Making Calc Calculate in Parallel

Tor Lillqvist

Collabora Productivity

@TorLillqvist

Background

Background

Number of cores in CPUs is increasing

Relatively soon 8 cores will be commonplace

Performance per core is not increasing so much

Background

Calc so far single-threaded

- Performance will not improve much no matter how many cores the machine has

OpenCL was supposed to be the solution

Typically runs on GPU, but can also run on CPU

For various reasons using OpenCL in LibreOffice did not work out as as nicely as expected

The OpenCL-generating code is hideously complicated

Very few developers even capable to work on it because of hardware/software issues

Background

Formula groups

- **Introduced as part of the OpenCL work some years ago**
- **Used when multiply sequential formulas in a column are “identical”: cell references are either absolute or to cells at an identical row and column offset**
- **For example:**
- **B1: =SUM(A$1:1)/D$1 + C1**
- **B2: =SUM(A$1:2)/D$1 + C2**
- **B3: =SUM(A$1:3)/D$1 + C3**

Only done vertically. That is how repeated formulas occur in practice.

Background

Each formula group is calculated as a whole, using either OpenCL or the “software interpreter”

Input for those two calculation methods is collected into a packed vector of values, and output is stored in a such during computation. Afterwards the output is stored into the formula group’s cells.

Future

Plans

Instead of OpenCL, threading of Calc should thus be done using plain C++ code

Lots of challenges with that

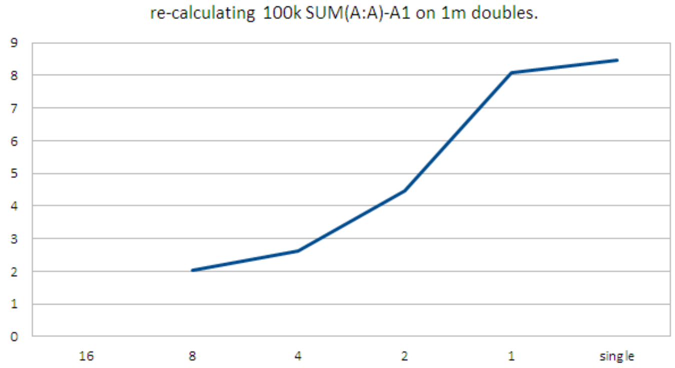
Multi-threading aspects have not really been a concern when the Calc code has been written

Data structures sub-optimal for multi-threaded use

What done

Approach to be taken: Find the right place where to start threads, and just do it. Check what breaks. Fix. Iterate.

Initial work done
Results fairly promising
For trivial but large sheets speedup in the order of number of threads

