Improving Calc parallel calculations

By Luboš Luňák
Software Developer at Collabora Productivity
Introduction
## Typical spreadsheet

<table>
<thead>
<tr>
<th>Name</th>
<th>Value1</th>
<th>Value2</th>
<th>Value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>10</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>Item2</td>
<td>15</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>Item3</td>
<td>20</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## Typical spreadsheet (#2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value1</th>
<th>Value2</th>
<th>Value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>10</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>Item2</td>
<td>15</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>Item3</td>
<td>20</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Formula group 1:**
- Value1
- Value2
- Value3

**Formula group 2:**
- Value1
- Value2
- Value3
**Typical spreadsheet (#3)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value1</th>
<th>Value2</th>
<th>Value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>10</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>Item2</td>
<td>15</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>Item3</td>
<td>20</td>
<td>=Value*2</td>
<td>=Value1+Value2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Independent rows
Parallel calculation

Rows are often “the same” but independent

Modern CPUs do not improve single core performance that much

But they have more cores

-> It makes sense to compute in parallel
  • Reasonably simple
  • Should scale well
Implementation

ScFormulaCell

- Each spreadsheet cell

ScFormulaCellGroup

- Grouped cells sharing the same code

Make each thread calculate different cells in the same group
Lockless (mostly)

Threads operate on separate data → no need to lock

Shared instances → per-thread instances

Lock only if needed or if not performance critical
For more details,
talk by Tor from 2017
Problems
Static data

static OUString myCachedValue;

- Use thread_local
- Simply remove the optimization
- Add locking, if worth it (local mutex)
- ScInterpreterContext
- ...

Storing state in classes

class ... { ... int currentIndex; ... };

- Protect class use with a mutex
- Move state to its own class (e.g. iterators)
- Move state to a function parameter

This includes also various caching.
On-demand initialization

If( singleton == nullptr ) singleton = new Singleton;

- use C++11 thread-safe statics (required now by LO build)
  - static Singleton* singleton = new Singleton;
    - Leaks memory
  - Singleton* getSingleton() { static Singleton s; return &s; }
    - Either case cannot be cleaned up
  - comphelper::doubleCheckedInit( singleton, []() { return new Singleton; } )
Unsafe code

A lot of library code is not thread-safe (even our code)

- Fix the code (if possible)
- Add locking to the code (if worth it)
- Protect code use from Calc with a mutex
SolarMutex

SolarMutex is still held by main thread while threaded calculation is in progress

- If not done, other threads might interfere (UNO calls, clipboard thread)

→ Calculation threads may not access code requiring SolarMutex

Maybe needs a solution for some cases ???

- Transfer SolarMutex ownership?
- Ask main thread to perform an operation?
Threaded calc assert

```c
assert(!IsThreadedGroupCalcInProgress());
```

- Code is not meant to be run in threads
- Use the proper function (if exists)
- Make sure code in threads does not modify spreadsheet
ScMutationGuard assert

Code in calculation threads should not modify the document

- (Except for calculating cell results)
- Check your code
- Move code outside of calculation threads
(More) Solutions
Unsupported opcode/type

INDIRECT() (oclIndirect) – may possibly make cells dependent

ocExternal – external functionality (UNO calls)
- Hard to check all code
- May easily deadlock (SolarMutex)

DDE() (ocDDE) – LinkManager class uses extensive caching without locking

External references – ScExternalRefManager uses extensive caching without locking
Unsupported opcode/type (#2)

Simply blacklist all formulas containing problematic opcodes/types

ScTokenArray::CheckForThreading()}
**ScInterpreterContext**

Per-thread data structure, pointer to it passed around

Per-thread class instances

- `ScDocument::GetFormatTable() → context→mpFormatter`
ScInterpreterContext (#2)

Caches (VLOOKUP)

- Finding result of VLOOKUP may be expensive
- Same lookup used several columns in the same row
- Values must survive between thread invocations
- SetupFrom/MergeBackIntoNonThreadedContext()
ScInterpreterContext (#3)

Moving operation to the main thread

- ScDocument::setNumberFormat() is not thread-safe
- Calls to it can be postponed
- Save relevant data in ScInterpreterContext
- Actual call(s) performed by main thread after calculation threads finish
Add asserts

assert(!IsThreadedGroupCalcInProgress());

- Add wherever need (especially if unsure)
Helgrind (Valgrind)

Detecting thread problems from the Valgrind tools suite

- `VALGRIND=helgrind start_lo.sh`
- Slow
- Can still save time when finding difficult problems
Testing
Ensure threaded calculation is used

Threads vs OpenCL vs normal (non-threaded)

- Modify settings in UI
- Temporarily hardcode in CalcConfig class functions

Test even with small formula groups

- Group calculation is normally used only for larger groups
- mnOpenCLMinimumFormulaGroupSize
- Should be improved to make possible running tests for everything with the wanted calculation method
Thank you.

By Luboš Luňák
l.lunak@collabora.com