular Applications?

- FPGAs provide an opportunity to implement application specific accelerators – fast, and power efficient
- Flexibility challenge: implementing accelerators in RTL (Register Transfer Level) languages is complex and time consuming
    - High-Level Synthesis (HLS): synthesizing RTL from descriptions in higher level languages (e.g., C)
    - However, conventional HLS tools typically target Digital Signal Processing algorithms, i.e., typical “regular” applications

- We can accelerate GEMS by:
    - Implementing some parts of the runtime on FPGA
    - Directly synthesizing queries (i.e., graph pattern matching routines)
        - ✓ Synthesis time is not a limitation: queries do not change often, datasets do

GEMS on FPGAs



- C code generated by GEMS SPARQL-to-C++ translator
- Code is then processed by Bambu, a High Level Synthesis tool from Politecnico di Milano
  - Heavily modified to support our new architectural templates
- Note: accelerating graph walks is a more complex problem than accelerating table 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]

Challenges for HLS of Irregular Applications

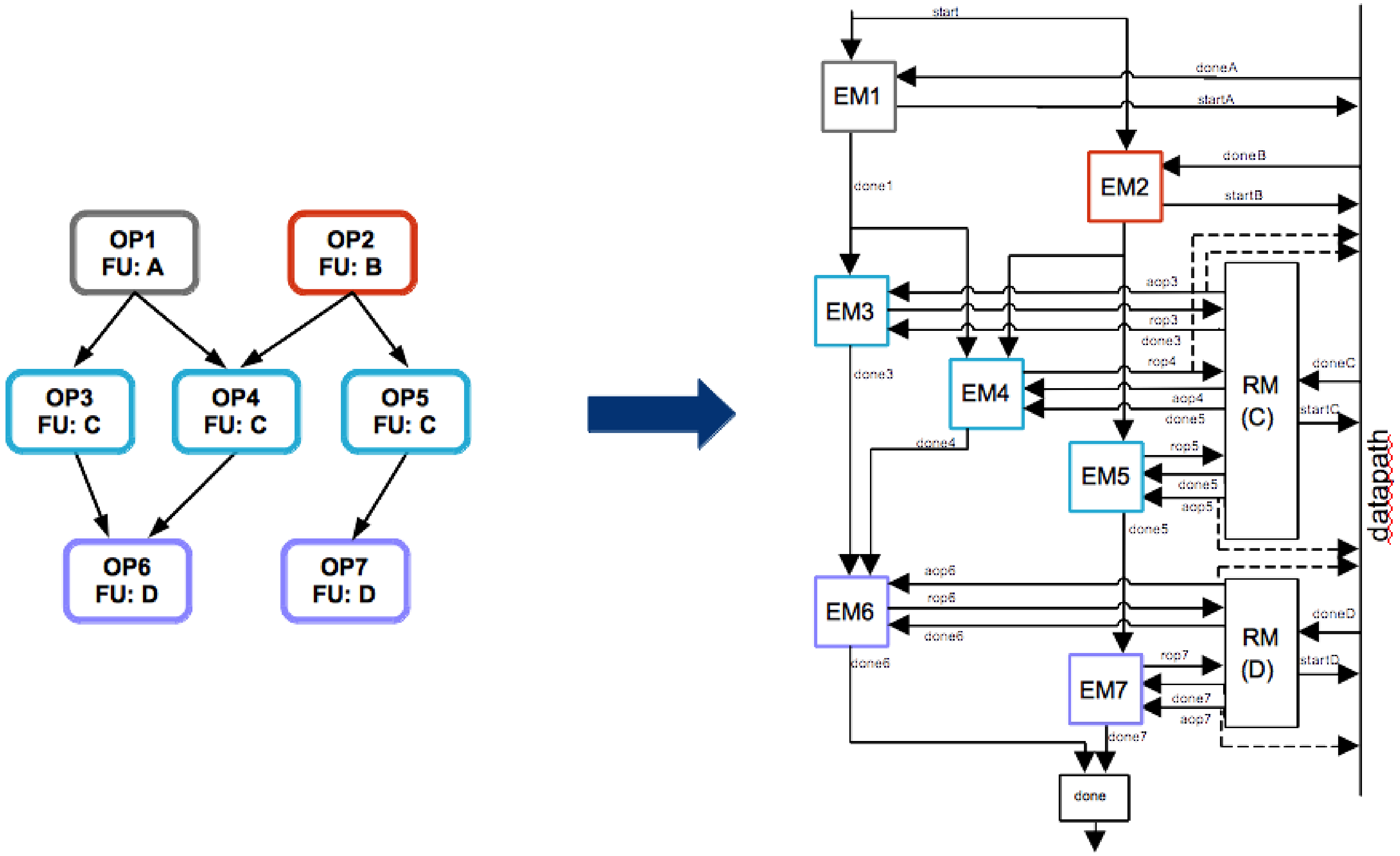
- **Challenge 1**: Coarse grained parallelism exploitation
    - Most HLS approaches focus on exploiting ILP and do not support TLP specifications (expressed through parallel programming APIs such as OpenMP, CUDA, OpenCL)
    - Concurrency and synchronization management
    - Target architecture design: HLS flows usually generate Finite State Machines with Datapaths, which are inherently serial
- **Challenge 2**: Support for complex memory subsystems
    - Dynamic resolution of memory addresses
    - Pointer-based data structures
    - Memory consistency and synchronization
    - Barriers, Atomic memory operations
    - Distributed/multi-ported memories

Solving Challenge 1: Parallel Distributed Controller

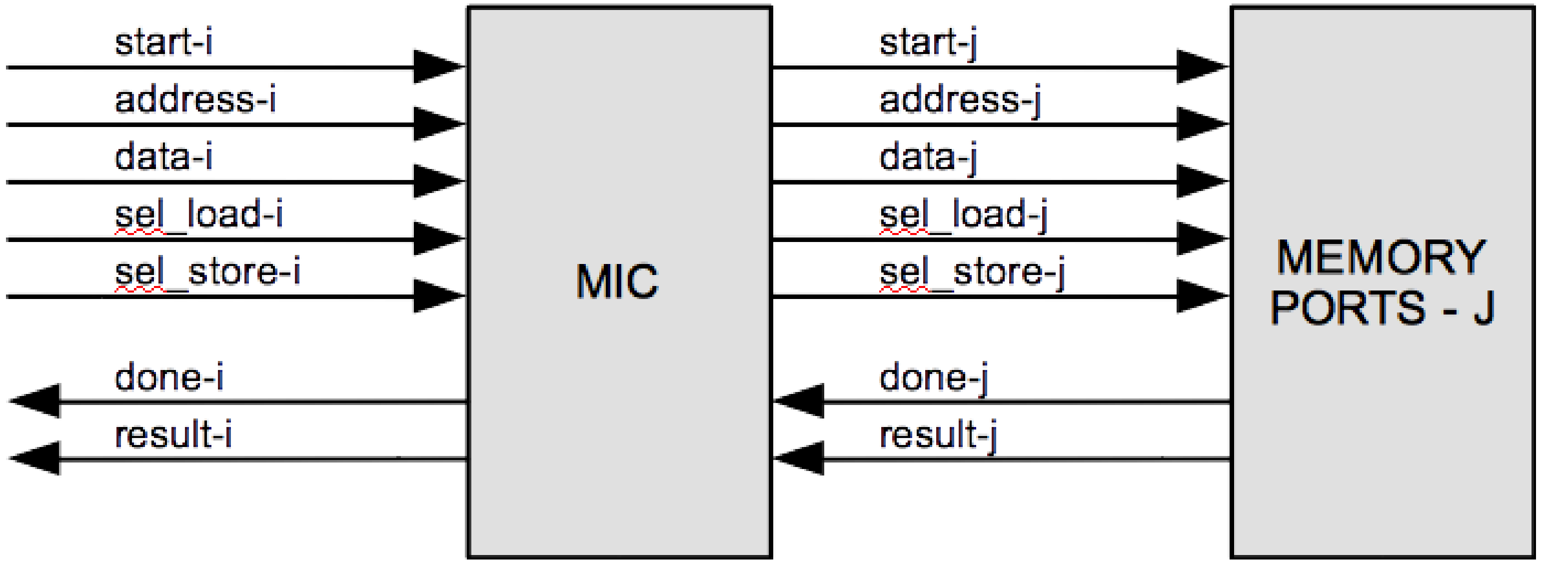
- From serial FSMD to a parallel distributed controller

