



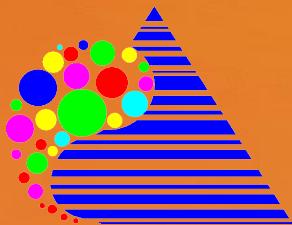
Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965

Enabling the High Level Synthesis of Data Analytics Accelerators

*MARCO MINUTOLI, VITO GIOVANNI CASTELLANA, ANTONINO TUMEO
PACIFIC NORTHWEST NATIONAL LABORATORY
RICHLAND, WA, USA*

*MARCO LATTUADA, FABRIZIO FERRANDI
POLITECNICO DI MILANO
MILANO, ITALY*





New generation of *irregular* HPC applications



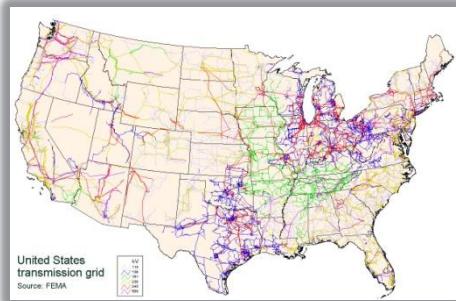
Big Science



Bioinformatics



Community Detection



Complex Networks



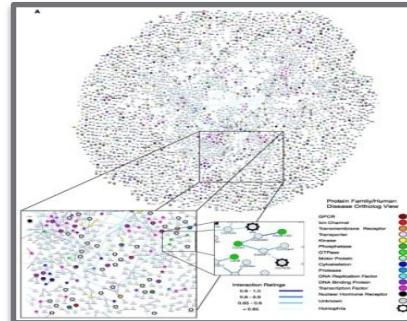
Graph Databases



Knowledge Discovery



Language Understanding



Pattern Recognition



Massive Social Networks

► Statistics



1 billion active users, even more objects

surpassed 1 billion active users

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



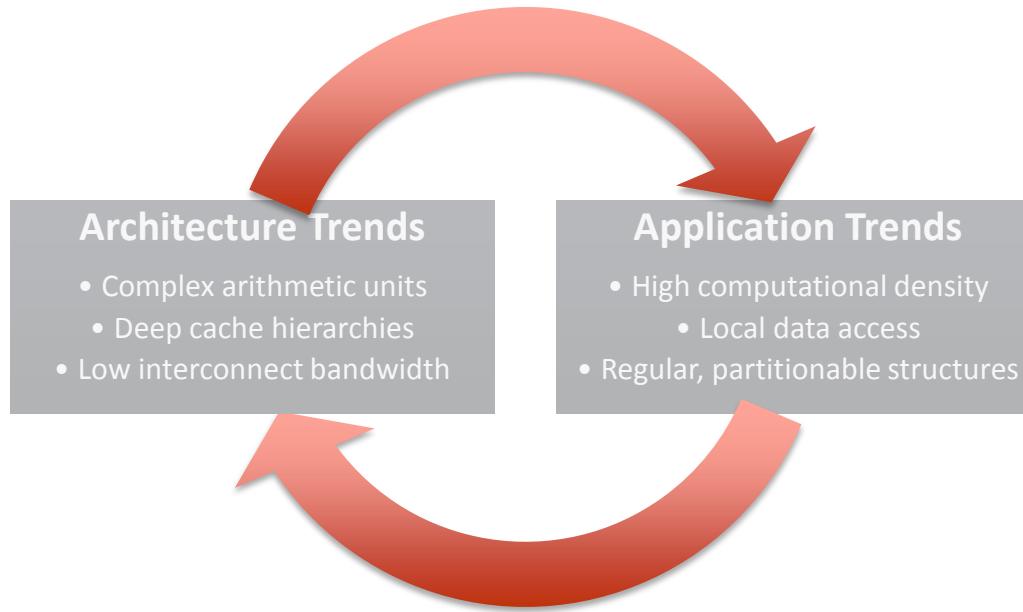
Additional Characteristics

- ▶ 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.



- 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



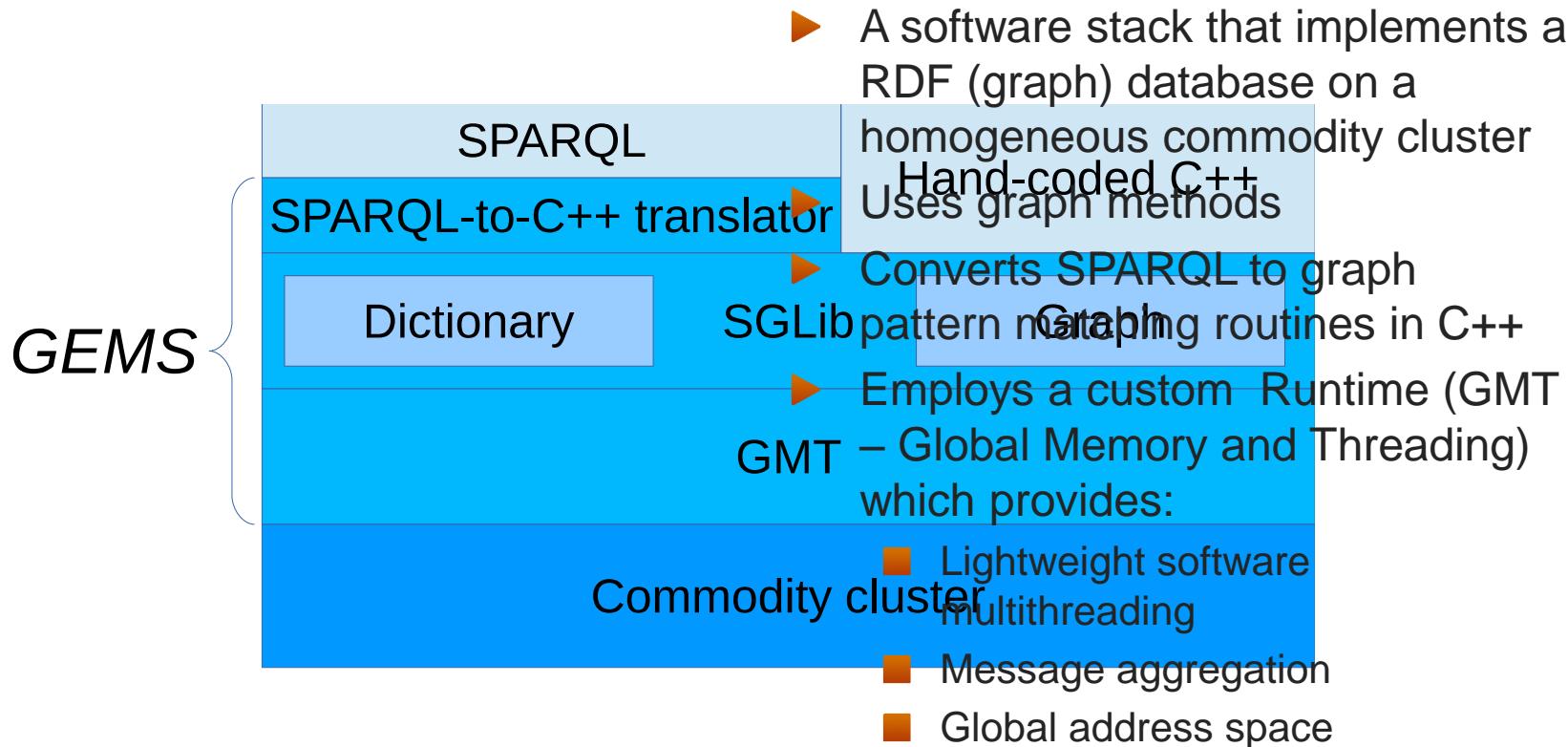
Exemplar Irregular & Data Analytics

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)



