A Reconfigurable System based on a Parallel and Pipelined solution for Regular Expression Matching

FPL 2010

F. Bruschi, M. Paolieri, V. Rana
Vincenzo Rana <rana@elet.polimi.it>
Outline

- Introduction
- State of the Art
- Proposed Approach (REMA)
  - Goals
  - Main Idea
  - REMA implementation details
  - REMA reconfiguration
- Experimental Results
- Conclusions and Future Work
Introduction

- Pattern Matching
  - Search for a specific *pattern* into a large amount of data
  - Very computational expensive task
  - Large spectrum of applications
    - DNA sequence matching
    - Data compression
    - Natural language analysis
    - Network intrusion detection systems
  - *High-throughput* solutions required
State of the Art (software)

- Software solutions
  - FSA/NFA
    - Sequential processing
  - POSIX standard
    - Grep
      - http://www.gnu.org/software/grep
State of the Art (hardware)

- Hardware solutions
  - [SID01]
    - O(n²) memory space, O(1) char/cc
    - Additional resources to build the NFA
  - [BIS06]
    - The output of the tool is a VHDL representations ready for logic synthesis
  - [SOU03]
    - They developed a C program that automatically generates the desired VHDL representation directly from pattern matching expressions

[SID01] Fast Regular Expression Matching using FPGAs
Reetinder Sidhu and Viktor Prasanna

[BIS06] Regular Expression Matching for Reconfigurable Packet Inspection
Joao Bispo, Ioannis Sourdis, Joao M.P. Cardoso and Stamatis Vassiliadis

[SOU03] Fast, Large-Scale String Match for a 10Gbps FPGA-based Network Intrusion Detection System
Ioannis Sourdis and Dionisios Pnevmatikatos
State of the Art (summary)

- Main limit of current SoA software solutions
  - Very low throughput
    - Need for performance improvement

- Characteristics of current SoA hardware solutions
  - Non-deterministic Finite Automaton
  - Implementation strongly dependent from RE
  - VHDL regeneration necessary for new patterns
    - New synthesis process required
REMA Goals

- Development of a reconfigurable architecture that performs the pattern matching task exploiting a set of reconfigurable cores working in parallel.

- **Flexibility** in being able to adapt the matching machine at run-time, without requiring any further synthesis of the HDL code.

- Definition of a methodology to map, through a dedicated compiler, a Regular Expression into a set of instructions to be executed by a set of reconfigurable cores.
Consider Regular Expressions as a *programming language*

- A Regular Expression can be seen as a sequence of conditions to be matched
- Each condition can be considered as an *instruction*
  - In our approach, an instruction - part of the RE - is composed by an *opcode* and some *reference characters*

- The program continues only if an instruction is valid (i.e. matched)
Main idea (2/2)

- If an instruction fails, the program is restarted
- Whenever the instructions are executed, the pattern has been found
- Simple example:
  - RE = (ABCD) | (aacde)
  - 4 comparison units

( call
  ABCD compare text with “ABCD”
) | return: process OR
( call
  aacd compare text with “aacd”
e) compare text with “e” and return, overall evaluation
Nop end of RE
Proposed approach
REMA implementation details

- REMA processor (first versions [PAO07] [BON08])
  - Fetch a RE from the instruction memory
  - Perform the matching with the data memory

- Architecture
  - Two-stages pipeline
  - Parallel accesses to memories
  - Parallel execution of multiple comparisons
  - Instructions- and data-prefetching to avoid pipeline stalls

- Compiler
  - Translation of standard REs in REMA instructions

[PAO07] Recpu: A parallel and pipelined architecture for regular expression matching
  Marco Paolieri, Ivano Bonesana, and Marco D. Santambrogio

[BON08] An adaptable FPGA-based system for regular expression matching
  I. Bonesana, M. Paolieri, and M.D. Santambrogio
Multiple Parallel Processing Units

