Calc: The challenges of scalable arithmetic

How threading can be challenging

Michael Meeks
General Manager at Collabora Productivity
michael.meeks@collabora.com
Skype - mmeeks,
G+ - mejmeeks@gmail.com

“Stand at the crossroads and look; ask for the ancient paths, ask where the good way is, and walk in it, and you will find rest for your souls...” - Jeremiah 6:16
Calc threading - Overview

- LibreOffice 6.0 Calc
- Existing structure & parallelism
- Why thread?
- The initial solution & problems
  - mis-factored code
  - dependency issues
- The group calculation piece
- Profiling & optimizing
- Future work & expansion …

Disclaimer & Thanks:
Almost all of this work was done by Tor Lillqvist & Dennis Francis – who can’t be here today.
Some great code reading & improvement.
LibreOffice 6.0 Calc ...

- A 30+ year old code-base
- Primary Data structures hugely improved recently
  - Still some scope for improvement: FormulaGroup vs. FormulaCell, per-cell dependency records etc.
- Calculation Engine in need of love
  - Some insights into how it works
  - Some problems wrt. threading.
Core structures since 4.3 (mdds::multi_type_vector)

ScDocument

- ScTable
  - ScColumn
    - svl::SharedString block
    - double block
      - EditTextObject block
    - Cell notes
    - Cell values
    - Text widths
    - Script types
    - Broadcasters

This bit:
Sample Token types (StackVar)
- svSingleRef → A1
- svDoubleRef → A1:C3
- svExternalSingleRef etc.
- svDouble → 42.0
- svString → “hello world”
- svByte → ocDiv, ocMacro ...
double ScFormulaCell::GetValue()
{
    MaybeInterpret();
    return GetRawValue();
}

void ScFormulaCell::Interpret()
{
    ... amazing recursion flattening ...
    InterpretTail() // ie. ...
    {
        ... new ScInterpreter( this, pDocument, rContext,
                                aPos, *pCode /* those tokens */);
        ->Interpret()
    }
}

StackVar ScInterpreter::Interpret()
{
    ... execute reverse-polish stack ...
    ... execute functions ...
    ... get cell values from references ...
InterpretFormulaGroup

Examine for safe cases

ScFormulaCellGroup

ScTokenArray

... Tokens

... RPN

getValues
Collected to Matrix

Interpret:
OpenCL Software

Even non-threaded software case: faster
Shares function input collection work.
Aggregated / linearized doubles / strings in the matrix
Why Thread?
CPUs get wider not faster

- Sometimes CPUs get slower ...
- Process clocks stymied at 3-4 GHz
  - IPC improvements ~stalled
- Real IPC wins:
  - Laptops $\rightarrow$ minimum 4 threads
    - Mid-range $\rightarrow$ 8 threads.
  - PC / Workstation
    - 8 $\rightarrow$ 16 threads: the new normal.
- Affordable too ...
- Many thanks to AMD for sponsoring this work.
2017 Crash reporting stats

- Frustratingly ‘cores’ not threads.
Initial Solution ...
Thread InterpretFormulaGroup

- Attempt re-use of existing formula core
  - Try to avoid special / sub-setting code-paths for existing formula-group conversion: a more generic solution.

- Concept:
  - Pre-calculate dependent cells to control recursion outside of threads.
  - Protect invariants with assertions
  - Black-list problematic functions ...
  - Parallelise using existing interpreter.
Parallelize existing interpreter

double ScFormulaCell::GetVal() {
    MaybeInterpret();
    return GetRawValue();
}

void ScFormulaCell::Interpret() {
    ... amazing recursion flattening ...
    InterpretTail() // ie. ...
    {
        ... new ScInterpreter( this, pDocument, rContext, aPos, *pCode /* those tokens */);
        ->Interpret()
    }
    StackVar ScInterpreter::Interpret()
    {
        ... execute reverse-polish stack ...
        ... execute functions ...
        ... get cell values from references ...
    }

    void ScFormulaCell::MaybeInterpret() ...
    assert(!pDocument->mbThreadedGroupCalcInProgress);

    Pre-fetch all dependent values – and lock-that down:

    Pre-calculated →
    No recursion
ScInterpreter: calcs formulae

- **ScDocument**
- **ScTable**
- **ScColumn**
- **ScFormulaCell block**
- **ScFormulaCellGroup**
- **ScTokenArray**
- **Vlookup Cache**
- **ScInterpreter**

**Dependencies**
- ScBroadcastAreaSlotMachine
- Number format, Link mgmt etc.
- Broadcasters
- Mutates: INDEX, OFFSET etc.
- Macros Ext'ns
- Mutates!
- Web fn's
- Cloud

**Tokens**
- ... Tokens
- ... RPN

**Ext'ns**
- Web fn's

**Cloud**
ScInterpreter: some fixes

• Basic iteration - broken:
  • class FormulaTokenArray
    - sal_uInt16 nIndex; // Current step index
    - FormulaToken* FirstRPN() { nIndex = 0; return NextRPN(); }
  • Now has an external iterator
    - a man-week+ to un-wind this, and debug the last pieces that relied on this.