[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)]

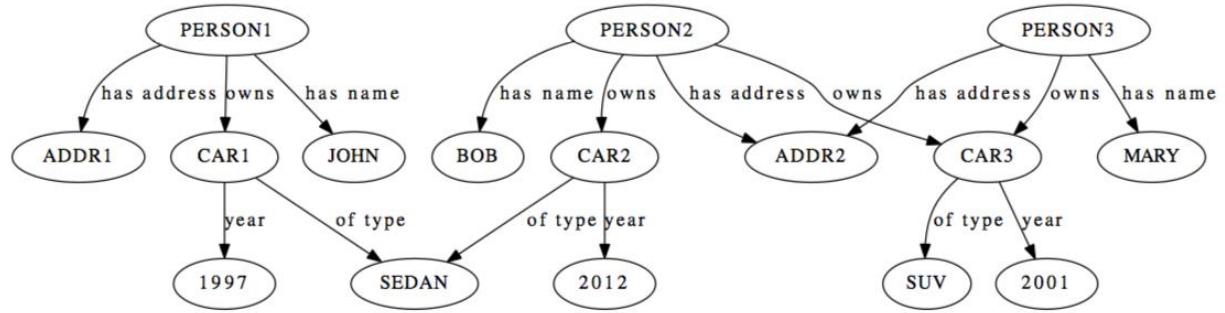


Query example

- 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 .
    
```

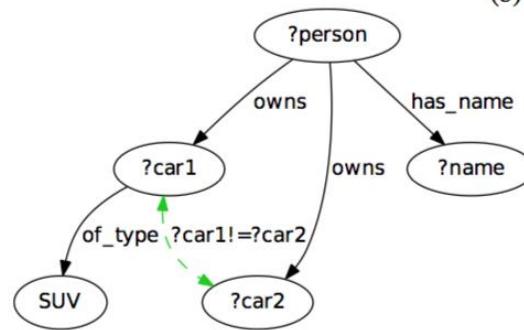


(a) Dataset in simplified N-Triples format

```

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

```

1 has_name = get_label("has_name")
2 of_type = get_label("of_type")
3 owns = get_label("owns")
4 suv = get_label("SUV")
5 forall e1 in edges(*, of_type, suv)
6   ?car1 = source_node(e1)
7   forall e2 in edges(*, owns, ?car1)
8     ?person = source_node(e2)
9     forall e3 in edges(?person, owns, *)
10    ?car2 = target_node(e3)
11    if (?car1 != ?car2)
12      forall e4 in edges(?person, has_name, *)
13        ?name = target_node(e4)
14        tuples.add(<?name>)
15  distinct(tuples)
    
```

(e) Pseudocode



Source Code Example

```

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

void kernel(size_t i_var3, Edge * var_2_1_inEdges, Graph
1   * graph, NodeId var_2, Label p_var_3, LabelId
2   p_var_4, LabelId p_var_5, LabelId p_var_7, LabelId
3   p_var_8, LabelId p_var_9) {
4   LabelId var_3; //el. with label "ub:subOrganizationOf"
5   var_3 = var_2_1_inEdges[i_var3].property;
6   NodeId var_1; //el. with label "?Y"
7   var_1 = var_2_1_inEdges[i_var3].node;
8   if(var_3 == p_var_3) {
9     size_t in_degree_var_1 = getInDegree(graph, var_1);
10    Edge * var_1_3_inEdges = getInEdges(graph, var_1);
11    for(size_t i_var_7 = 0; i_var_7 < in_degree_var_1;
12        i_var_7++) {
13      // Same as Fig. 5a lines {13--38}
14      ...
15    }
16  }
17 }

void search(Graph * graph, NodeId var_2, Label p_var_3,
18   LabelId p_var_4, LabelId p_var_5, LabelId p_var_7,
19   LabelId p_var_8, LabelId p_var_9) {
20   size_t in_degree_var_2 = getInDegree(graph, var_2);
21   Edge * var_2_1_inEdges = getInEdges(graph, var_2);
22   size_t i_var_3;
23
24   for(i_var_3=0; i_var_3 < in_degree_var_2%4; i_var_3++) {
25     kernel(i_var3, var_2_1_inEdges, graph, p_var_3,
26           p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
27   }
28
29   for(; i_var_3 < in_degree_var_2%4; i_var_3+=4) {
30     kernel(i_var3, var_2_1_inEdges, graph, p_var_3,
31           p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
32     kernel(i_var3+1, var_2_1_inEdges, graph, p_var_3,
33           p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
34     kernel(i_var3+2, var_2_1_inEdges, graph, p_var_3,
35           p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
36     kernel(i_var3+3, var_2_1_inEdges, graph, p_var_3,
37           p_var_4, p_var_5, p_var_7, p_var_8, p_var_9);
38   }
39 }
```

(a)