- Architecture:
  - Set of communicating control elements, called Execution Managers (EMs)
    - ✓ Each EM establishes when an operation/task can start at **runtime**
    - ✓ Dynamic execution paradigm
  - Dedicated hardware (Resource Managers – RMs) for checking:
    - ✓ Satisfaction of dependence constraints
    - ✓ Resource availability
  - Natural support for variable latency operations/tasks
    - ✓ ASAP execution

Example of Parallel Distributed Controller



Solving Challenge 2: Hierarchical Memory Interface Controller (HMI)



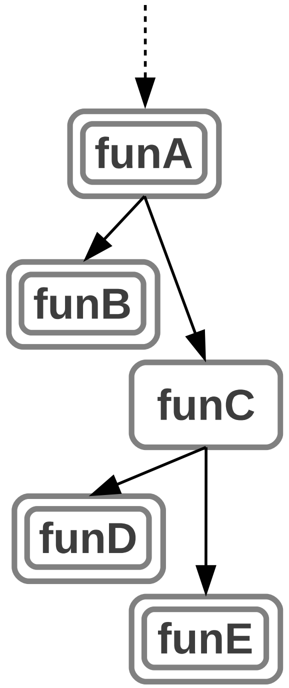
[V.G. Castellana, A. Tumeo, F. Ferrandi: An adaptive Memory Interface Controller for improving bandwidth utilization of hybrid and reconfigurable systems. DATE 2014.]

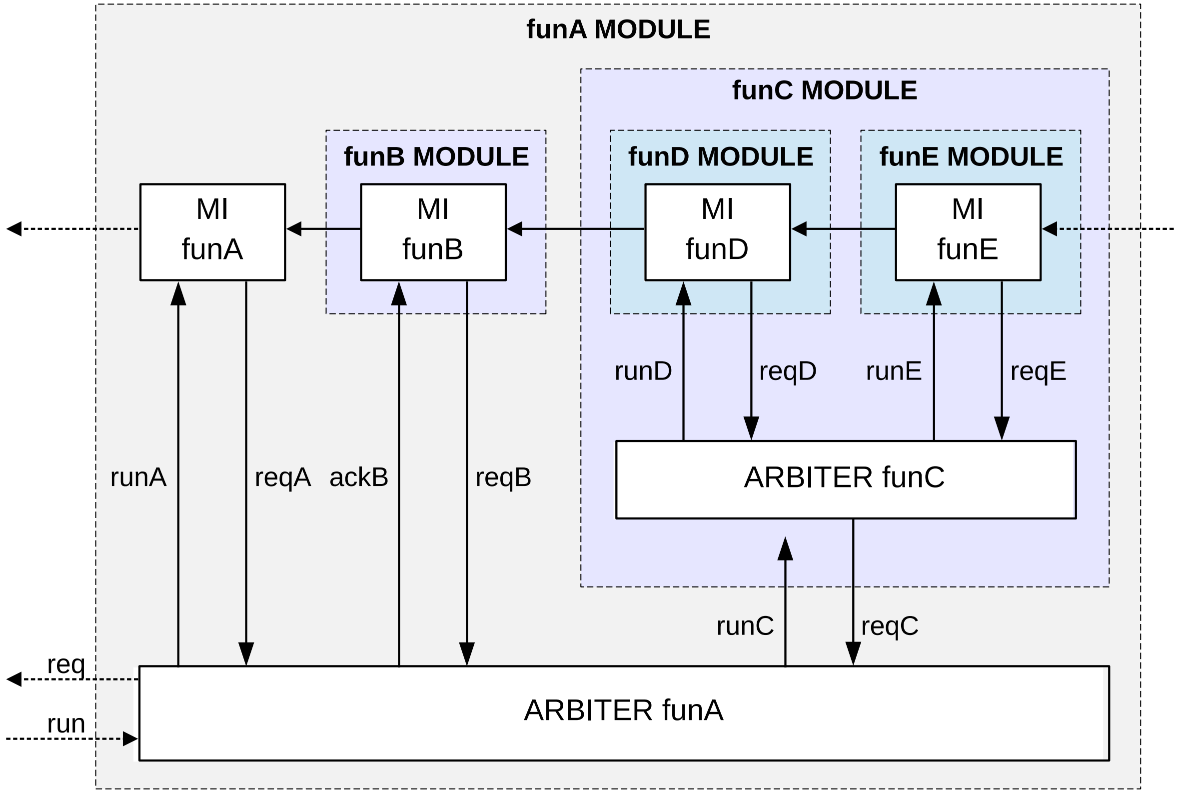
- Allows concurrent memory accesses on distributed/multi-ported shared memories
- Dynamically resolves memory addresses
- Manages concurrency and synchronization (supporting atomic operations)

HMI details

- Manages **concurrency**
  - Provides memory scrambling (solves hotspot)
  - Collects memory requests, and if their target addresses collide, it forwards them one at a time
  - Each memory port is managed by a dedicated arbiter
  - No delay penalties
- Manages **synchronization**
  - Directly implements **atomic operations** (e.g. atomic increment, compare and swap)
  - Inside the interface, through dedicated hardware
  - While running, atomic operations *lock* the associated memory port

