Accelerated, Threaded XML Parsing

loading your documents quicker …

Matúš Kukan <matus.kukan@collabora.com>
Michael Meeks <michael.meeks@collabora.com>
matus & mmeeks, #libreoffice-dev, irc.freenode.net
Big data needs Document Load optimization
Parallelized Loading ...

- Desktop CPU cores are often idle.
- XML parsing:
  - The ideal application of parallelism
  - SAX parsers:
  - “Sucking icAche eXperience” parsers
    - read, parse a tiny piece of XML & emit an event ... punch that deep into the core of the APP logic, and return ..
    - Parse another tiny piece of XML.
  - Better APIs and impl's needed: Tokenizing, Namespace handling etc.
  - Luckily easy to retro-fit threading ...
  - Dozens of performance wins in XFastParser.
XML format lameness ...

- Spreadsheets have a great way of expressing repeated formulae:
  - **R1C1 notation:**
    - `=SUM($A$1:$A$5)-A1`
    - → `=SUM(R1C1:R5C1)-R[-2]C[-1]`
  - Looks ugly – but it's constant down a column.
  - Lunatic standardizers for ODF ( & OOXML ) ignored me on this ... 

- Formulae hard and expensive to parse, so don't ...
  - Predictive generation down a column & comparison.
    - Removes tons of token allocations etc.
Parallelised load: (boxes are threads).

- Split XML Parse & Sheet populate

- Parallelised Sheet Loading ...

- Parallel to GPU compilation

```
=COVAR(A1:A300,B1:B300)
→ OpenCL code
→ Ready to execute kernels
```

Tools->Options->Advanced->”Experimental Mode” required for parallel loading
FastSaxParserImpl::parseStream(...)
{ get event list from maPendingEvents; }

FastSaxParserImpl::consume(...)
{ based on event type, call ..,
while the list is not empty; }

Entity::startElement(...)
{ create child ContextHandler
and call 'startElement' on it;
then put it into stack; }

Entity::characters(...)
{ retrieve contextHandler and
call 'characters' on it; }

Entity::endElement(...)
{ call 'endElement' on
ContextHandler and
remove it from stack; }

xml::sax::XFastContextHandler
Interface implemented by the caller
of FastSaxParser::parseStream()
where actual processing
of tokenized elements happens.
```
ParserThread

FastSaxParserImpl::parse()

XML_Parse(...)

FastSaxParserImpl::callbackEndElement(...)  
{ pop from namespace stack 
  and create 'endElement' event; }

FastSaxParserImpl::callbackStartElement(...)  
{ create 'startElement' event - 
  contains list of attributes 
  with computed tokens; 
  push current namespace 
  into stack; }

FastSaxParserImpl::callbackCharacters(...)  
{ create 'characters' event - 
  contains string 
  inside an element; }

FastSaxParserImpl::produce(...)  
{ if limit is hit, push produced 
  events into maPendingEvents; }

maPendingEvents protected by mutex

xml::sax::XFastTokenHandler
Interface implemented by the caller
of FastSaxParser::parseStream() 
used to compute element tokens.
```
Code improvements

- We try hard to avoid any (de)allocations
  - FastSaxParserImpl::consume(EventList *pList) just processes the list of events without freeing any memory. pList is then pushed into mutex protected buffer and reused in FastSaxParserImpl::callbackFoo(...) functions.
  - commit “don't allocate uno::Sequences when we don't need to.”
    - Makes small strings reuse one sequence buffer
  - commit “FastAttributeList: avoid OStrings in attribute list; just use char buffer”
    - something like vector<char> wrt. allocations
    - easy to get attribute strings
Code improvements

- **cache values**
  - commit “fastparser: cache default namespace token for ooxml.”
  - commit “oox: special-case single-character a-z token mapping.”
    - 50% of OOXML tokens are primarily lower-case character, a-z
  - commit “fastparser: avoid std::stack::top() - cache it's results.”
    - amazingly std::stack::top() takes 146 pseudo-cycles to do not much, so instead cache the result in a single pointer in lieu of burning that code.
Does it work? with GPU enabled

Wall-clock time to load set of large XLSX spreadsheets: 8 thread Intel machine

Apologies for another log scale: Average 5X vs. 4.1.3
dates-worked.xlsx

- 1 sheet with half plain data and half formulas
- 4 sheets with both data and formulas
• 4 sheets of formulas
matrix-inverse.xlsx

- Mostly just 1 sheet with numbers
stock-history.xlsm

- Mostly 2 sheets with both data and formulas
1 sheet with a lot of numbers + 1 sheet with some formulas and diagrams

3.7X vs. 4.1.3

seconds
numbers-100k.xlsx

• 1 sheet with 100k numbers

4.4X vs. 4.1.3

seconds

Calc 4.1.3
Calc
Reference
numbers-formula-100k.xlsx

- 1 sheet with 100k numbers and 100k formulas

![Bar chart showing comparison between Calc 4.1.3 and 4.6X vs. 4.1.3 in seconds]
8 sheets with 100k numbers and 100k formulas
• 2 sheets with 1m numbers and 1m formulas each
Quick demo & questions on Calc / threaded loading bits?
LibreOffice is innovating:
  - Threaded parsing is just one example
    - a new, elegant, efficient means to parse XML with SAX

Plenty more to do
  - Next steps are porting more code to use XFastParser
  - AutoCorrection: huge dictionaries & slow parsing
  - ODF filters
  - XFastSerializer needs love etc.

Thanks for all of your help and testing!

Questions?