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

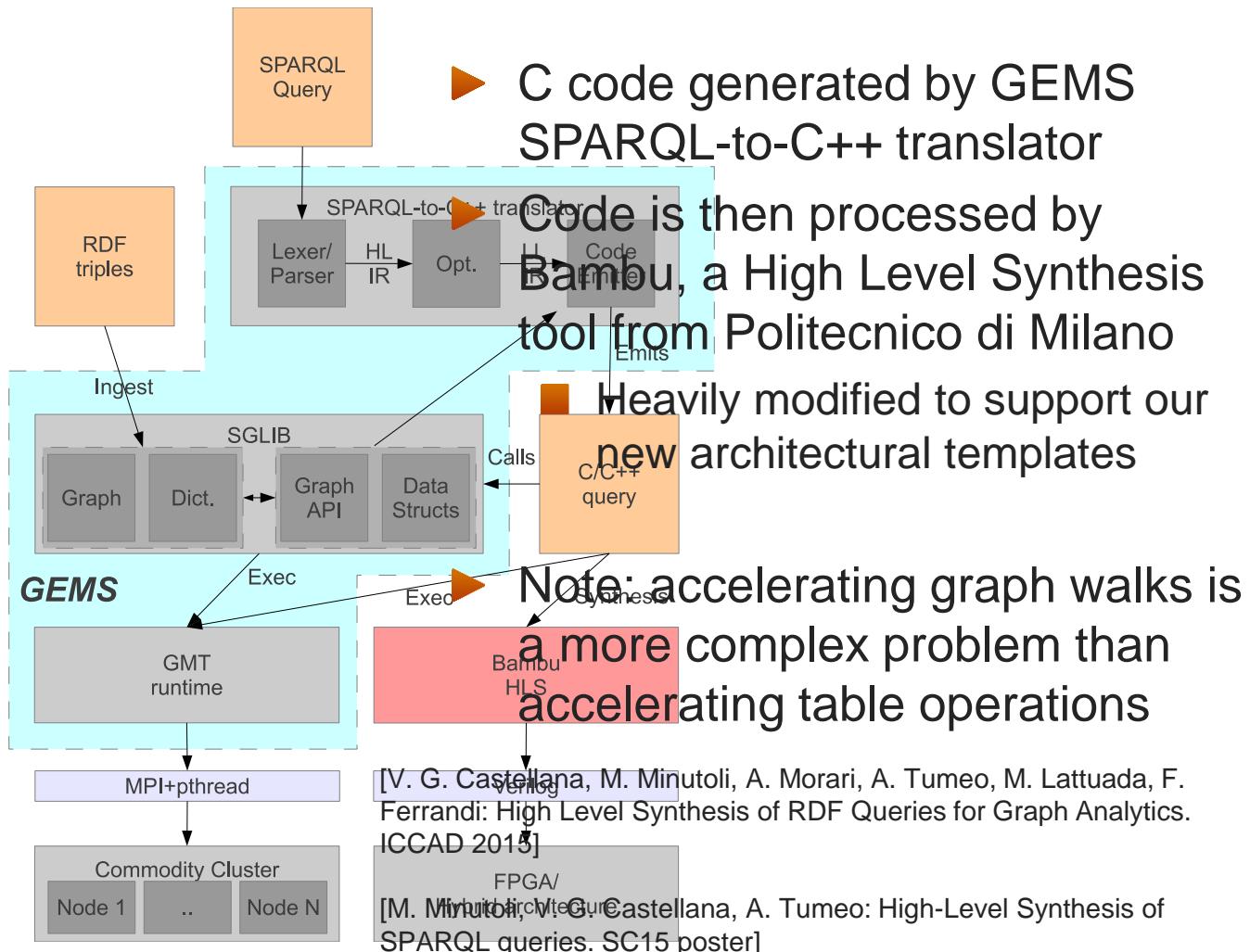


Can we use FPGAs to accelerate Data Analytics and Irregular Applications?