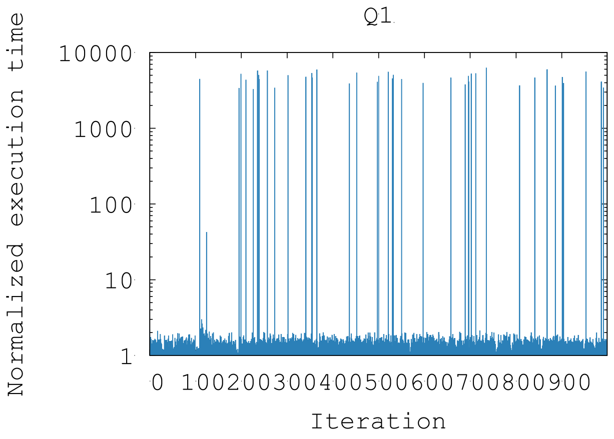
Hierarchical Approach to HLS

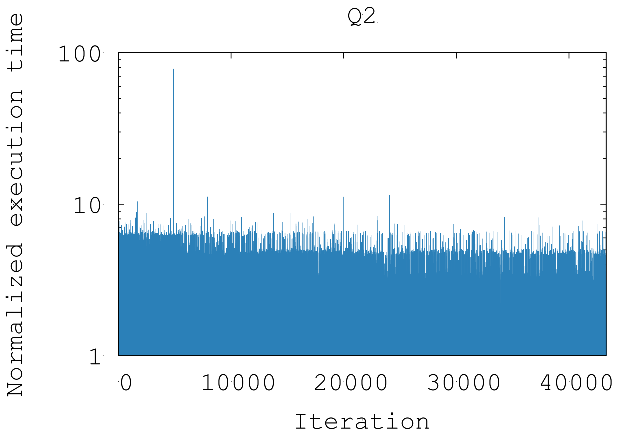


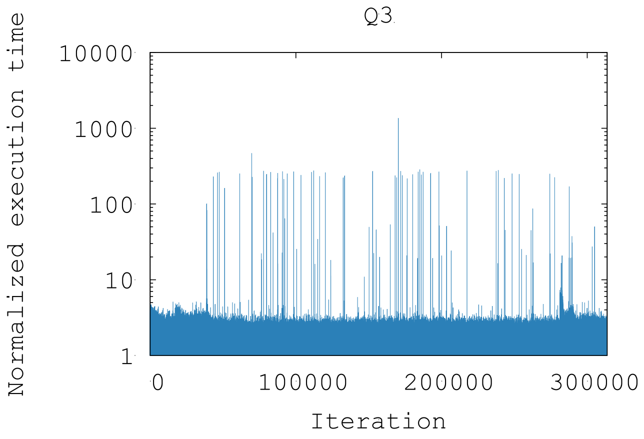


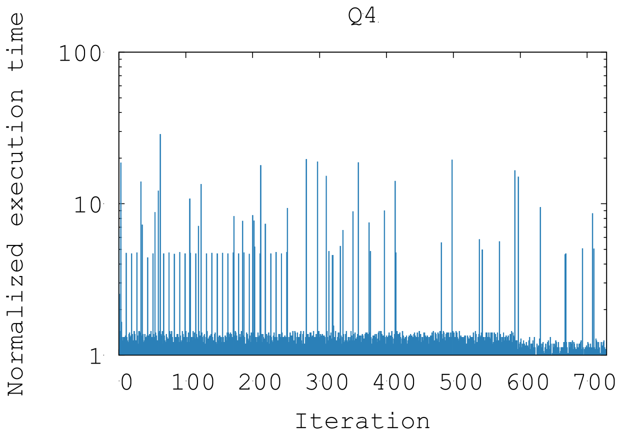
- A function is a module, and can contain other function modules
- Function modules can use parallel controller (e.g., launching iterations of a loop, where each iteration has a corresponding hardware kernel), or a FSMD module (e.g., the loop iteration itself)
- Allows generating accelerators for irregular codes with nested loops
  - Graph algorithms & queries

