Case Studies in Hardware Xpath Acceleration

SYSTOR’11, May 30–June 1, 2011, Haifa, Israel.

Dorit Nuzman, Victor Kaplansky, Sergei Dyshel, Alon Dayan
David Maze, Matthias Nicola, Glenn Marcy
Main Goal and Results.

Acceleration of Xpath processing by Hardware in two real world applications – WBM and DB2-pureXML.

Websphere Business Monitor – 27% improvement in total running time.  
DB2-pureXML – up to x6.2 improvement in total query processing time.
IBM's Power Edge of Network (PowerEN)

6. XSLT transformation

5. XPath Matching (filtering/tagging)

4. XML parsing

3. HTTP handling

2. SSL decryption

1. receive network traffic

7. output

XML

XPath compiler

Pervasive Logic

PIC

Crypto

Comp / Decompon

Pattern Matching Engine

Flash ROM and Misc IO Logic

Bus Internal I/F Controllers

Bus External I/F Controller

Bus

Ethernet Packet Offload Engine

PCI Express Gen. 2

Flash ROM

XML

XPath

Pattern Matching Engine

Comp / Decompon

Crypto

Pervasive Logic

PIC

2x 1GE MAC

4x 10GE MAC

x4 PHY

x4 PHY

x4 PHY

x4 PHY

x4 PHY

x4 PHY

x1 PHY

x1 PHY

4B + 4B Eli3

4B + 4B Eli3

4B + 4B Eli3

x4 PHY

x4 PHY

x4 PHY

x4 PHY

x4 PHY

x4 PHY

x1 PHY

x1 PHY

Misc I/O

1. receive network traffic

2. SSL decryption

3. HTTP handling

4. XML parsing

5. XPath Matching (filtering/tagging)

6. XSLT transformation

7. output
XPath is a language used to navigate through elements and attributes in an XML document.
XPath acceleration opportunities: 1) XML in Healthcare / XML Databases

```
<SubjectData ID="ab123">
  <gender>male</gender>
  <handedness>left</handedness>
  <dob>1967-08-13</dob>
  <education>30</education>
  <ses>2</ses>
</SubjectData>
```

Clinical XML data

XML: de-facto standard for electronic medical health record interoperability

Hospital

Appliance

PowerEN

DICOM

PACS

DB2-pureXML

XNAT

Medical images
Current DB2-pureXML flow

Processing may consist of:

1. Table operations on indexed elements
2. Navigation of the XML documents
Proposed DB2-pureXML flow with PowerEN

1) indexed part: Filter documents (table rows)

2) XML navigation part: Selects relevant parts from the documents (XPath matching within documents)

Result:

<table>
<thead>
<tr>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>52</td>
</tr>
</tbody>
</table>
XPath acceleration opportunities: 2) WebSphere Business Monitor

(A) XPath expression list
1: cbe:CommonBaseEvents/cbe:CommonBaseEvent/@globalInstanceId
2: cbe:CommonBaseEvents/cbe:CommonBaseEvent/@creationTime
3: wbi:event/wbi:eventHeaderData/wbi:ECSCurrentID/text()
4: wbi:event/wbi:eventHeaderData/wbi:ECSParentID/text()
5: wbi:event/wbi:eventPointData/wbi:eventNature/text()
6: wbi:event/wbi:eventPointData/bpc:processTemplateName/text()
7: wbi:event/wbi:eventPointData/bpc:bpelId/text()

(B) Incoming CBE event
<
cbe:CommonBaseEvents
  ="...">cbe:CommonBaseEvent globalInstanceId="..." creationTime <
cbe:contextDataElements name="WBIEventVersion" <
    type="string
  >cbe:contextValue>6.1</cbe:contextValue <
cbe:contextDataElements> </
    <wbi:event
        wbi:eventHeaderData <
        wbi:ECSCurrentID>...</wbi:ECSCurrentID <
        wbi:ECSParentID>...</wbi:ECSParentID <
        wbi:eventPointData <
        wbi:eventNature>ENTRY</wbi:eventNature <
        bpc:BPCEventCode>21000</bpc:BPCEventCode <
        bpc:processTemplateName>...</bpc:processTemplateName <
        wbi:eventPointData </
        wbi:event <
    cbe:CommonBaseEvent <
> cbe:CommonBaseEvents</

(C) WBM
(1) Setup
Register XPaths, create compiler

(2) Compile
XPaths

(3) XML processing
(a) parse, (b) XPath tag/filter,
(c) deliver matched values,
(d) fill in expression table

(4) Business processing
Apply processing methods according to keyed values

(D) expression keyed
table of matching values

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>globalInstanceId=&quot;...&quot;</td>
</tr>
<tr>
<td>2</td>
<td>creationTime=&quot;...&quot;</td>
</tr>
<tr>
<td>3</td>
<td><a href="">wbi:ECSCurrentID</a>.../wbi:ECSCurrentID</td>
</tr>
<tr>
<td>4</td>
<td><a href="">wbi:ECSParentID</a>.../wbi:ECSParentID</td>
</tr>
<tr>
<td>5</td>
<td><a href="">wbi:eventNature</a>ENTRY&lt;/wbi:eventNature</td>
</tr>
<tr>
<td>6</td>
<td><a href="">bpc:processTemplateName</a>...&lt;/bpc:processTemplateName</td>
</tr>
<tr>
<td>7</td>
<td>null</td>
</tr>
</tbody>
</table>

(B) Incoming CBE event

(A) XPath expression list
Technical details

1) Compiler

2) XML accel. Code (PPE program)

3) XML accelerator

4) Matching data items

5) Bridge layer

6) Query results

XPath file

XML file

PPE
A few technical details: 1) the XPath compiler

Filter: /catalog/book
Tag: //year
/catalog/cd
A few technical details: 1) the XPath compiler - cont.