- ▶ 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





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

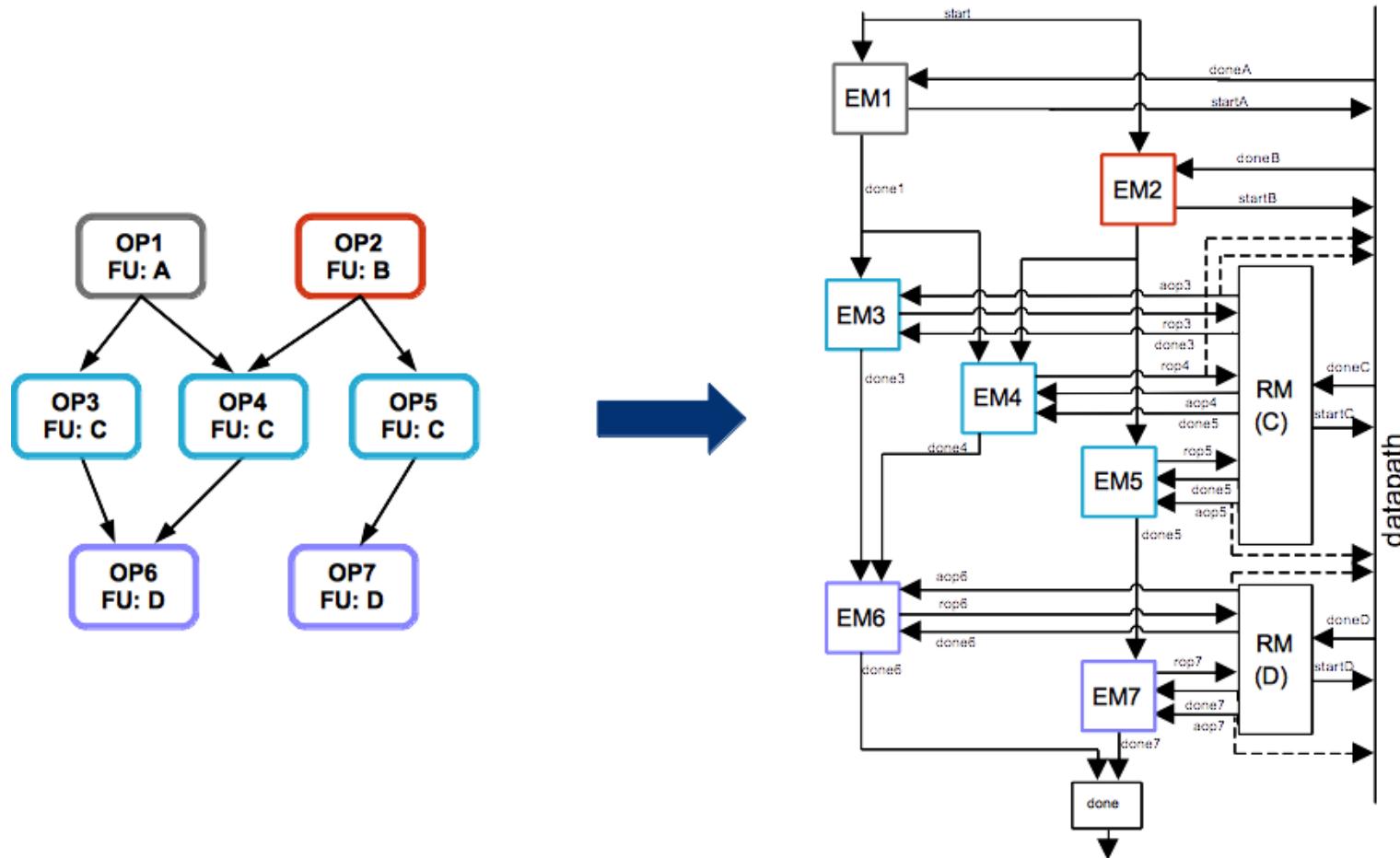


Solving Challenge 1: Parallel Distributed Controller

- ▶ 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

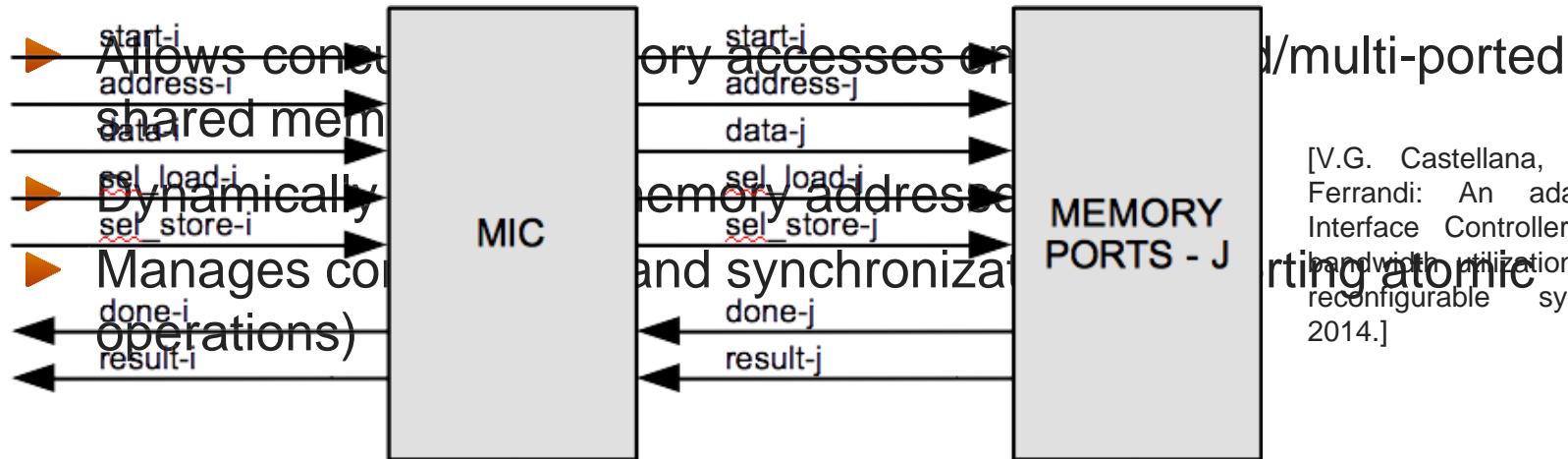


Example of Parallel Distributed Controller





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.]



HMI details

► Manages **concurrency**

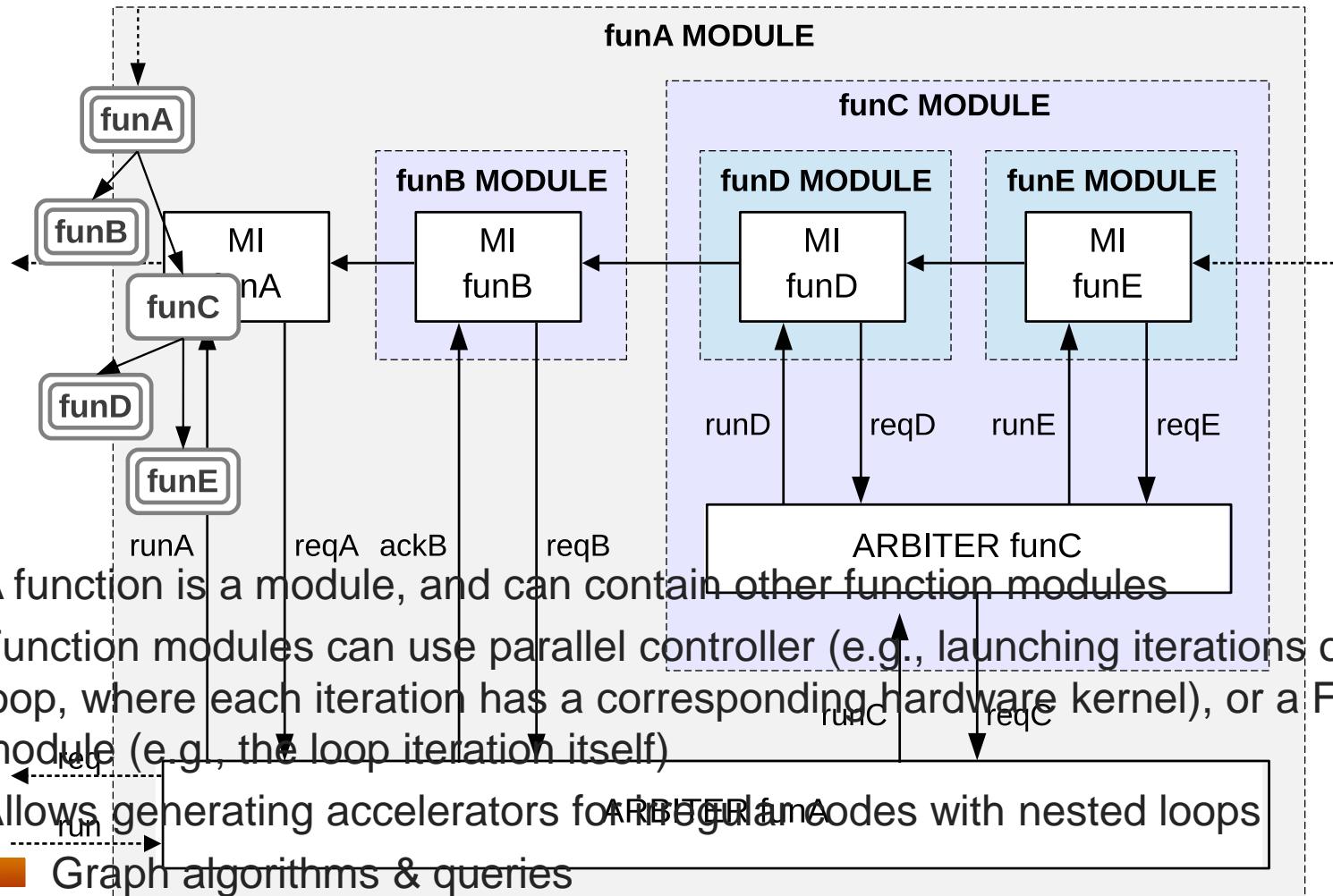
- Provides memory scrambling (solves hotspot)
- Collects memory requests, and if their target addresses collide, it forwards them one at a time
- Each memory port is managed by a dedicated arbiter
- No delay penalties

