Towards Highly Parallel Event Processing through Reconfigurable Hardware

Mohammad Sadoghi Harsh Singh Hans-Arno Jacobsen

University of Toronto

June 13, 2011

1 Real-time Event Processing Scenario

Real-time Event Processing Scenario

Algorithm Trading

Algorithmic trading is a computer-based approach to execute buy and sell orders on financial instruments such as securities (e.g., stocks and bonds.)

Real-time Event Processing Scenario

Algorithm Trading

Algorithmic trading is a computer-based approach to execute buy and sell orders on financial instruments such as securities (e.g., stocks and bonds.)

Algorithmic Trading Challenges

1. Sustain a high event rate because algorithmic trading dominates financial markets and accounts for over 70% of all trading
2. Minimize matching time because every 1-millisecond generates a staggering amount of $100 million annually

Real-time Event Processing Challenges

Event Processing Requirements

An event processing platform must efficiently find all patterns or specifications (subscriptions) that match incoming events at a rate up to a million events per second.

Real-time Event Processing Challenges

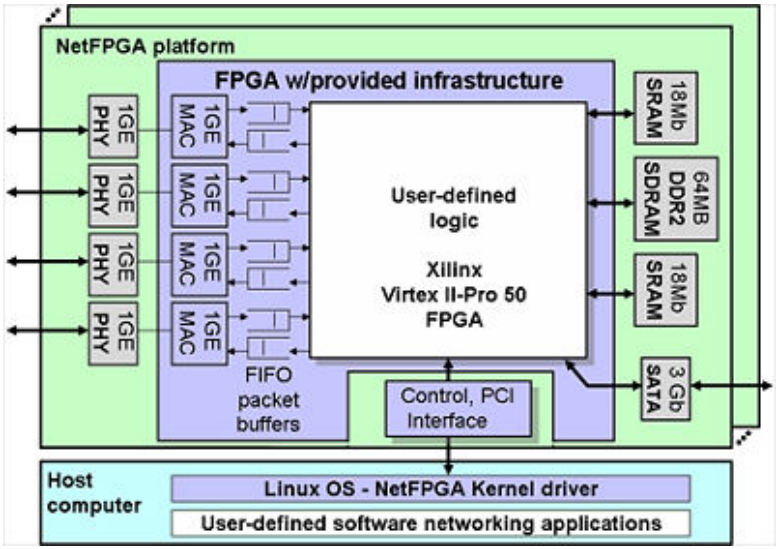
Event Processing Requirements

An event processing platform must efficiently find all patterns or specifications (subscriptions) that match incoming events at a rate up to a million events per second.

Our Solution

We propose a novel FPGA-based event processing platform to significantly accelerate event processing computations, namely, event parsing and event matching against patterns or specifications.

NetFPGA Card



Verilog Snippet

```
case (CurrentState)
        IDLE: begin
                if (Go)
                        NextState = SELECT_CLUSTER_ID;
                else
                        NextState = IDLE;
        end
        SELECT_CLUSTER_ID: begin                        // Select a valid cluster index
                if (curCluster > LAST_CLUSTER)          // Finished reading last cluster
                                                        // When all clusters are invalid
                        NextState = WAIT;
                else
                        NextState = START_ADDRESS;
        end
        START_ADDRESS: begin                            // Select cluster address
                if (curCluster > LAST_CLUSTER)          // Finished reading last cluster
                                                        // When all clusters are invalid
                        NextState = WAIT;
                else if (can_take_more_requests)
                        NextState = MEM_BURST_WAIT;
                else
                        NextState = START_ADDRESS;      // If 'can_take_more_requests' is low, wait
        end
        NEXT_ADDRESS: begin                             // Check data from cluster terminator
                if (clusterEndFound)
                        NextState = SELECT_CLUSTER_ID;
                else if (can_take_more_requests)
                        NextState = MEM_BURST_WAIT;     // Increment curAddr by 16 bytes
                else
                        NextState = NEXT_ADDRESS;       // If 'can_take_more_requests' is low, wait
        end
        default
                NextState = IDLE;
endcase
```

Verilog Snippet

```
case (CurrentState)
        IDLE: begin
                if (Go)
                        NextState = SELECT_CLUSTER_ID;
                else
                        NextState = IDLE;
        end
        SELECT_CLUSTER_ID: begin                        // Select a valid cluster index
                if (curCluster > LAST_CLUSTER)          // Finished reading last cluster
                                                        // When all clusters are invalid
                        NextState = WAIT;
                else
                        NextState = START_ADDRESS;
        end
```

```
SELECT_CLUSTER_ID: begin            //Select a valid cluster index
     if (curCluster > LAST_CLUSTER) //Finished reading last cluster
                                    //When all clusters are invalid
          NextState = WAIT;
     else
          NextState = START_ADDRESS;
     end
```

```
                        NextState = NEXT_ADDRESS;       // If 'can_take_more_requests' is low, wait
        end
        default
                NextState = IDLE;
endcase
```

FPGAs Challenges

FPGA caveats

1. The latest FPGA (e.g., 800MHz Xilinx Virtex 6) operates at significantly lower speed compared to CPUs (e.g., 3.46GHz Intel i7)

FPGAs Challenges

FPGA caveats

1. The latest FPGA (e.g., 800MHz Xilinx Virtex 6) operates at significantly lower speed compared to CPUs (e.g., 3.46GHz Intel i7)
2. The accelerated application functionality has to be amenable to parallel processing

FPGAs Challenges

FPGA caveats

1. The latest FPGA (e.g., 800MHz Xilinx Virtex 6) operates at significantly lower speed compared to CPUs (e.g., 3.46GHz Intel i7)
2. The accelerated application functionality has to be amenable to parallel processing
3. The memory bandwidth must keep up with chip processing speeds to realize a speedup by keeping the custom-built processing pipeline busy

FPGAs Challenges

FPGA caveats

1. The latest FPGA (e.g., 800MHz Xilinx Virtex 6) operates at significantly lower speed compared to CPUs (e.g., 3.46GHz Intel i7)
2. The accelerated application functionality has to be amenable to parallel processing
3. The memory bandwidth must keep up with chip processing speeds to realize a speedup by keeping the custom-built processing pipeline busy

Open Problems

The true success of FPGAs is rooted in three distinctive features: hardware reconfigurability, hardware parallelism, and onboard packet processing.