Filter: /catalog/book
Tag: //year
/catalog/cd

Note: streamable XPaths only (don’t support /catalog/book[special-edition]/year)
A few technical details: 2) the bridge layer

XPath

XCI program (Example1.java)
1) Registration and Initialization
2) Prepare(): compile XPath “//year”
3) Execute(): create a cursor to navigate to matching locations
4) Navigate (toNext(), fork(), toChildren(), toAttributes())

PowerEN XCI adapter layer

PowerEN Java layer

PowerEN C layer (libxj.so, XG5 card)

XJ

XG5

XPath-matcher (xpd_executable)
The integrated experiment, Using JDBC

(1) Filter documents (rows)
(2) Navigate the parsed document to find matches
(3) Serialize the results
(4) Transmit the results to the client

(1) DB2 filter and serialize documents
(2) Send the XML document from host to Prism
(3) Parse the document to find matches (+ compile the XPath query into a program that would run on the XML accelerator)
(4) Send the results back to the host
(5) Serialize the results
(6) Transmit the results to the client

Processor:
1) dual x86 Harpertown Processors @2.83GHz
2) PRISM offloading the XML processing
XPath query acceleration speedups, HL7

Many matches, large output

Many matches, small output

Count (many matches, no output)

Single match, large output

<table>
<thead>
<tr>
<th>Query</th>
<th>3.2MBx10</th>
<th>320KBx100</th>
<th>32KBx1000</th>
<th>4.3KBX10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Breakdown of accelerated path, HL7 query

- DB2 read and serialize documents
- Process query
- Serialize results

<table>
<thead>
<tr>
<th>Query</th>
<th>DB2 read and serialize documents</th>
<th>Process query</th>
<th>Serialize results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query1</td>
<td>[3.2MBx10, 320KBx100, 32KBx1000, 4.3KBx10000]</td>
<td>[3.2MBx10, 320KBx100, 32KBx10000]</td>
<td>4.3KBx10000</td>
</tr>
<tr>
<td>Query2</td>
<td>[3.2MBx10, 320KBx100, 32KBx10000]</td>
<td>[3.2MBx10, 320KBx100, 32KBx10000]</td>
<td>4.3KBx10000</td>
</tr>
<tr>
<td>Query3</td>
<td>[3.2MBx10, 320KBx100, 32KBx10000]</td>
<td>[3.2MBx10, 320KBx100, 32KBx10000]</td>
<td>4.3KBx10000</td>
</tr>
<tr>
<td>Query4</td>
<td>[3.2MBx10, 320KBx100, 32KBx10000]</td>
<td>3.2MBx10</td>
<td>320KBx100</td>
</tr>
<tr>
<td>Query5</td>
<td>[3.2MBx10, 320KBx100, 32KBx10000]</td>
<td>3.2MBx10</td>
<td>320KBx100</td>
</tr>
</tbody>
</table>

- Many matches, large output
- Many matches, large output
- Many matches, small output
- Count (many matches, no output)
- Single match, large output
Websphere Business Monitor acceleration speedups

- XML processing part improved by 27% → WBM Overall application improved by 11%

  An efficient bridging layer is critical for overall accelerated performance
  - buffering of requests to the accelerator
  - reduced JNI calls/Java–C conversions

Applications have to use the “right” API
Conclusions:

- High potential for acceleration can be found in applications using large documents and XPath queries matching large numbers of XML nodes and producing large outputs, such as in the healthcare and life sciences domains.

- Limited potential for acceleration can be found in applications using small documents and XPath requests matching small numbers of XML nodes or producing small outputs, such as in the event processing and financial domains.

- An efficient bridging layer is critical for overall accelerated performance. Optimizations to the software bridging layers, such as buffering of requests to the accelerator, reduced JNI calls and Java–C conversion overheads, yielding a 33% improvement to the WBM accelerated path, and up to 2.7x improvement to the HL7 accelerated query processing path.
Future Work

- extend the applicability of XPath acceleration coprocessors

- increase speedups:
  Devise a cost model that can automatically identify scenarios that can profit from XPath acceleration.

  Extend XML APIs to express more involved XPath scenarios (such as simultaneous filtering and tagging, and multi-step XML processing).

  Specifically in the native XML Database domain, data serialization costs are relatively high, and support for compact data formats by the hardware XPath accelerator is critical.
The End

Questions?