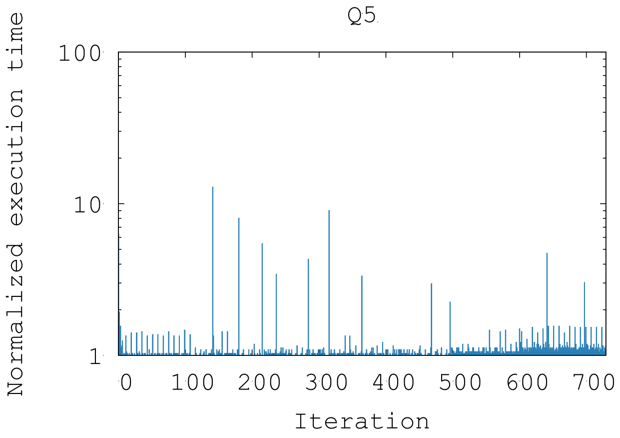
Load Unbalancing in Queries

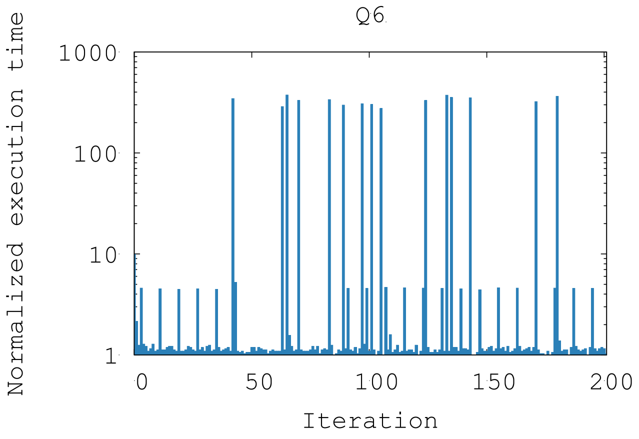


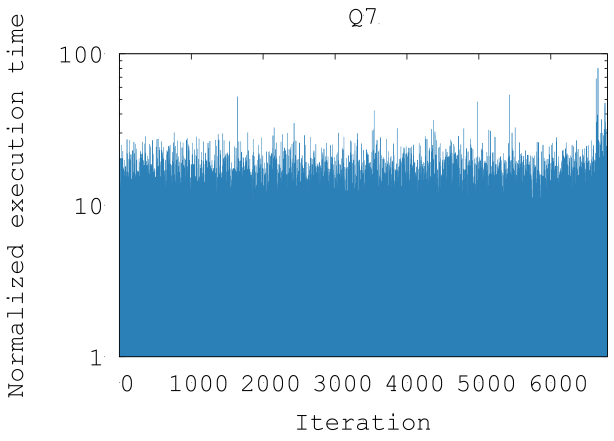






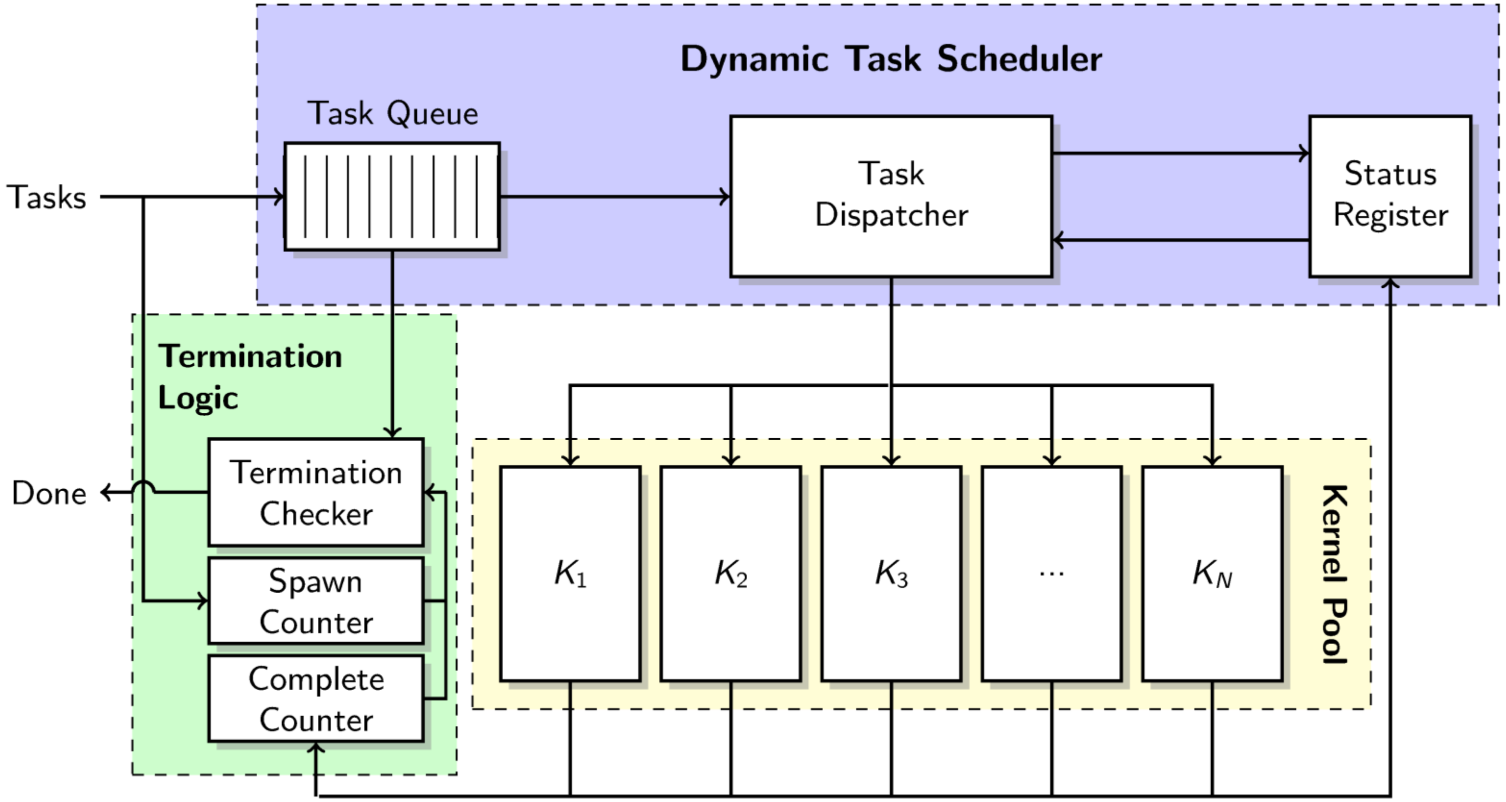






- Lehigh University Benchmark (reference benchmark for the semantic web)
- 5,309,056 triples (LUBM-40); Queries Q1-Q7

Dynamic Task Scheduler

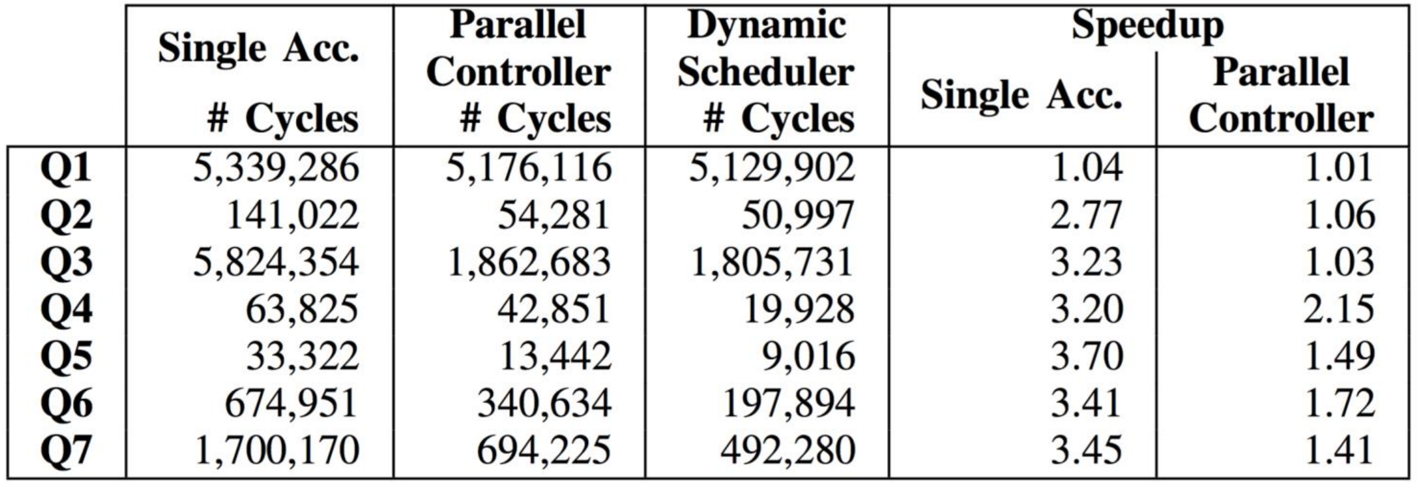


[Marco Minutoli, Vito Giovanni Castellana, Antonino Tumeo, Marco Lattuada, Fabrizio Ferrandi: Efficient Synthesis of Graph Methods: A Dynamically Scheduled. ICCAD 2016]

- 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

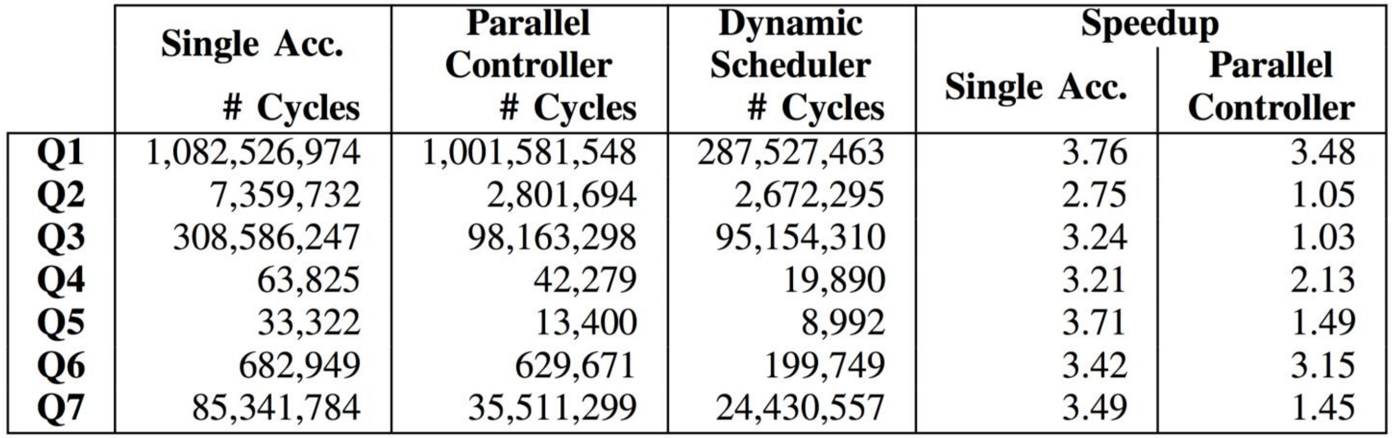
Experimental Evaluation

LUBM-1 (100k triples)