Why FPGAs

FPGA distinctive features

1. *Hardware reconfigurability:* re-configuring the application on-demand into a highly parallel custom processors

Why FPGAs

FPGA distinctive features

1. *Hardware reconfigurability:* re-configuring the application on-demand into a highly parallel custom processors
2. *Hardware parallelism:* eliminating inter-processor signalling and message passing overhead associated with the concurrency management at the program and the OS level

Why FPGAs

FPGA distinctive features

1. *Hardware reconfigurability:* re-configuring the application on-demand into a highly parallel custom processors
2. *Hardware parallelism:* eliminating inter-processor signalling and message passing overhead associated with the concurrency management at the program and the OS level
3. *Onboard packet processing:* using multiple high bandwidth (giga-bit) I/O pins to eliminate the OS layer latency overhead in moving data between input and output ports

Why FPGAs

FPGA distinctive features

1. *Hardware reconfigurability:* re-configuring the application on-demand into a highly parallel custom processors
2. *Hardware parallelism:* eliminating inter-processor signalling and message passing overhead associated with the concurrency management at the program and the OS level
3. *Onboard packet processing:* using multiple high bandwidth (giga-bit) I/O pins to eliminate the OS layer latency overhead in moving data between input and output ports
4. *Cost-effective and Energy-efficient*

Language and Data Model

- *Event* is modeled as a value assignment to attributes.

Language and Data Model

- *Event* is modeled as a value assignment to attributes.
- *Subscription* is modeled as a Boolean expression.

Language and Data Model

- *Event* is modeled as a value assignment to attributes.
- *Subscription* is modeled as a Boolean expression.
- A predicate P is a triple consisting of an attribute uniquely representing a dimension in n-dimensional space, an operator, and/or a set of values, denoted by $P^{(\texttt{attr},\texttt{opt},\texttt{val})}(x)$ triplet or $P^{(\texttt{attr},\texttt{opt},\texttt{attr})}(x)$ triplet.

Language and Data Model

- *Event* is modeled as a value assignment to attributes.
- *Subscription* is modeled as a Boolean expression.
- A predicate P is a triple consisting of an attribute uniquely representing a dimension in n-dimensional space, an operator, and/or a set of values, denoted by $P^{(\texttt{attr},\texttt{opt},\texttt{val})}(x)$ triplet or $P^{(\texttt{attr},\texttt{opt},\texttt{attr})}(x)$ triplet.
- A predicate $P(x)$ either accepts or rejects an input x such that $P(x) : x \longrightarrow \{\texttt{True}, \texttt{False}\}$, where $x \in \texttt{Dom}(P)$.

Language and Data Model

- *Event* is modeled as a value assignment to attributes.
- *Subscription* is modeled as a Boolean expression.
- A predicate P is a triple consisting of an attribute uniquely representing a dimension in n-dimensional space, an operator, and/or a set of values, denoted by $P^{(\texttt{attr},\texttt{opt},\texttt{val})}(x)$ triplet or $P^{(\texttt{attr},\texttt{opt},\texttt{attr})}(x)$ triplet.
- A predicate $P(x)$ either accepts or rejects an input x such that $P(x) : x \longrightarrow \{\texttt{True}, \texttt{False}\}$, where $x \in \texttt{Dom}(P)$.
- Formally, a Boolean expression e is defined over an n-dimensional space as follows

Language and Data Model

- *Event* is modeled as a value assignment to attributes.
- *Subscription* is modeled as a Boolean expression.
- A predicate P is a triple consisting of an attribute uniquely representing a dimension in n-dimensional space, an operator, and/or a set of values, denoted by $P^{(\texttt{attr},\texttt{opt},\texttt{val})}(x)$ triplet or $P^{(\texttt{attr},\texttt{opt},\texttt{attr})}(x)$ triplet.
- A predicate $P(x)$ either accepts or rejects an input x such that $P(x) : x \longrightarrow \{\texttt{True}, \texttt{False}\}$, where $x \in \texttt{Dom}(P)$.
- Formally, a Boolean expression e is defined over an n-dimensional space as follows