► 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



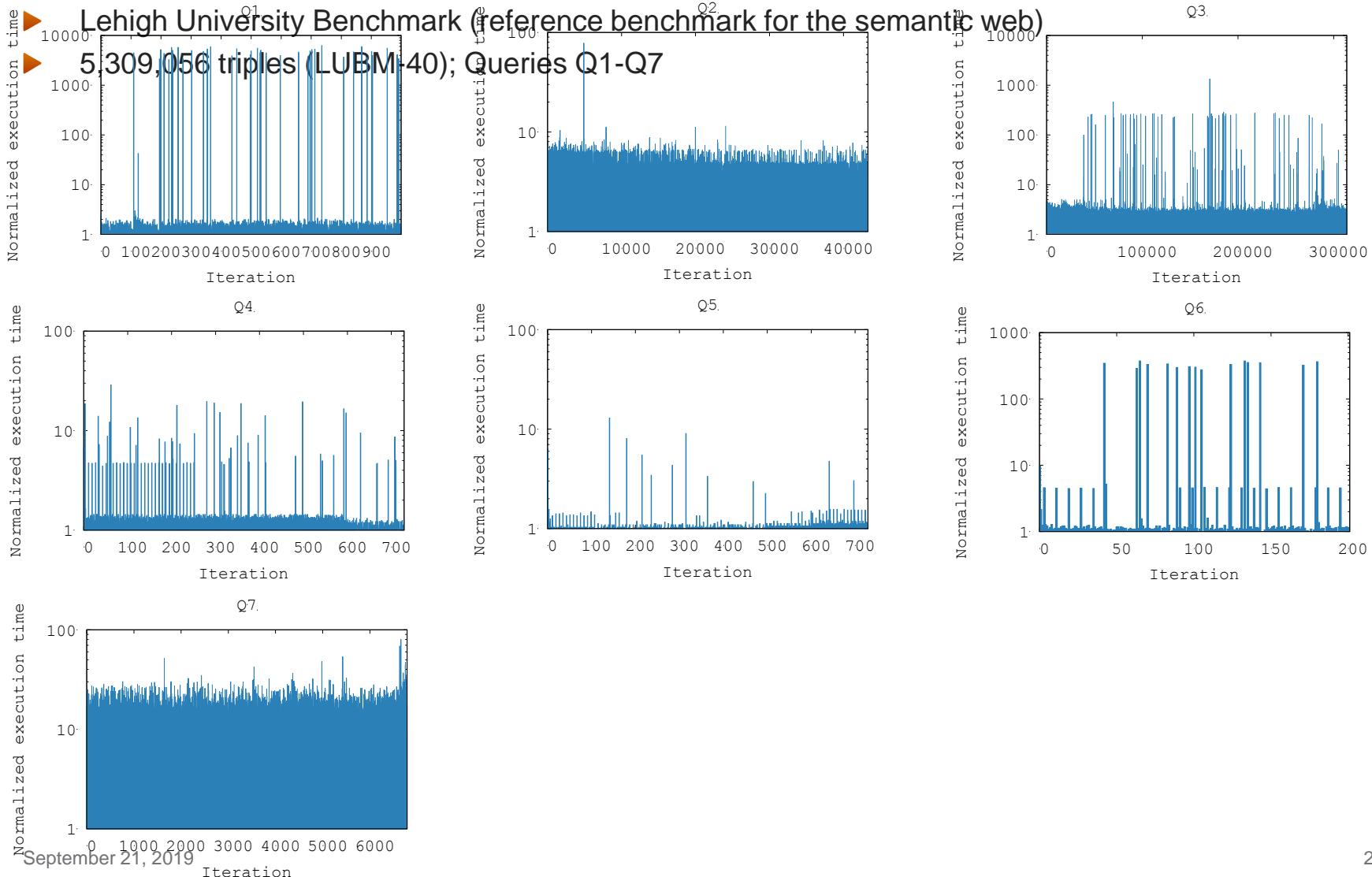
Hierarchical Approach to HLS



- ▶ 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 ARBITER modes with nested loops
 - Graph algorithms & queries

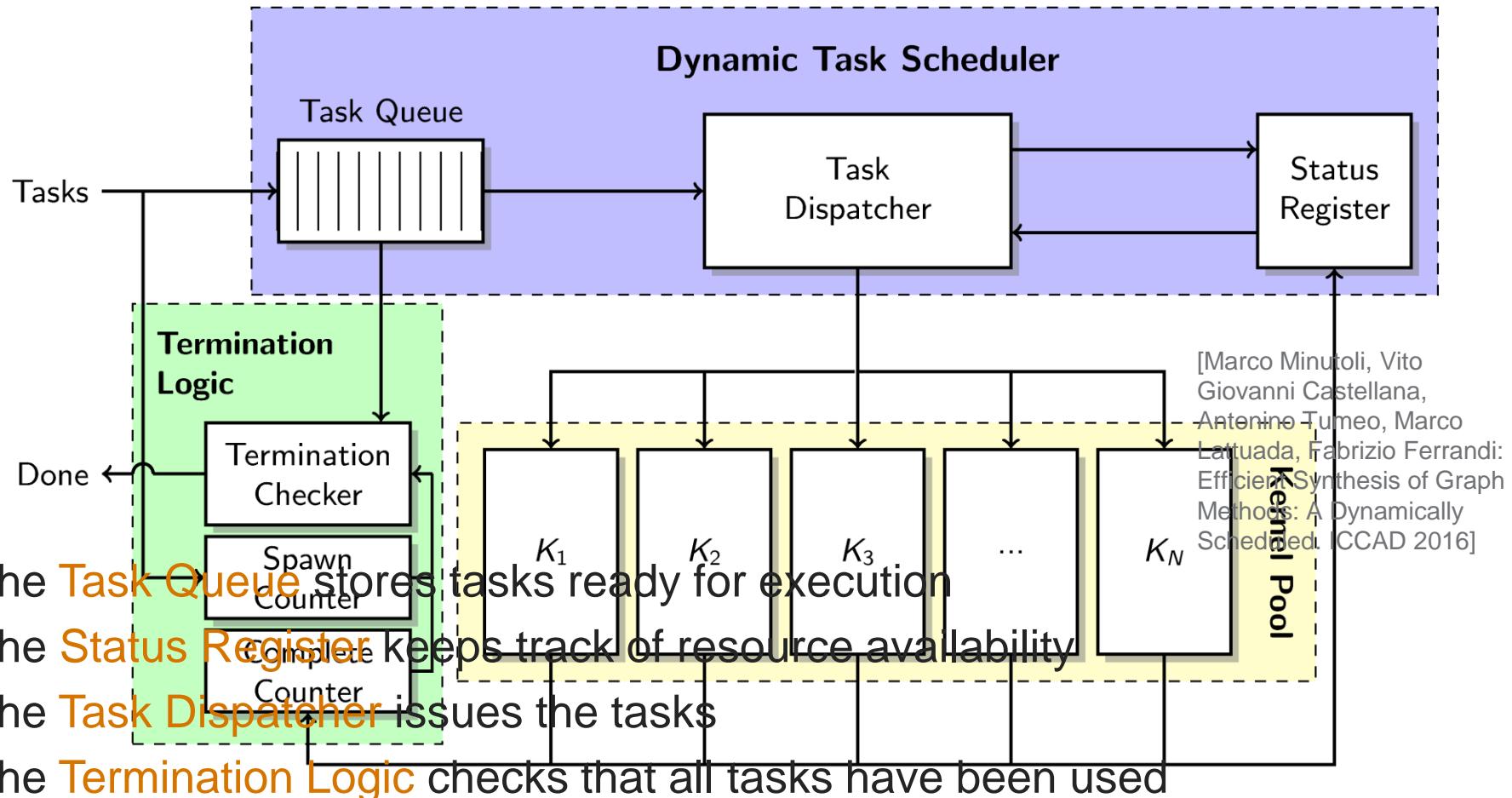


Load Unbalancing in Queries





Dynamic Task Scheduler





Experimental Evaluation

LUBM-1 (100k triples)