	Single Acc. # Cycles	Parallel Controller # Cycles	Dynamic Scheduler # Cycles	Speedup Single Acc.	Speedup 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)



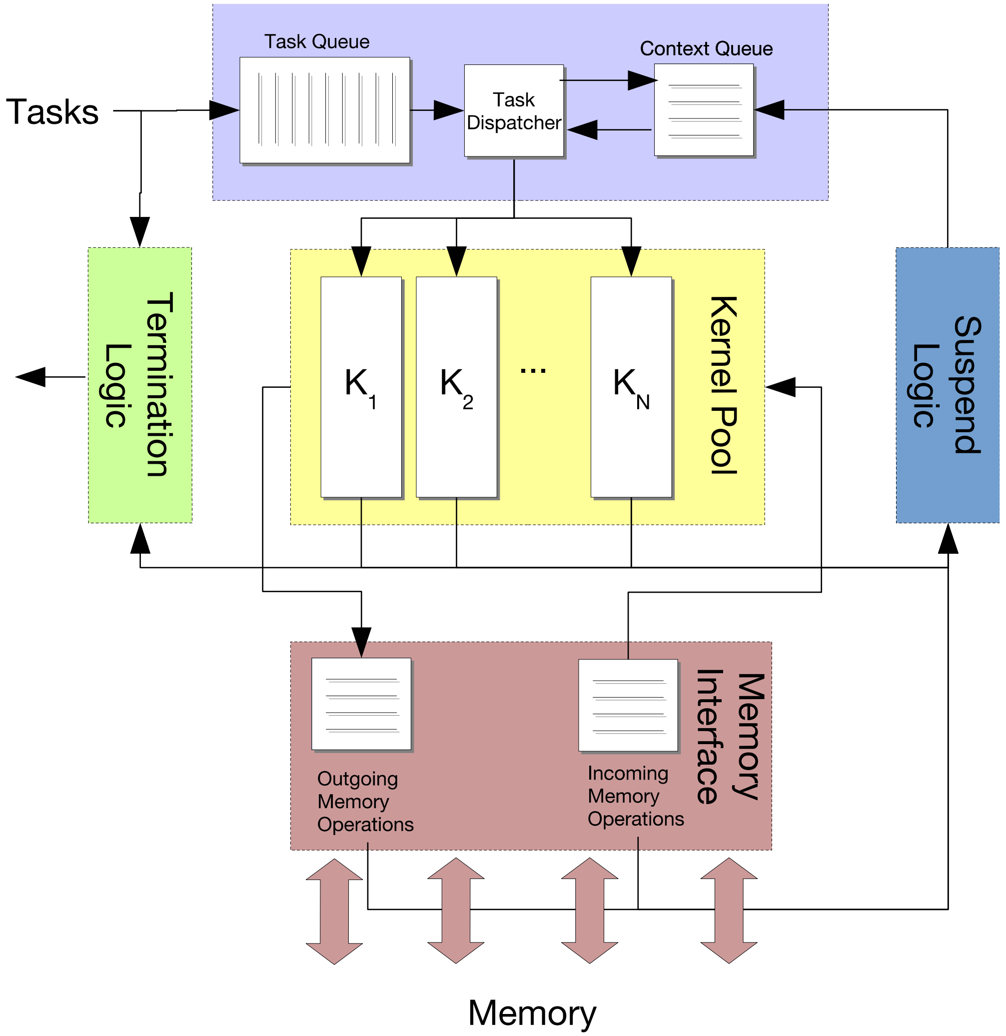
	Single Acc. # Cycles	Parallel Controller # Cycles	Dynamic Scheduler # Cycles	Speedup Single Acc.	Speedup Parallel Controller
Q1	1,082,526,974	1,001,581,548	287,527,463	3.76	3.48
Q2	7,359,732	2,801,694	2,672,295	2.75	1.05
Q3	308,586,247	98,163,298	95,154,310	3.24	1.03
Q4	63,825	42,279	19,890	3.21	2.13
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

- Architectures integrating dynamic scheduling vs. solutions only integrating PC+MIC
- 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)

Pacific Northwest
NATIONAL LABORATORY

Latest Extension: Temporal Multithreading

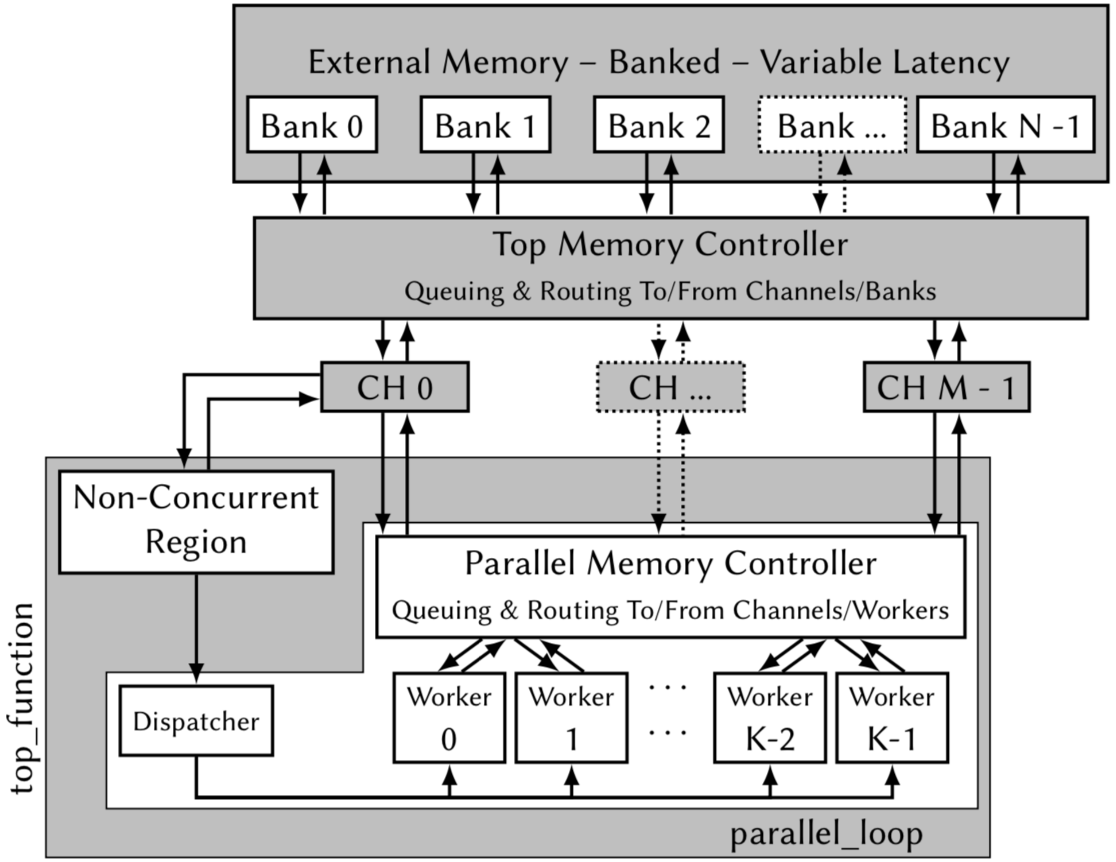
Svelto Methodology

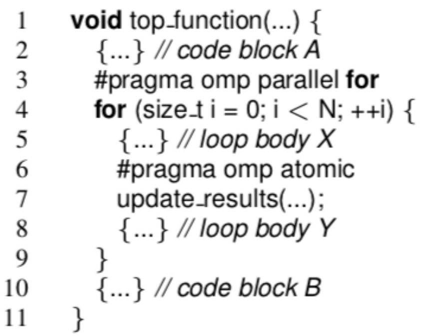


[M. Minutoli *et al.*, "Svelto: High-Level Synthesis of Multi-Threaded Accelerators for Graph Analytics," in *IEEE Transactions on Computers*, doi: 10.1109/TC.2021.3057860.]