Definition

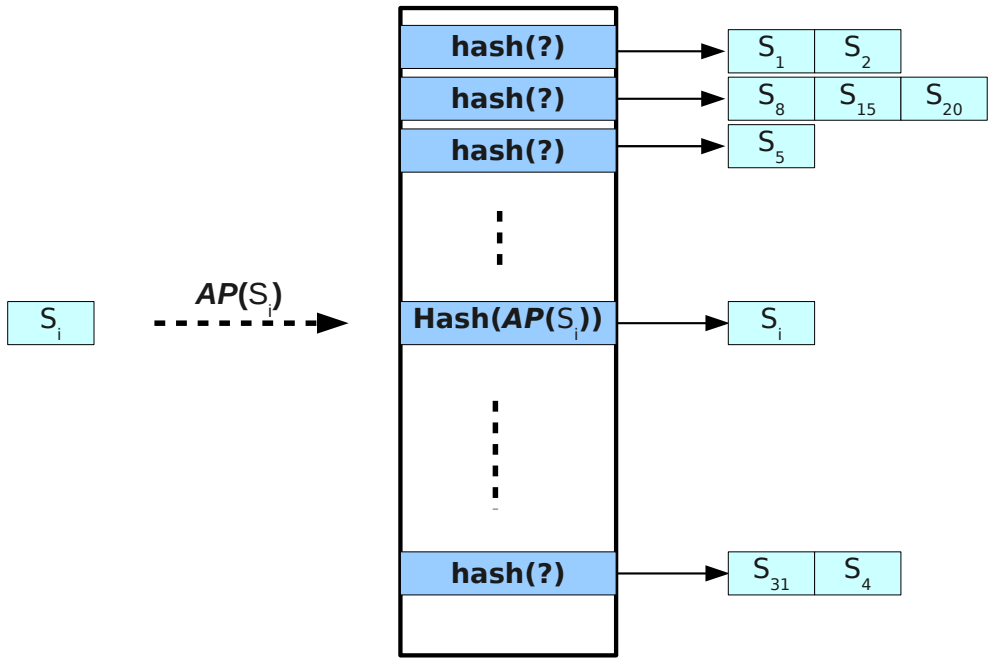
$$e = \left\{ P_1^{(\texttt{attr}_\texttt{i},\texttt{opt},\texttt{val})} \wedge \cdots \wedge P_k^{(\texttt{attr}_\texttt{j},\texttt{opt},\texttt{attr}_\texttt{l})} \right\}.$$

Matching Semantics

Matching Problem

Given a event e and a set of subscriptions $\mathbf{s}$, find all subscriptions $s_i \in \mathbf{s}$ satisfied by e.

An Abstract View of Propagation Data Structure



Propagation (Fabret, Jacobsen, Llirbat, Pereira, Ross, and Shasha, SIGMOD'01)

1. Distribute subscriptions in disjoint clusters to achieve high degree of parallelism
2. Store subscriptions as contiguous blocks of memory, which enables fast sequential access to improve memory locality