- ▶ 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)
- | Single Acc. | Parallel Controller | Dynamic Scheduler | Speedup | Parallel Controller |
|-------------|---------------------|-------------------|-------------|---------------------|
| # Cycles | # Cycles | # Cycles | Single Acc. | Parallel Controller |
| Q1 | 5,339,286 | 5,176,116 | 5,129,902 | 1.04 |
| Q2 | 141,022 | 54,281 | 50,997 | 2.77 |
| Q3 | 5,824,354 | 1,862,683 | 1,805,731 | 3.23 |
| Q4 | 63,825 | 42,851 | 19,928 | 3.20 |
| Q5 | 33,322 | 13,442 | 9,016 | 3.70 |

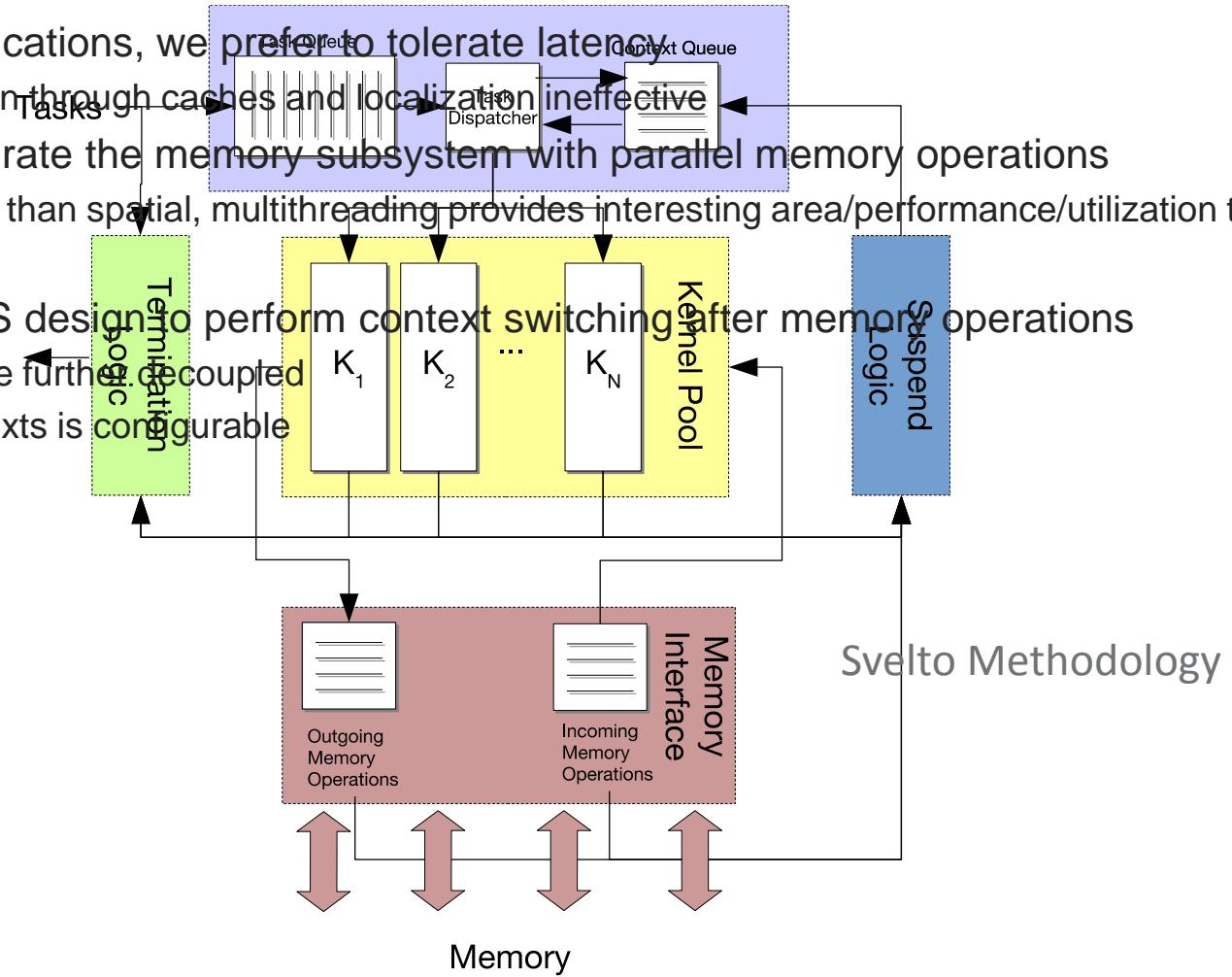
Single Acc.	Parallel Controller	Dynamic Scheduler	Speedup	
# Cycles	# Cycles	# Cycles	Single Acc.	Parallel Controller
Q1	5,339,286	5,176,116	5,129,902	1.04
Q2	141,022	54,281	50,997	2.77
Q3	5,824,354	1,862,683	1,805,731	3.23
Q4	63,825	42,851	19,928	3.20
Q5	33,322	13,442	9,016	3.70

