International Characters

Method and Apparatus for Processing Character Streams
"Except insofar as ways can be found to use ... processors more efficiently, e.g. by exploiting ... on chip SIMD accelerators, gains from further tuning or alternative approaches [to XML processing and parsing] are likely to be modest."

SIMD
Single-Instruction, Multiple-Data

Computer instructions for performing operations on multiple data elements per register

XMM registers of Pentium-compatible processors and Altivec registers of Power PC processors are 128 bits wide

The key concept in Rob Cameron’s inventions: Character streams represented as parallel data streams allow 128 code units per register!
Parallel UTF-8 Data Streams

<table>
<thead>
<tr>
<th>Text</th>
<th>$</th>
<th>,</th>
<th>¥</th>
<th>,</th>
<th>€</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>0x24</td>
<td>0x2C</td>
<td>0x20</td>
<td>0xC2</td>
<td>0xA5</td>
<td>0x2C</td>
</tr>
<tr>
<td>Bit0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Bit1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bit3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Bit5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bit6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Eight 128-bit registers can each hold one bit from 128 bytes.
Identify UTF-8 Prefix Bytes

Bit0 \land Bit1

<table>
<thead>
<tr>
<th>Text</th>
<th>$</th>
<th>,</th>
<th>¥</th>
<th>,</th>
<th>€</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>0x24</td>
<td>0x2C</td>
<td>0x20</td>
<td>0xC2</td>
<td>0xA5</td>
<td>0x2C</td>
</tr>
<tr>
<td>Bit0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Bit1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bit3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Bit5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bit6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Do this 128 bytes at a time using 128-bit SIMD registers.
Identify UTF-8 Prefix Bytes

Identify all UTF-8 prefix bytes in a 128-byte stream:

\[ \text{Prefix}(i) = \text{Bit0}(i) \land \text{Bit1}(i) \]

Identify all UTF-8 two-byte prefix bytes in a 128-byte stream:

\[ \text{Prefix2}(i) = \text{Prefix}(i) \land \neg \text{Bit2}(i) \]

Prior art using a byte-at-a-time loop for determining and classifying prefix bytes would require several hundred operations to process 128 bytes.
Identify UTF-8 Violations

Prefix(i) = Bit0(i) \land Bit1(i)
Suffix(i) = Bit0(i) \land \neg Bit1(i)
Prefix2(i) = Prefix(i) \land \neg Bit2(i)
Prefix3or4(i) = Prefix(i) \land Bit2(i)
Prefix3(i) = Prefix3or4(i) \land \neg Bit3(i)
Prefix4(i) = Prefix3or4(i) \land Bit3(i)
Badpfx2(i) = Prefix2(i) \land
\neg (Bit3(i) \lor Bit4(i) \lor Bit5(i) \lor Bit6(i))
Badpfx4(i) = Prefix4(i) \land
(Bit4(i) \lor Bit5(i) \land (Bit6(i) \lor Bit7(i)))
Scope22(i) = Prefix2(i-1)
Scope32(i) = Prefix3(i-1)
Scope33(i) = Prefix3(i-2)
Scope42(i) = Prefix4(i-1)
Scope43(i) = Prefix4(i-2)
Scope44(i) = Prefix4(i-3)
Special(i) = \neg Bit6(i) \land \neg (Bit4(i) \oplus Bit7(i) \land
(Bit3(i) \lor (\neg (Bit4(i) \oplus Bit5(i))))
Anyscope(i) = Scope22(i) \lor Scope32(i) \lor
Scope33(i) \lor Scope42(i) \lor
Scope43(i) \lor Scope44(i)
Mismatch(i) = Anyscope(i) \oplus Suffix(i)
Badsfx32(i) = Scope32(i) \land Special(i-1) \land
\neg (Bit4(i-1) \oplus Bit2(i))
Badsfx42(i) = Scope42(i) \land Special(i-1) \land
\neg (Bit4(i-1) \oplus (Bit2(i) \lor Bit3(i)))
Invalid(i) = Badpfx2(i) \lor Badpfx4(i) \lor
Mismatch(i) \lor Badsfx32(i) \lor Badsfx42(i)

36 logic (and, or, xor) and 8 shift operations (i-x) per 128 bytes (0.34 cycles/byte)!

Copyright (C) International Characters Inc. 2006.
Patent-Pending Methods

- Transposing serial bytes to/from parallel bit streams
- UTF-8 byte classification, validation and decoding
- UTF-16 validation
- ASCII optimization
- Transcoding between UTF-8 and UTF-16
- Exact string and exact set search
- Search with single- and multi-character wildcards
- Caseless search
- Lexical processing of textual notation
- XML character validation and line break normalization
- XML and other types of lexical analysis
- New SIMD instructions for inductive doubling
Advantages of Parallel Processing

- Minimizes looping and branch misprediction.
- Can be effectively pipelined.
- Many fewer instructions per byte processed.
- Opportunity for improved performance by increasing SIMD register width.
- Opportunity for improved performance by implementing inductive doubling in SIMD.
- Potential energy and hardware cost savings in server farms.
Measures of Performance
UTF-8 to UTF-16 Transcoding

**iconv software**
- byte-at-a-time processing
- up to 7 branch instructions per byte

**u8u16 Transcoder from International Characters**
- 128 bytes at a time
- 4 branches per 128 bytes
- 7x to 45x faster on PowerPC with Altivec processors
- 20x faster typical on XML data
International Characters

Patent pending in the U.S. and other countries

Available for free for:

• Open source software
• Experimentation, research or teaching

Also available under commercial license from:

International Characters Inc.
2225 Folkestone Way
West Vancouver, B.C. V7S 2Y6, Canada
info@international-characters.com