UNIVERSITY OF TORONTO

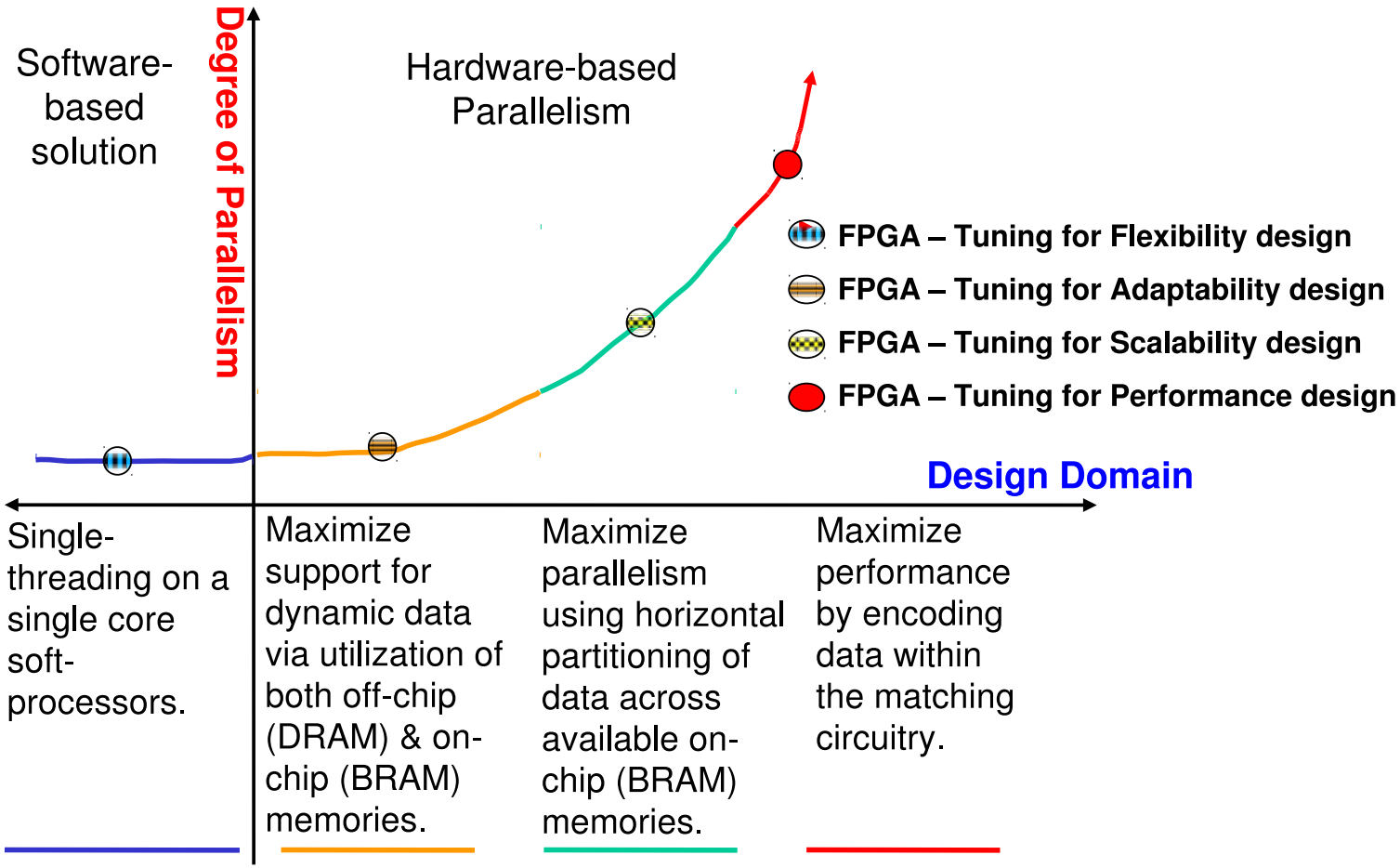
MIDDLEWARE SYSTEMS RESEARCH GROUP

An Overview of Our Design Space

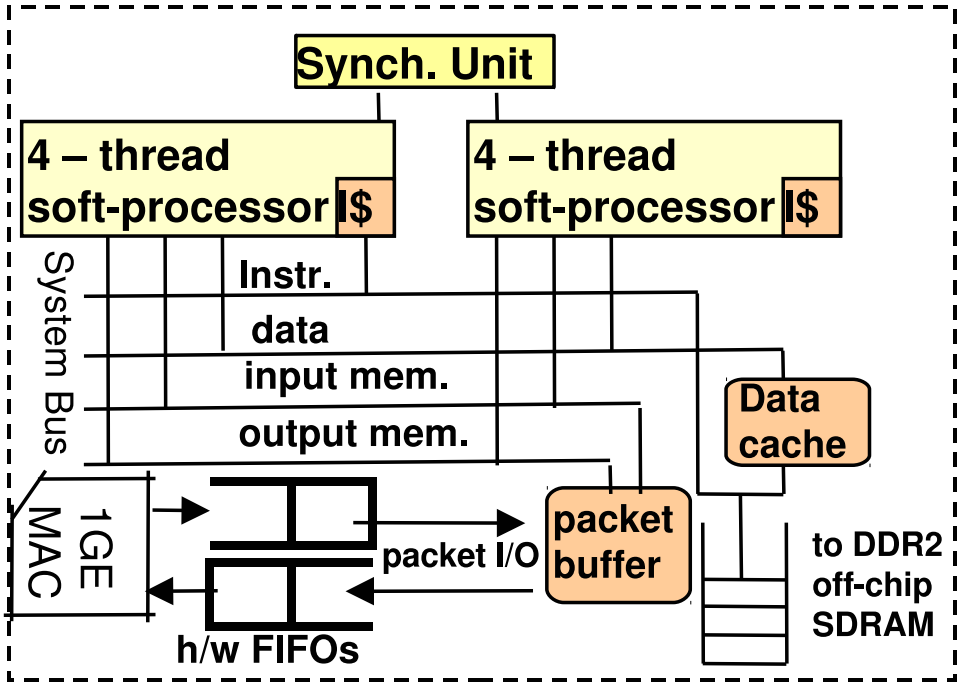
Four key designs

1. **Flexibility:** providing ease of the development and deployment cycle
2. **Adaptability:** supporting subscription updates
3. **Scalability:** relying on horizontal data partitioning
4. **Performance:** achieving the highest level of parallelism