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



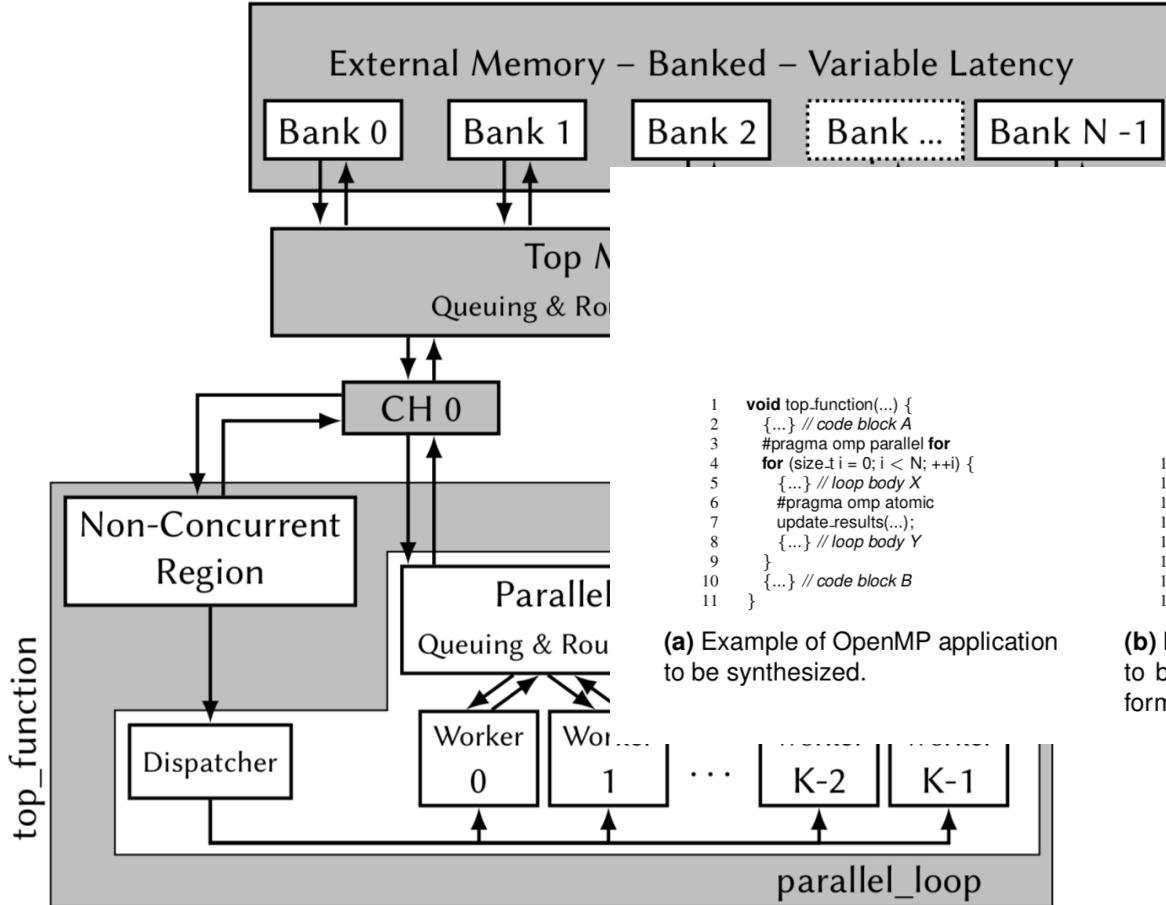
Latest Extension: Temporal Multithreading

- ▶ 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





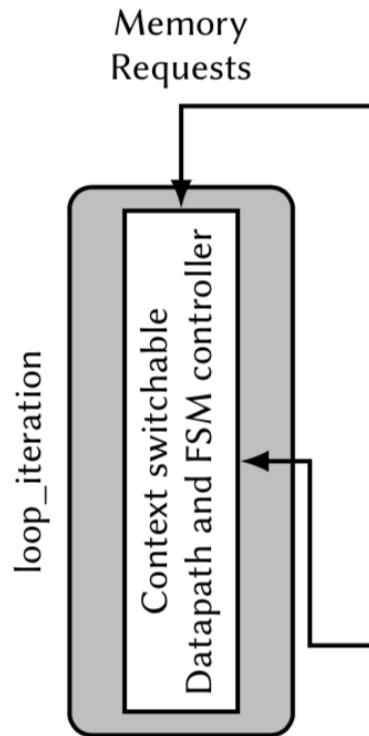
Architectural Template Mapping



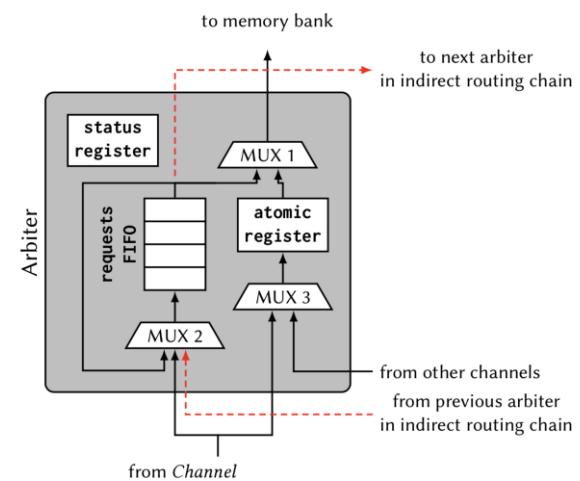
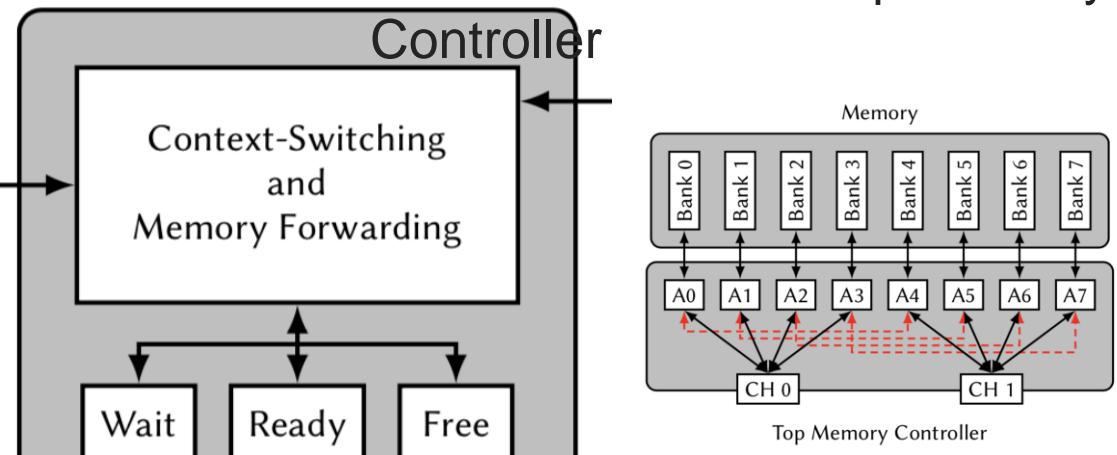


Multithreaded Architecture Template: worker and memory controller

- Architecture of a Single multithreaded worker



- Architecture of the Top Memory Controller





First example – Sequential Accelerator

- ▶ Create a sequential accelerator implementing tq4 specifying input/output testbench with xml file
- ▶ Hints:
 - Start from the bambu.sh script
 - Set the top function equal to search
 - Specify that the memories are outside the accelerator
 - Set the memory delay to an high value
 - Specify as testbench argument the xml that is in the directory



Solution

```
bambu --compiler=I386_GCC49 --std=c99 --experimental-
set=BAMBU -O3 -v3 -fno-delete-null-pointer-checks --
channels-type=MEM_ACC_11 --memory-allocation-
policy=NO_BRAM --device-name=xc7vx690t-3ffg1930-VVD --
clock-period=10 -DMAX_VERTEX_NUMBER=26455 -
DMAX_EDGE_NUMBER=100573 -DN_THREADS=1
${root_dir}/common/atomInIncrement.c
${root_dir}/common/data.c -I${root_dir}/common/
${root_dir}/trinityq4/lubm_trinityq4.c \
--top-fname=search \
--mem-delay-read=20 --mem-delay-write=20 \
--memory-allocation-policy=NO_BRAM \
--simulate --generate-tb=${root_dir}/test-1.xml
```