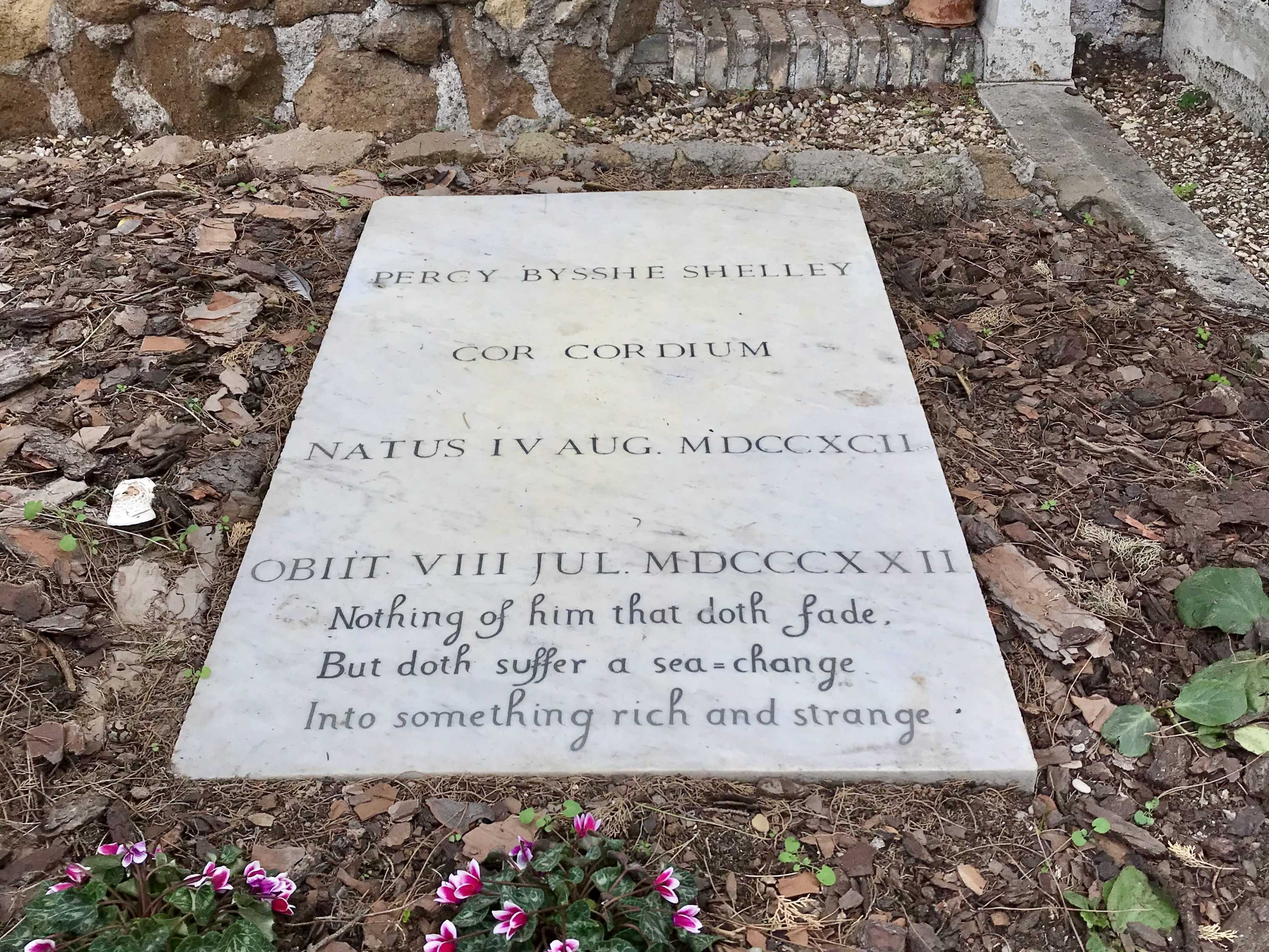


Collabora Productivity

Future

Eventually OpenCL could be retired

Optionality of “software interpreter” should really go away. The less options the better. Use it automatically when it makes sense.



PERCY BYSSHE SHELLEY

COR CORDIUM

NATUS IV AUG. MDCCXCII.

OBIIT VIII JUL. MDCCCXXII

Nothing of him that doth fade,
But doth suffer a sea-change
Into something rich and strange.

Implementation

Implementation plan

Add a fourth code path for formula cell calculation

Existing:
Plain traditional single-threaded, one formula cell at a time
Formula group with “software interpreter”
Formula group with OpenCL

New:
Formula group in parallel

Implementation questions

When to use the parallel calculation?
When OpenCL is not available?
Also when there is OpenCL, but the formula is not eligible for OpenCL?
Should the “software interpreter” be preferred when eligible?

Implementation

Basic steps, examples:
Make a few random static local variables thread-local

```
        case ScMatrixMode::Reference :
        {
-           static SCCOL nC;
-           static SCROW nR;
+            static thread_local SCCOL nC;
+            static thread_local SCROW nR;
            ScAddress aOrg;
            if ( !GetMatrixOrigin( aOrg ) )
               return sc::MatrixEdge::Nothing;
```

Implementation

Basic steps, examples:
Make a static local variable thread-local, or otherwise make the function multi-thread safe
We used to have:

```
bool ScTable::ValidQuery(
    SCROW nRow, const ScQueryParam& rParam, ScRefCellValue* pCell, bool* pbTestEqualCondition)
{
    SCSIZE nEntryCount = rParam.GetEntryCount();

    typedef std::pair<bool,bool> ResultType;
    static std::vector<ResultType> aResults;
    if (aResults.size() < nEntryCount)
        aResults.resize(nEntryCount);
...
```

Just revert this optimisation

Implementation

Basic steps, more:
Move iterator index of FormulaTokenArray out of the class into separate class

```
class FORMULA_DLLPUBLIC FormulaTokenArray
{
...
    FormulaToken**  pCode;                  // Token code array
    FormulaToken**  pRPN;                   // RPN array
    sal_uInt16      nLen;                   // Length of token array
    sal_uInt16      nRPN;                   // Length of RPN array
    sal_uInt16      nIndex;                 // Current step index
    FormulaError    nError;                 // Error code
```

Instead added a separate iterator class

Implementation

Run threads in ScFormulaCell::
InterpretFormulaGroup()
Split work into as equal pieces as possible
Use same minimum formula group size as for OpenCL.
Except that we now use “weight,” not just size. Also
number of input cells taken into account.

Implementation

Before running threads, calculate values of cells referenced by the formula where necessary, to avoid threaded recursive interpretation

Make sure through assertions that when doing threaded calculation, shared data structures are not mutated.