Degrees of Offered Parallelism

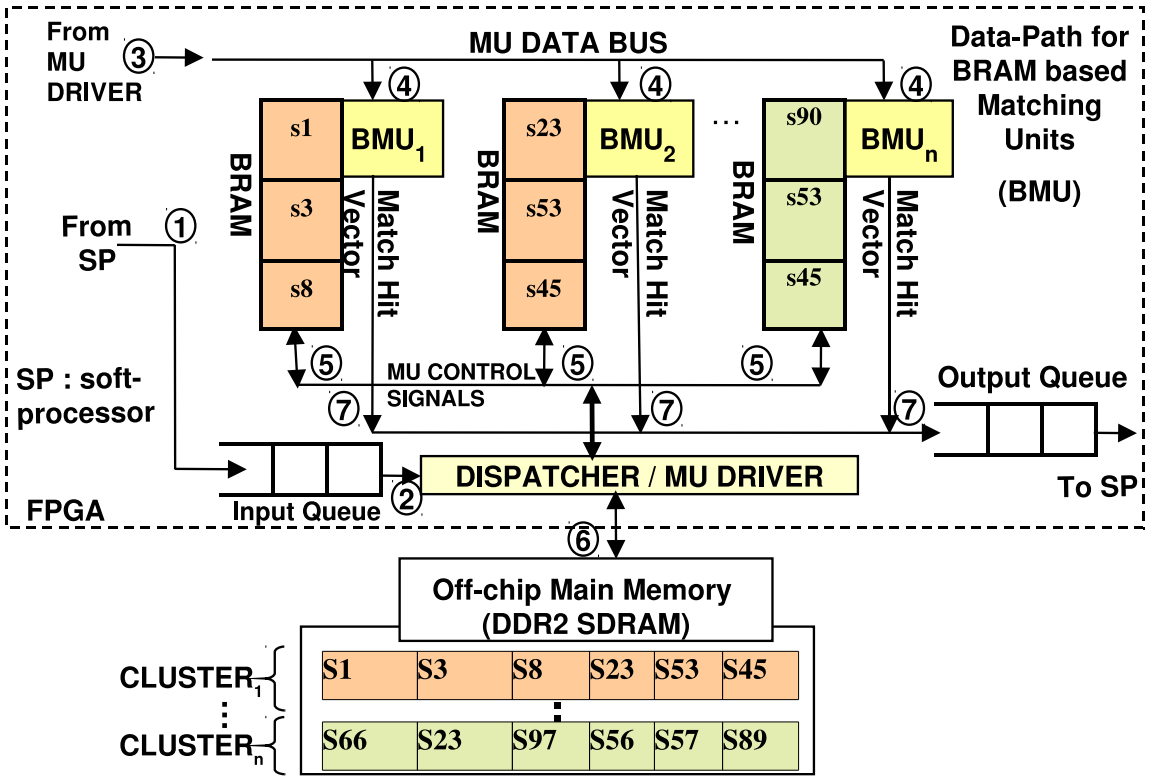


Tuning for Flexibility



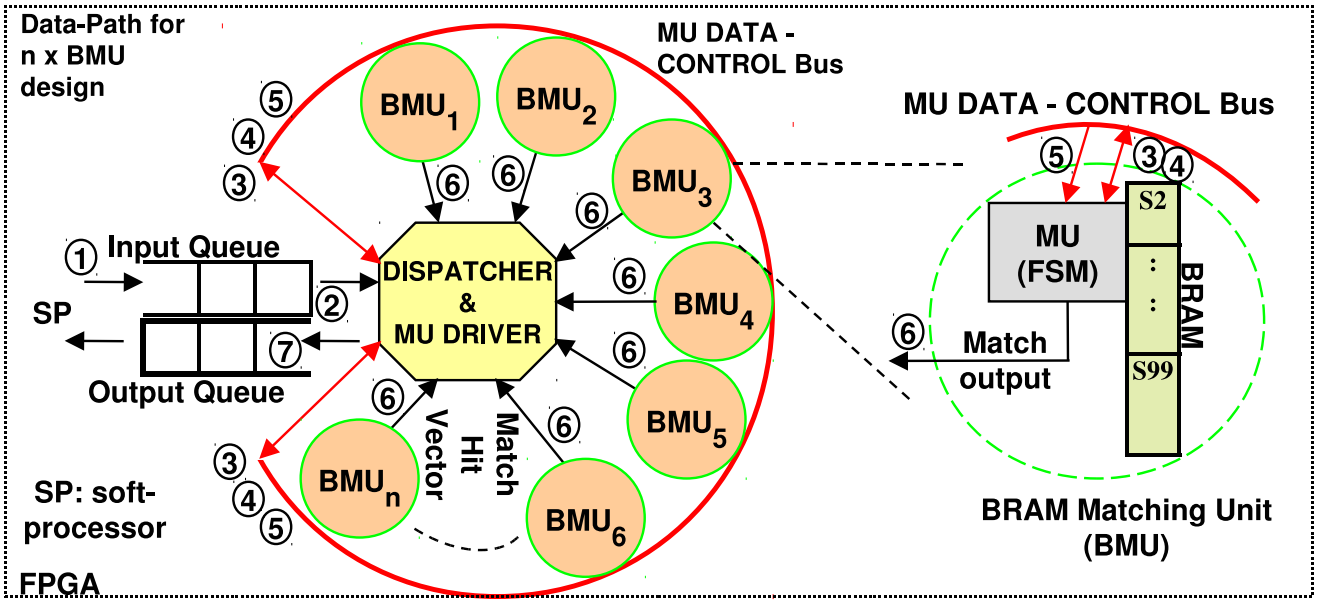
1. Eliminate software-to-hardware porting effort or the need for specialized hardware knowledge
2. Compile and execute the original PC source code on FPGA soft-core processors

Tuning for Adaptability



1. Employ a shared memory model to store propagation data structure
2. Support up to four matching units (custom processors) in parallel (parallelism limited by the off-chip memory-to-processor bandwidth)
3. Scale up to hundreds of thousands of subscriptions and support subscription updates

Tuning for Scalability

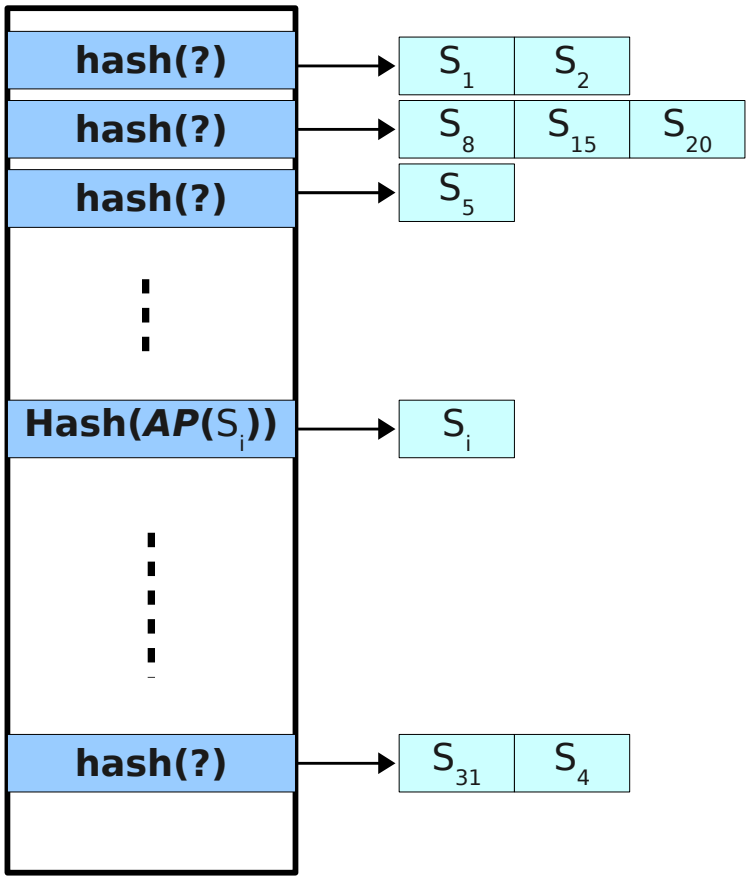


1. Inherit most of the benefits of *tuning for adaptability design*
2. Obtain an unprecedented level parallelism through horizontal data partitioning
3. Assign each matching unit a dedicated on-chip memory to minimize processor idleness
4. Realize full memory-to-processor bandwidth match via direct interconnect to dedicated memory units