- With Irregular Applications, we prefer to tolerate latency
  - Latency reduction through caches and localization ineffective
- Objective is to saturate the memory subsystem with parallel memory operations
  - Temporal, rather than spatial, multithreading provides interesting area/performance/utilization tradeoffs
- We extend the DTS design to perform context switching after memory operations
  - Memory interface further decoupled
  - Number of contexts is configurable

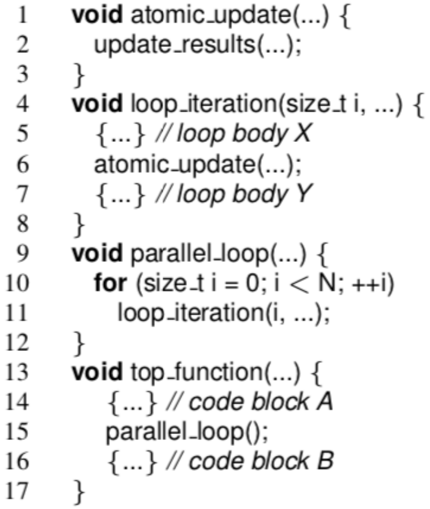
Architectural Template Mapping





```
void top_function(...) {
  {...} // code block A
  #pragma omp parallel for
  for (size_t i = 0; i < N; ++i) {
    {...} // loop body X
    #pragma omp atomic
    update_results(...);
    {...} // loop body Y
  }
  {...} // code block B
}
```

(a) Example of OpenMP application to be synthesized.

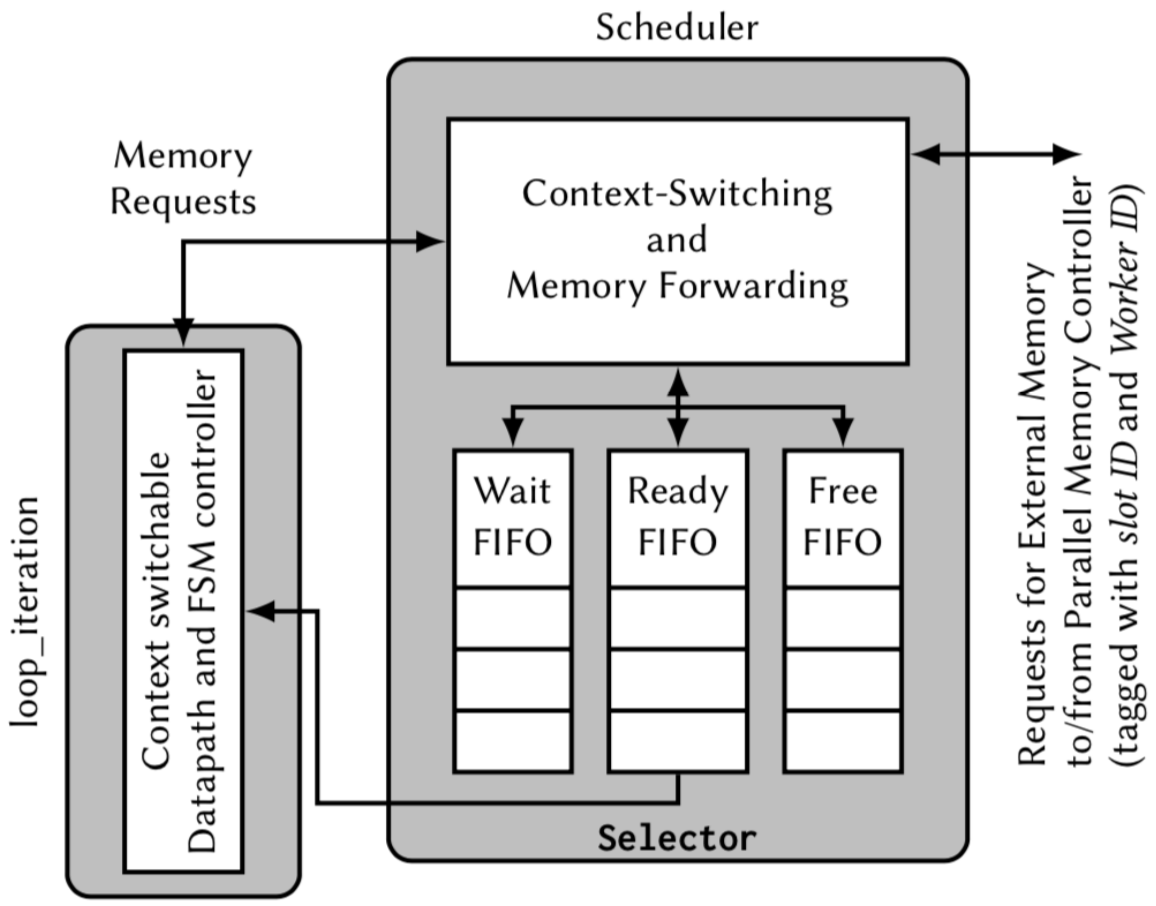


```
void atomic_update(...) {
  update_results(...);
}
void loop_iteration(size_t i, ...) {
  {...} // loop body X
  atomic_update(...);
  {...} // loop body Y
}
void parallel_loop(...) {
  for (size_t i = 0; i < N; ++i)
    loop_iteration(i, ...);
}
void top_function(...) {
  {...} // code block A
  parallel_loop();
  {...} // code block B
}
```

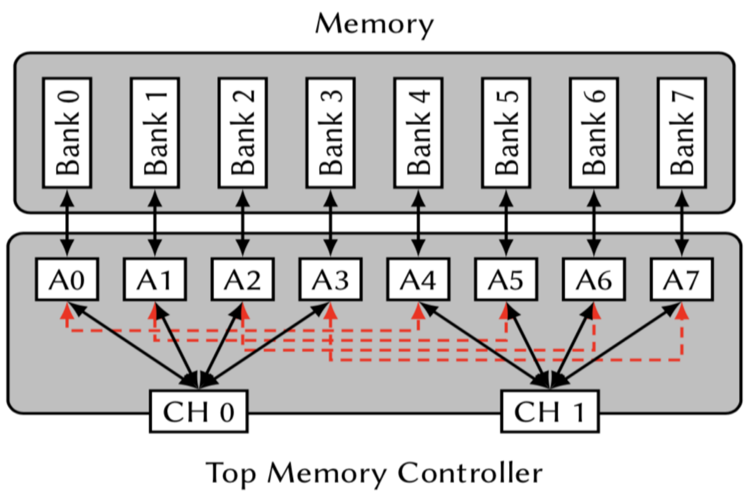
(b) Example of OpenMP application to be synthesized after HLS transformations.