For example, don’t manipulate the formula “tree” (actually a list) while in threads

Implementation

A Calc document is represented by a ScDocument

It also holds much stuff that is related to formula interpretation

This is obviously a problem when running multiple interpreters (ScInterpreter) in parallel
Move those fields into a new struct, ScInterpreterContext
Allocate a such for each interpreter thread, pass around to functions that need it

Implementation

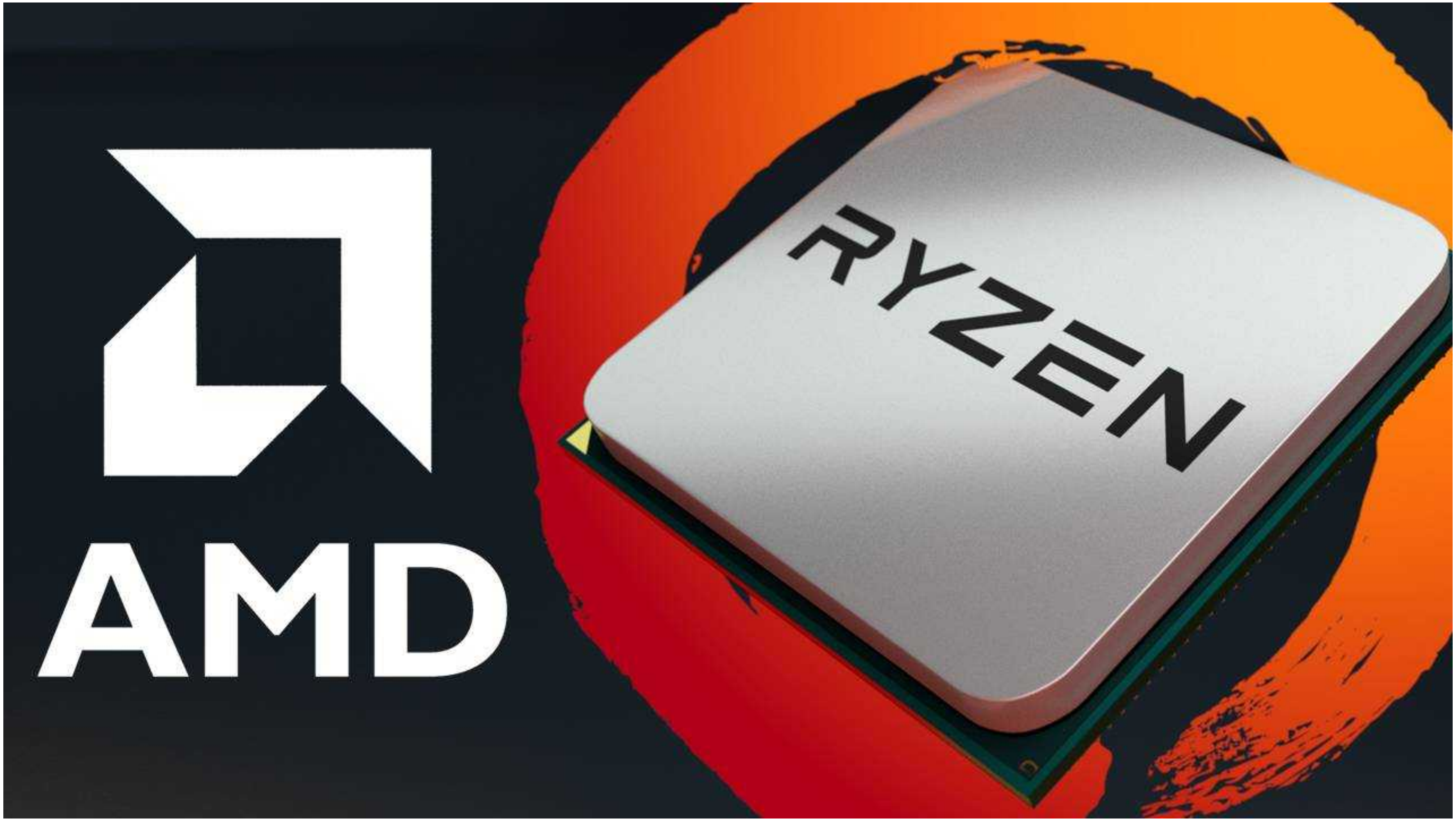
So far in experimentation it has worked surprisingly well

Simple cases indeed speed up as expected

But in some cases not that much

Tweaks needed

Thanks to AMD for funding this work



ROMA
1927
TI AMO
RA·16
0316
Enysh-Mask

Thank you

Tor Lillqvist

@TorLillqvist

tml@collabora.com