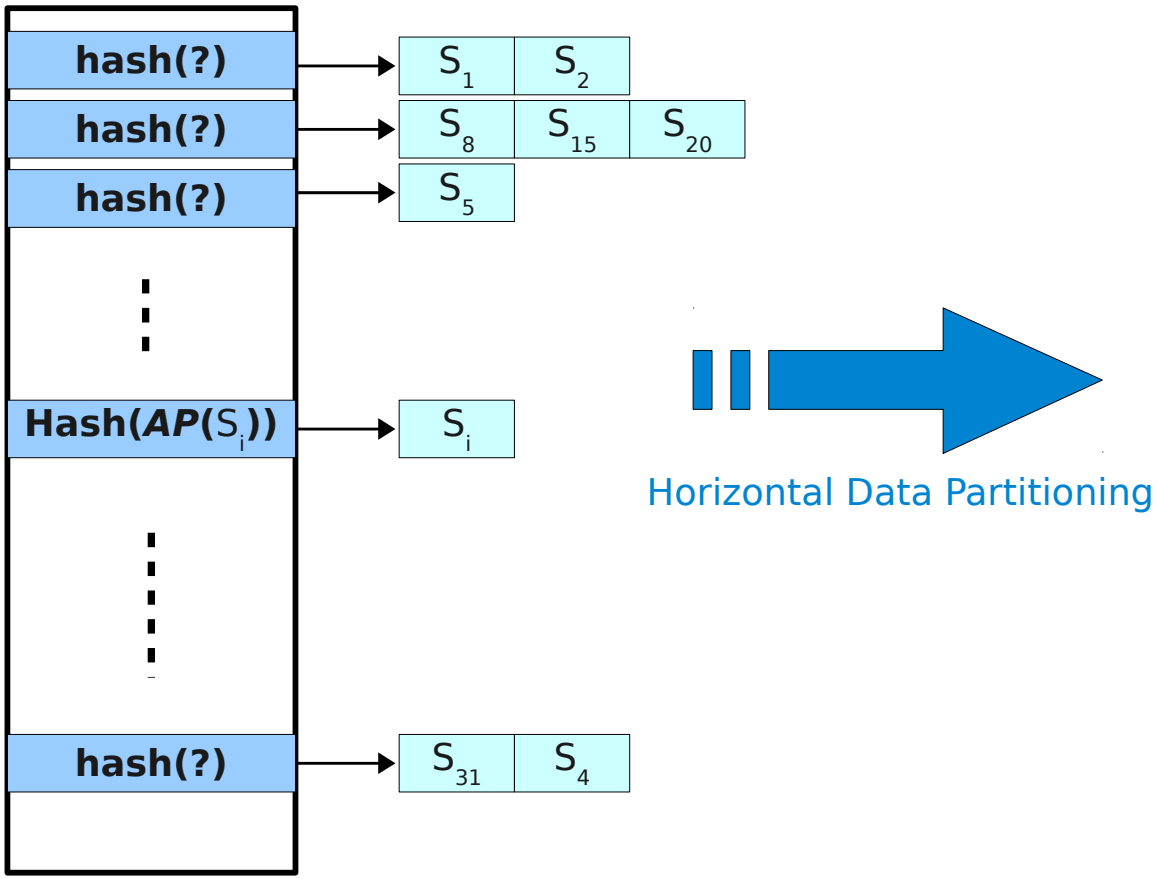
UNIVERSITY OF TORONTO

MIDDLEWARE SYSTEMS RESEARCH GROUP

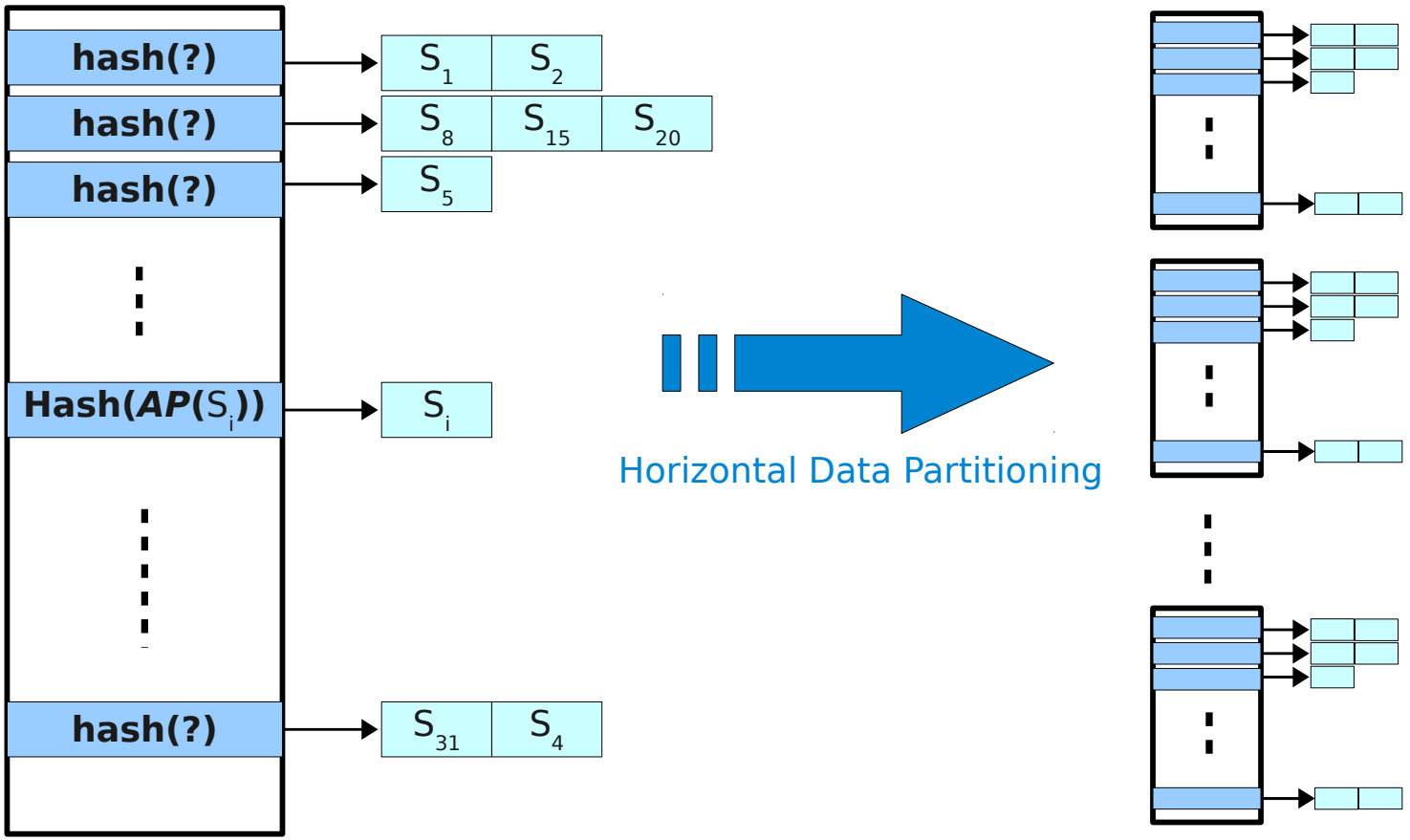
Horizontal Data Partitioning



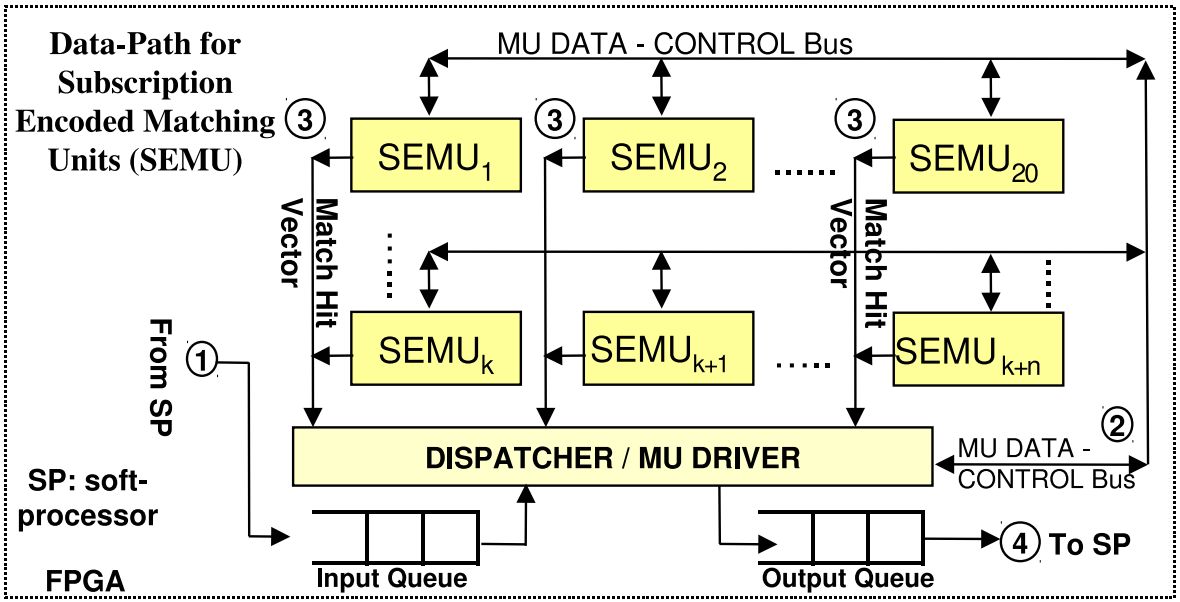
Horizontal Data Partitioning



Horizontal Data Partitioning



Tuning for Performance



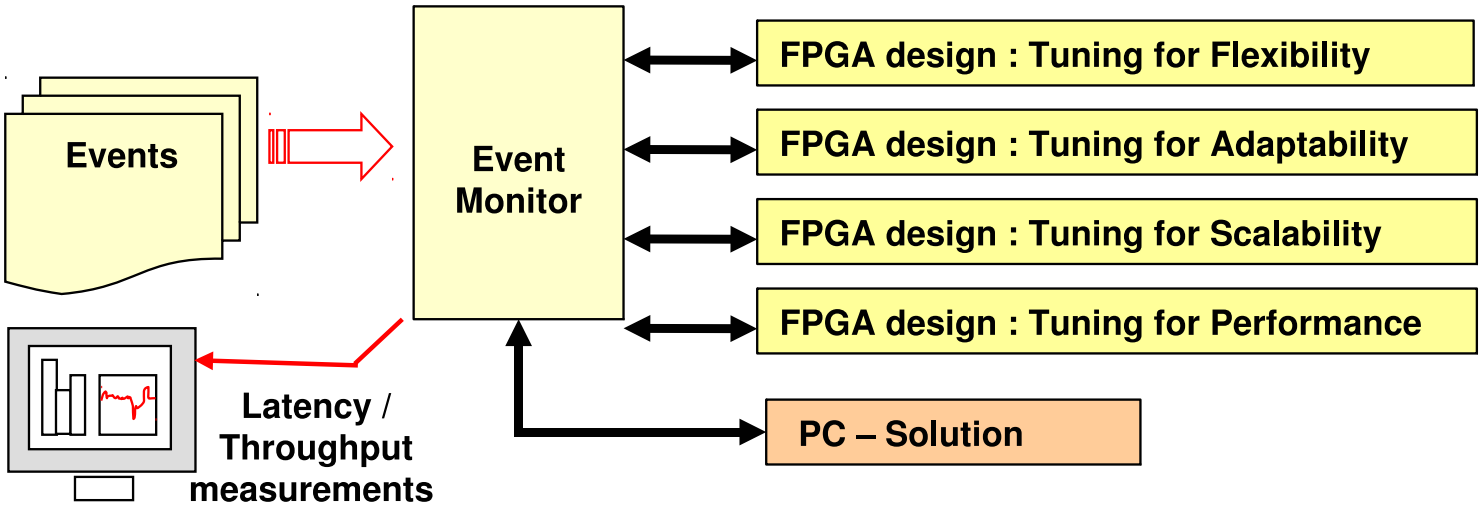
1. Encode each subscription as a matching unit (a custom processor)
2. Sustain a high rate of matching due to lack of memory access
3. Achieve a highest degree of parallelism, since all subscriptions are executed in parallel
4. Scale up to hundreds of subscriptions

Evaluation Design Space

Different Approaches

1. **PC:** PC Solution
2. **Flexibility:** FPGA embedded system (soft-core)
3. **Adaptability:** FPGA matching units (processors) + off-chip main memory
4. **Scalability:** FPGA matching units (processors) + on-chip main memory + horizontal data partitioning
5. **Performance:** Hardware encoded data + no on-/off-chip memory

Evaluation Testbed



1. **Throughput** is the maximum sustainable input packet rate, determined through a bisection search, when no packets is dropped.
2. **Latency** is the interval between the time an event packet leaves the Event Monitor output queue to the time the action is received.