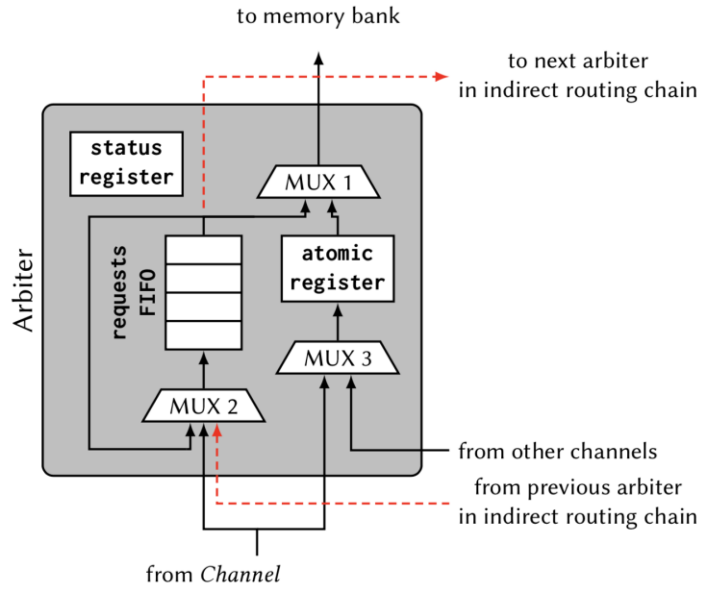
Multithreaded Architecture Template: worker and memory controller