- Execution example
  - RE = AABC
  - 4 cluster units
Handling Special Cases

- RE operators are treated in the following way:
  - *and, or, not*: directly handled
  - +, *: treated as **loops**
  - nested parenthesis by means of **function-call technique** (stack)
Simple Example

- RE = \((ab|c)^*cabab(ef|g)\)
- Input string = \(abcabccababg\)

- Case 1: “ab” matches “\((ab|c)^*\)”
  - but “cabccababg” does not match “\(cabab(ef|g)\)”
- Case 2: “abcab” matches “\((ab|c)^*\)”
  - but “ccababg” does not match “\(cabab(ef|g)\)”
- Case 3: “abcabc” matches “\((ab|c)^*\)”
  - and “cababg” matches “\(cabab(ef|g)\)”

A sequential exploration of the solution space can bring to the evaluation of a huge amount of combinations that will fail
REMA reconfiguration

- Each core addresses one of the subtasks into which the Regular Expression matching problem is partitioned.
- These cores work in parallel on the same string:
  - They analyze different possible matchings.
  - Pattern matching takes the time of the fastest module.
- The run-time reconfigurability feature allows to implement a *just in time* logic usage strategy.
Experimental results (1/2)

- Comparison with Grep on a file of 65 K characters

<table>
<thead>
<tr>
<th>Pattern</th>
<th>grep</th>
<th>REMA (without reconfiguration overhead)</th>
<th>Reconfiguration overhead</th>
<th>REMA (with reconfiguration overhead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>F</td>
<td>G</td>
<td>HAA</td>
<td>32.7 ms</td>
</tr>
<tr>
<td>ABCD</td>
<td>32.8 ms</td>
<td>14.01 us</td>
<td>33.7 ms</td>
<td>33.7 ms</td>
</tr>
<tr>
<td>(ABCD)+</td>
<td>393.1 ms</td>
<td>26.2 us</td>
<td>33.7 ms</td>
<td>33.7 ms</td>
</tr>
</tbody>
</table>

- Area usage of a single REMA core on a Virtex-IV device (xc4vfx60)
  - 471 slices (out of 25280)
  - Up to 50 REMA cores can be configured on the selected device
Experimental results (2/2)

- Comparison with other literature hardware solutions

<table>
<thead>
<tr>
<th>Solution published in</th>
<th>Bit-Rate (Gbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BIS06]</td>
<td>From 2 to 2.9</td>
</tr>
<tr>
<td>[CHO05]</td>
<td>7.14</td>
</tr>
<tr>
<td>[SOU03]</td>
<td>From 5.47 to 12.67</td>
</tr>
<tr>
<td>[SID01]</td>
<td>Around 0.8</td>
</tr>
<tr>
<td>REMA</td>
<td>From 10.19 to 18.18</td>
</tr>
</tbody>
</table>

[BIS06] Regular Expression Matching for Reconfigurable Packet Inspection
Joao Bispo, Ioannis Sourdis, Joao M.P. Cardoso and Stamatis Vassiliadis

[CHO05] A Pattern Matching Co-processor for Network Security
Young H. Cho and William H. Mangione-Smith

[SOU03] Fast, Large-Scale String Match for a 10Gbps FPGA-based Network Intrusion Detection System
Ioannis Sourdis and Dionisios Pnevmatikatos

[SID01] Fast Regular Expression Matching using FPGAs
Reetinder Sidhu and Viktor Prasanna
Concluding remarks

- High performance solutions for pattern matching are required
- Necessary to move towards hardware solutions
- A novel approach for hardware implementation of pattern matching has been proposed
  - RE as a programming language
  - Exploiting parallelism and reconfiguration to increase throughput and flexibility
Future work

- Improve the compiler-side of the flow in order to optimize the dispatching of the subtasks to the REMA cores
- Improve the analysis of the performance of the proposed solution w.r.t. the number of REMA cores available on the device
Questions?

Thank you for your attention!