Second example – Parallel Accelerator

- ▶ Create a parallel accelerator implementing tq4 specifying input/output testbench with xml file
- ▶ Configuration to be considered
 - 2 Copies of the kernel
 - 4 External memory banks
 - 2 Internal channels
 - No context switch (i.e., context-switch==1)



Solution

```
bambu --compiler=I386_GCC49 --std=c99 --experimental-set=BAMBU -O3 -v3 -fno-delete-null-pointer-checks --channels-type=MEM_ACC_11 --ddddd=NO_BRAM --device-name=xc7vx690t-3ffg1930-VVD --clock-period=10 -DMAX_VERTEX_NUMBER=26455 -DMAX_EDGE_NUMBER=100573 -DN_THREADS=2 ${root_dir}/common/atomInIncrement.c ${root_dir}/common/data.c -I${root_dir}/common/ ${root_dir}/trinityq4/lubm_trinityq4.c --top-fname=search --mem-delay-read=20 --mem-delay-write=20 --memory-allocation-policy=NO_BRAM --simulate --generate-tb=${root_dir}/test-1.xml \
--num-threads=2 \
--memory-banks-number=4 \
--channels-number=2 \
--context_switch=1
```



Third example – Context Switch

- ▶ Create a parallel accelerator implementing tq4 specifying input/output testbench with xml file
- ▶ Configuration to be considered
 - 2 Copies of the kernel
 - 4 External memory banks
 - 2 Internal channels
 - 4 Logic Threads x kernel



Solution

```
bambu --compiler=I386_GCC49 --std=c99 --experimental-set=BAMBU -O3 -v3 -fno-delete-null-pointer-checks --channels-type=MEM_ACC_11 --ddddd=NO_BRAM --device-name=xc7vx690t-3ffg1930-VVD --clock-period=10 -DMAX_VERTEX_NUMBER=26455 -DMAX_EDGE_NUMBER=100573 -DN_THREADS=2 ${root_dir}/common/atomInIncrement.c ${root_dir}/common/data.c -I${root_dir}/common/ ${root_dir}/trinityq4/lubm_trinityq4.c --top-fname=search --mem-delay-read=20 --mem-delay-write=20 --memory-allocation-policy=NO_BRAM --simulate --generate-tb=${root_dir}/test-1.xml \
--num-threads=2 \
--memory-banks-number=4 \
--channels-number=2 \
--context_switch=4
```



Fourth example – Context Switch DSE

- ▶ Create parallel accelerators implementing tq4 specifying input/output testbench with xml file using context switch
- ▶ Evaluate different configurations
 - Double the number of contexts
 - Double the number of memory banks
 - Etc.

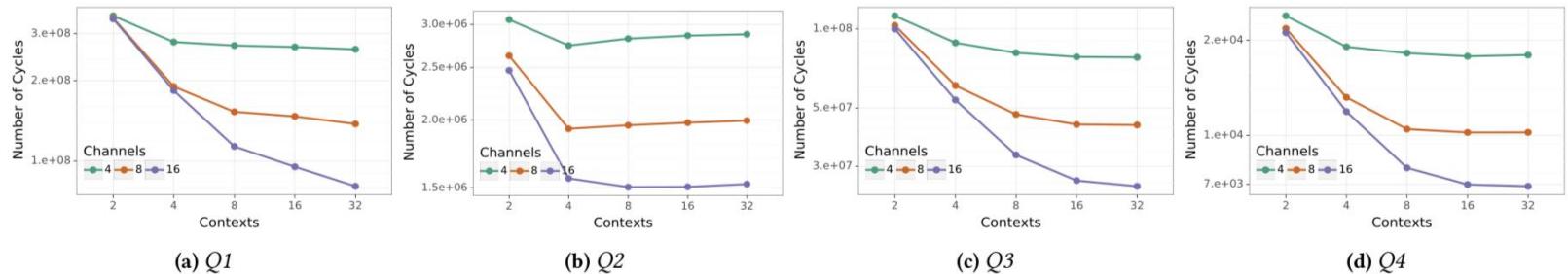


Solution

- ▶ --memory-banks-number=4 --channels-number=2
--context_switch=8
- ▶ --memory-banks-number=8 --channels-number=2
--context_switch=8
- ▶ --memory-banks-number=16 --channels-number=2
--context_switch=8



Design Space Exploration

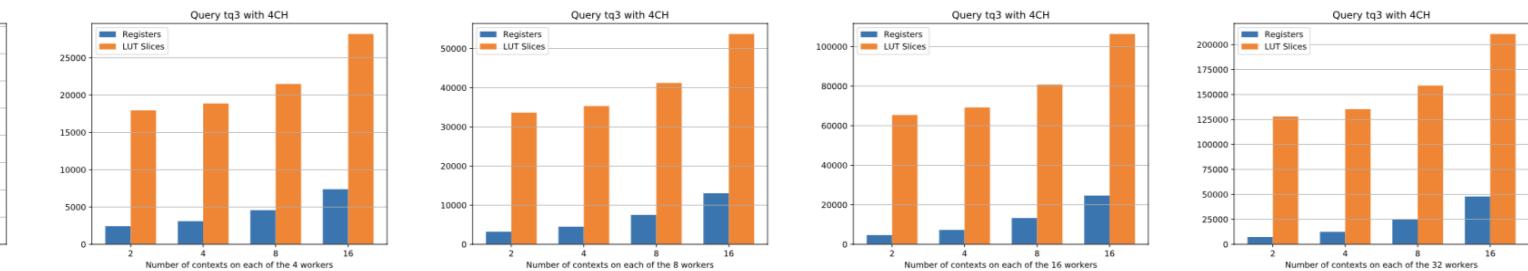
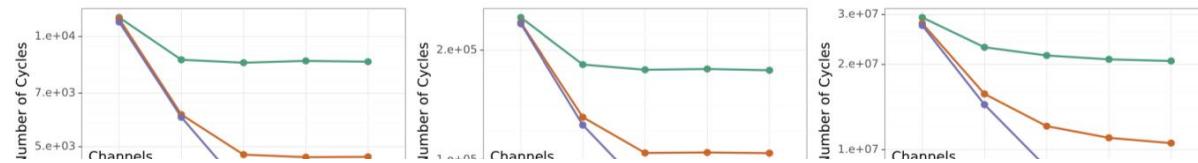


(a) Q1

(b) Q2

(c) Q3

(d) Q4





Results comparison

Parallel Controller	Freq. (Mhz)	LUTs (#)	Parallel	Dynamic	Svelto	Speedup		Svelto vs DS							
			Controller	Scheduler	# Cycles	PC	DS								
			# Cycles	# Cycles	# Cycles										
Parallel Controller	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									
Q1	113.37	13,469													
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	8.58	31.45	33.95
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	-8.80	0.52	-11.39
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	30.81	31.06	
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	7.50	12.18	7.98
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	33.86	10.39	3.42
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



Thank you for your attention!

► Questions?

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

- marco.lattuada@polimi.it
- fabrizio.ferrandi@polimi.it