• Added mutation guards:
  • ScMutationGuard aGuard(this, ScMutationGuardFlags::CORE);
    - In all likely-looking places: where core state is changed.
Disabling nasties:

- Dependency graph manipulation
  - During calculation:
    - Indirect, Offset, Match, Cell, ocTableOp
- Other stuff
  - Macros – disabled for now.
    - Could detect ‘pure’ ie. non-mutating functions
    - Also parallelize the basic/ interpreter (?)
  - Info → grab-bag of bits.
  - ocExternal → UNO extensions:
    - currently in: but can do ~un-controlled mutation (?)
More nasties ...

- Several global variables
  - No-where obvious to hang them
  - Now some thread_local variables
    - Calculation stack
    - Current-document being calculated
    - Matrix positions – nC,nR
- Somewhat horrific: fix obsolete Mac toolchain.
- ScInterpreterContext
  - Added – passed through all functions.
    - Impacts eg. ‘GetValue’ though ...
How did that look: initially ...

- Faster
  - Getting some nice speedups – ignoring the hyper-threaded-ness:
    - 8.5s → 2.5 with 4 threads → 3.4x
    - 4.7 → 0.86 - ~5.5x with 8 threads
Up to this point:

- Plain Old calculation – single threaded (POC)
- Group calculation
  - A) Single Threaded Software Group calc (STSG)
  - B) OpenCL: GPU parallelism after conversion
  - C) New threaded calculation (NTC)
- Then: C) slower than A) in some cases ...
  - Collecting data from sheets, branching, type handling, etc. again and again for each formulacell ...
    - Expensive – threading doesn’t help.
    - A) collects once – and has some SSE2 goodness ...
- So → add a ‘threaded A)’ - simple & better ...
- Weighting decision: POC vs. ... based on complexity.
Improving performance ...

- Why don’t we get a 8x for 8 threads?
  - Terrible profiling tools on Windows.
  - Linux – used ‘perf’ looking for threading issues:
    - `sudo perf record --call-graph dwarf \ 
      --switch-events -c 1 # etc.`
- Looking for false-sharing
  - And other horrors.
Horror: rampant heap thrash

- RPN calculation – stack based:
  - Tons of stack operations: pushing values etc.
  - Do memory allocation & frees.
    - Using the ancient / internal allocator – never intended for heavy parallel use.
      → drop the custom allocator → hugely faster
      → Re-use tokens where possible too.
- std::stack → deque → lists ...
  - Horrible: std::vector instead → far better.
- Re-using ScInterpreterContext ...
Other issues ...

- Where ‘GetDouble’ meets SfxItemSet ...
- fixed SvNumberFormatter thread safety.
Threading & optimizing story:

Benchmarking some of our sample sheets ...

Note this perf. Regression from threading for some workloads came from avoiding the SoftwareGroup interpreter.

- Baseline from recent master
- Group Interpreter work by Tor
- Thread Software Interpreter
- Avoid TokenArray thrash
- Halve the number of threads if HT is active
- Use a cache for FormulaDoubleToken allocation
- Disable custom allocator
- Make token cache thread local
- Halve threadcount if HT active for group interpreter too
- Use C++ threads

23 / 25
Future work

- Stop the Crash-testing from asserting ...
  - Implicit intersection: killing us (again)
    - Move RPN to have precise ranges
- Extend threaded unit tests further ...
- Move more global variables to ScInterpreterContext
- Make FormulaCell a 1x item group
  - Make POC calculation a forced-single-threaded calc
    - Always thread SoftwareGroup Interpreter
- De-bong the format-typeuse
  - =J20 – should not change format type if J20 changes format.
    - A sheet-creation-time optimization ...
    - Intersects with ‘units’ work too.
Conclusions

- Calculation can be threaded
- Significant speedups are possible
- Profiling & optimizing works
  - “it is slow” == “not enough invested yet”
    - All problems are just economics
- Many thanks to AMD for their support.

Oh, that my words were recorded, that they were written on a scroll, that they were inscribed with an iron tool on lead, or engraved in rock for ever! I know that my Redeemer lives, and that in the end he will stand upon the earth. And though this body has been destroyed yet in my flesh I will see God, I myself will see him, with my own eyes - I and not another. How my heart yearns within me. - Job 19: 23-27