- Architecture of a Single multithreaded worker

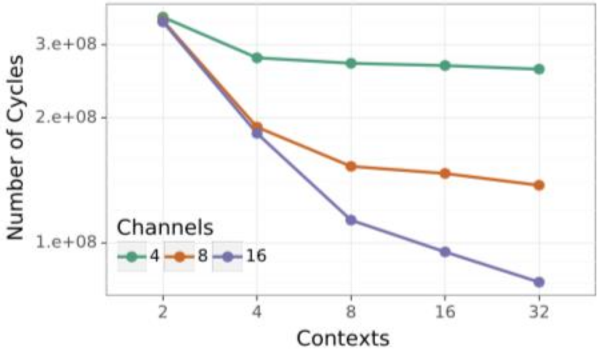


- Architecture of the Top Memory Controller

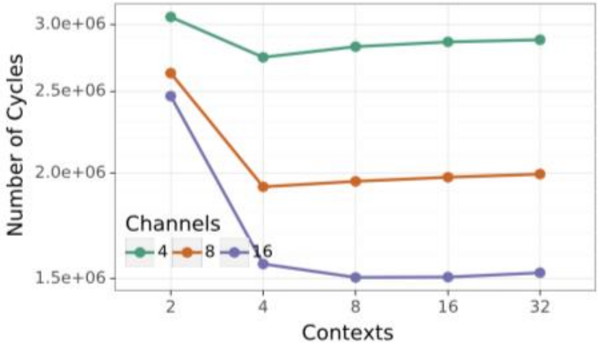




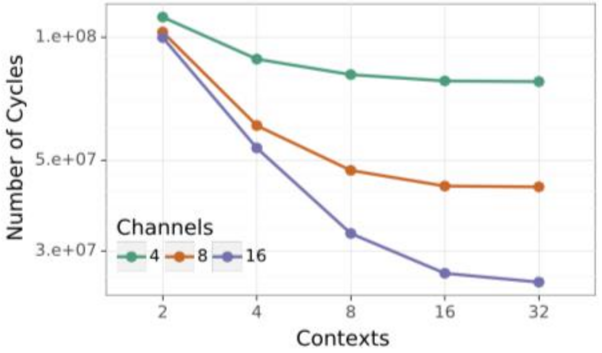
Design Space Exploration



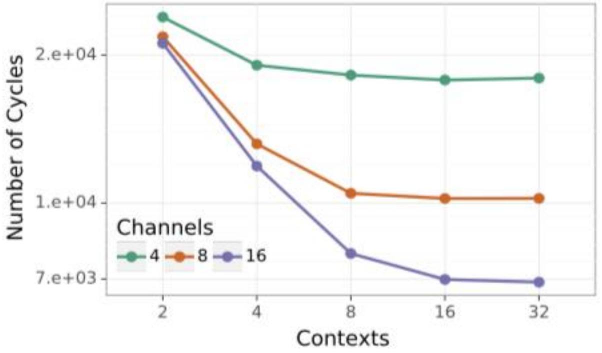
(a) *Q1*



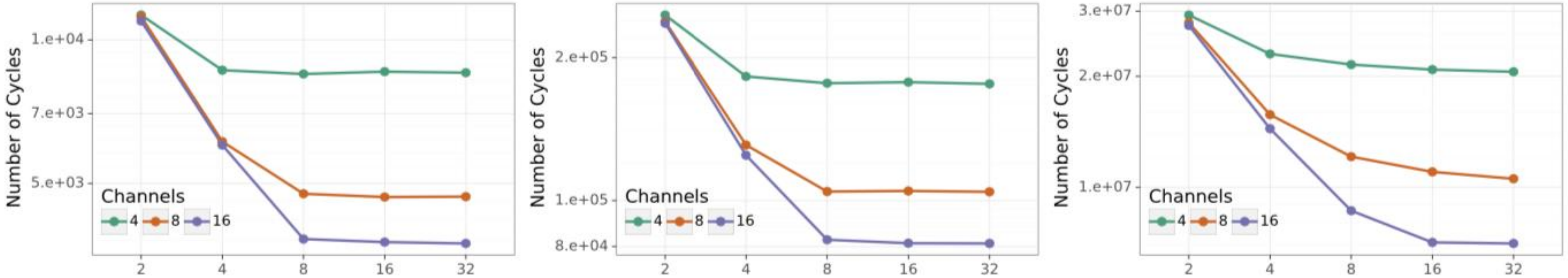
(b) *Q2*

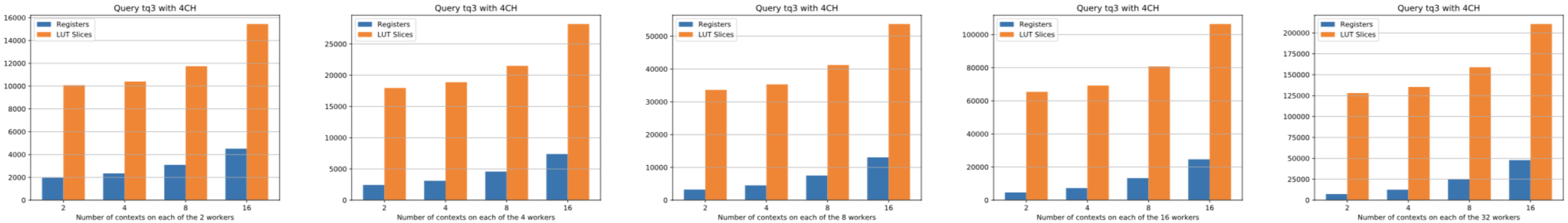


(c) *Q3*

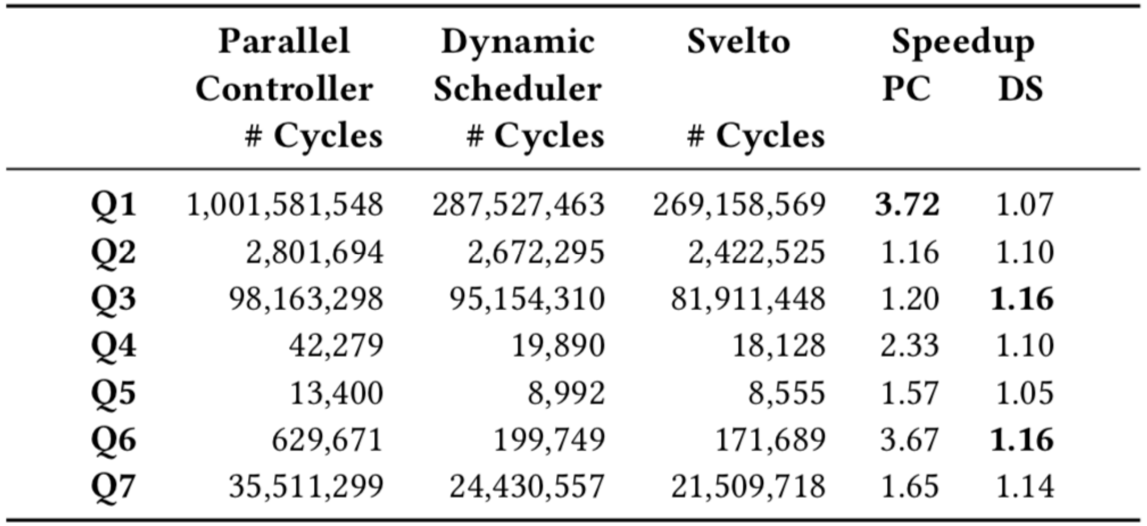


(d) *Q4*

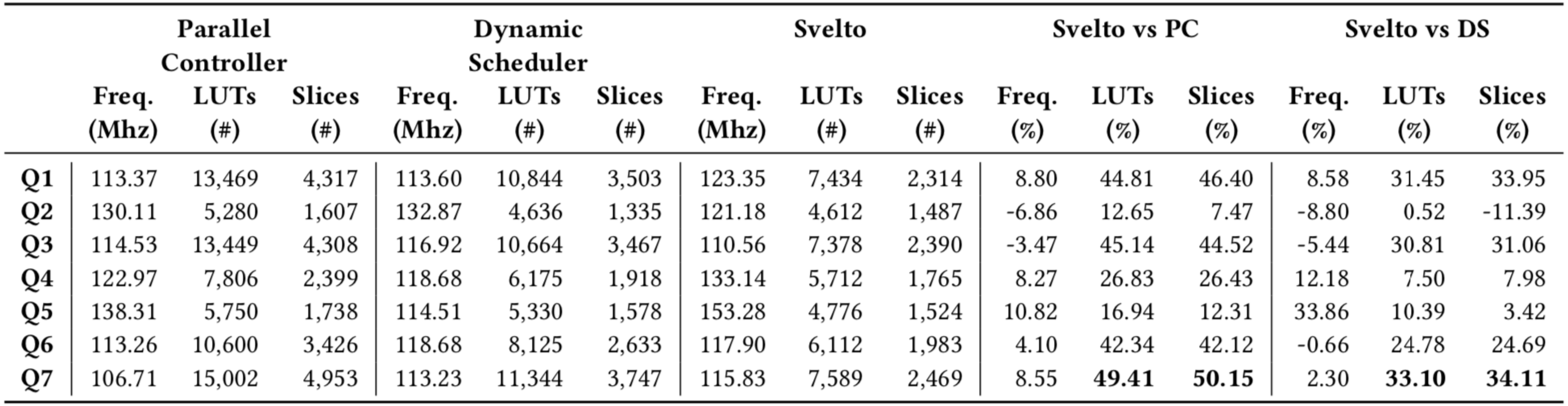




Results comparison



	Parallel Controller # Cycles	Dynamic Scheduler # Cycles	Svelto # Cycles	Speedup	
				PC	DS
Q1	1,001,581,548	287,527,463	269,158,569	**3.72**	1.07
Q2	2,801,694	2,672,295	2,422,525	1.16	1.10
Q3	98,163,298	95,154,310	81,911,448	1.20	**1.16**
Q4	42,279	19,890	18,128	2.33	1.10
Q5	13,400	8,992	8,555	1.57	1.05
Q6	629,671	199,749	171,689	3.67	**1.16**
Q7	35,511,299	24,430,557	21,509,718	1.65	1.14



	Parallel Controller			Dynamic Scheduler			Svelto			Svelto vs PC			Svelto vs DS		
	Freq. (Mhz)	LUTs (#)	Slices (#)	Freq. (Mhz)	LUTs (#)	Slices (#)	Freq. (Mhz)	LUTs (#)	Slices (#)	Freq. (%)	LUTs (%)	Slices (%)	Freq. (%)	LUTs (%)	Slices (%)
Q1	113.37	13,469	4,317	113.60	10,844	3,503	123.35	7,434	2,314	8.80	44.81	46.40	8.58	31.45	33.95
Q2	130.11	5,280	1,607	132.87	4,636	1,335	121.18	4,612	1,487	-6.86	12.65	7.47	-8.80	0.52	-11.39
Q3	114.53	13,449	4,308	116.92	10,664	3,467	110.56	7,378	2,390	-3.47	45.14	44.52	-5.44	30.81	31.06
Q4	122.97	7,806	2,399	118.68	6,175	1,918	133.14	5,712	1,765	8.27	26.83	26.43	12.18	7.50	7.98
Q5	138.31	5,750	1,738	114.51	5,330	1,578	153.28	4,776	1,524	10.82	16.94	12.31	33.86	10.39	3.42
Q6	113.26	10,600	3,426	118.68	8,125	2,633	117.90	6,112	1,983	4.10	42.34	42.12	-0.66	24.78	24.69
Q7	106.71	15,002	4,953	113.23	11,344	3,747	115.83	7,589	2,469	8.55	**49.41**	**50.15**	2.30	**33.10**	**34.11**

Conclusions

- Identified challenges due to irregular behaviors in Data Analytics applications
- Introduced GEMS, our graph database for homogeneous clusters, which supports RDF and SPARQL
- Highlighted possible ways to accelerate SPARQL queries with custom accelerators in our framework
- Identified current limitations of High-Level Synthesis (HLS) for Data Analytics applications
- Presented architectural templates and methodologies for the HLS of Data Analytics applications
- Presented results of the synthesis of SPARQL queries with our methodology

- Questions?
  - antonino.tumeo@pnnl.gov
  - vitoGiovanni.castellana@pnnl.gov
  - marco.minutoli@pnnl.gov

  - serena.curzel@polimi.it
  - michele.fiorito@polimi.it
  - fabrizio.ferrandi@polimi.it

Thank you