Effect of the # of Matching Units (MUs) vs. Latency (μs)

Scalability Design Results

Workload	1x MU	4x MUs	32x MUs	128x MUs
250	7.5	5.5	5.0	**3.6**
1K	9.3	6.1	4.3	**4.3**
10K	64.0	19.0	6.8	**5.4**
50K	223.5	59.9	12.3	**7.3**

Effect of Workload Size vs. End-to-end Latency (μs)

Workload	PC	Flexibility	Adaptability	Scalability	Performance
250	**53.9**	71.0	6.4	**3.6**	**3.2**
1K	**60.7**	199.4	7.5	**4.3**	N/A
10K	**150.0**	1,617.8	87.8	**5.4**	N/A
100K	2,001.2	16,422.8	1,307.3	N/A	N/A

System Throughput (events/sec)

Workload	PC	Flexibility	Adaptability	Scalability	Performance
250	**122,654**	14,671	282,142	**740,740**	**1,024,590**
1K	**66760**	5,089	202,500	**487,804**	N/A
10K	**9594**	619	11,779	**317,460**	N/A
100K	511	60	766	N/A	N/A

UNIVERSITY OF TORONTO

MIDDLEWARE SYSTEMS RESEARCH GROUP

Percentage (%) of Line-rate Utilization

Workload	PC	Flexibility	Adaptability	Scalability	Performance
250	**7.45**	0.89	17.15	**45.04**	**62.29**
1K	**4.01**	3.09	12.31	**29.66**	N/A
10K	**0.58**	0.04	0.72	**19.30**	N/A
100K	0.031	0.01	0.05	N/A	N/A

UNIVERSITY OF TORONTO

MIDDLEWARE SYSTEMS RESEARCH GROUP

Conclusions

Conclusions

1. Reconfigurable hardware (FPGA)
   - accelerate using custom logic circuit
   - utilize hardware parallelism

Conclusions

1. Reconfigurable hardware (FPGA)
   - accelerate using custom logic circuit
   - utilize hardware parallelism
2. Line-rate event processing
   - eliminate OS layer latency
   - leverage on-board packet processing

Conclusions

1. Reconfigurable hardware (FPGA)
   - accelerate using custom logic circuit
   - utilize hardware parallelism
2. Line-rate event processing
   - eliminate OS layer latency
   - leverage on-board packet processing
3. Effective data placement of subscriptions
   - horizontally partition the data (propagation data structure)
   - increase the memory bandwidth
   - maximize the level of parallelism

Conclusions

1. Reconfigurable hardware (FPGA)
   - accelerate using custom logic circuit
   - utilize hardware parallelism
2. Line-rate event processing
   - eliminate OS layer latency
   - leverage on-board packet processing
3. Effective data placement of subscriptions
   - horizontally partition the data (propagation data structure)
   - increase the memory bandwidth
   - maximize the level of parallelism
4. Other FPGAs benefits
   - cost-effective
   - energy-efficient

Thank You,