Exploiting Concurrency

How I stopped worrying and started threading

Michael Meeks
michael.meeks@collabora.com
mmeeks / irc.freenode.net

Collabora Productivity

“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
The exciting (insurmountable?) opportunity

- Processor clocks are stymied at 3-4GHz
  - Various clever tricks improve IPC still
- But the real IPC wins:
  - Those other cores – 16 threads: the new normal
  - If we use a single thread: we waste 90+%
So where are we at?
LibreOffice Threading Status

• We use lots of threads
  • Start impress: 6 threads.
    - VCL / main – thread
    - custom allocator thread: rtl_cache
    - factory thread → accept on argument pipe.
    - update-check thread → in a sleep.
    - gio thread → ? pulseaudio thread → ?
  • Windows: worse → GDI + real-time timer threads
• Approximately 1+epsilon – zero concurrency
Its not all bad ...

• We have some threads that make sense
  • eg. writing configuration thread (short lived)
• Thread-pool for larger tasks
  • Large high-fidelity image scaling:
    – we parallelize nicely: easy to partition task.
• Zipping → ODF & DOCX
  – Deflation → expensive → implemented this.
  – Turns out the big problem: images → solved.
• XML parsing / tokenize / sheet loading.
• Drawing Layer / 3D / Software rendering fallback
Why thread?

• Benefits:
  • Focus limited CPU resources; Limited !?
    - Branch predictor complexity
    - L1 data cache size
    - memory bandwidth
    - Instruction decode logic & micro-op caches
  • Splitting your work into a series of passes:
    - Each going in a thread → maximizes resource use
      • eg. XML parsing.
    - Safe message passing along the pipeline …
  • Technology change → provides opportunity …
  • Plenty of problems to match …
SAX / parsing etc.

- Arbitrary size XML SAX parsing in constant time*
  * - assuming you have a spare CPU and are slower to consume the events
  also tokenizing (to reduce buffer sizes etc.)
- Threaded OOXML import from ages back – 5.0+ ?
  - Calc – threading concurrent core of sheet importing
    - occasional data loss → recently fixed by Kohei
      - Unhelpful race in package/ code.
- ODF formats from 5.4+
  - Switch to XFastParser
    - incremental work by Mohammed Abdul Azeem
    - Plenty of good work here to significantly accelerate import
Where can we win?
What next? XML ...

- XML parsing still can be improved
  - Splitting I/O & unzip → another 15%
  - Unfortunately XML parse / tokenize -the- bottleneck:
    - Faster to interpret data, parse doubles, populate spreadsheet – vs. XML parsing, namespace handle, tokenize.
- Fundamentally parsing XML is slow.
  - We get big files ...
- 256k rows of double, string → 122Mb of XML.
- Custom – look ahead, threaded XML parser?
  - 1 million rows → needs data feeding.
What next? Images ...

- Handling images is slow
  - Text / strings are small.
  - Images are -big- and getting worse: many Mb.
  - 100 pages of text → 5Mb, 1 image → 5Mb

- Image handling is a mess:
  - But – have nice opaque image handles
  - Can have instant image load:
    - Queue threaded loading.
    - re-serialize when image actually used ...
    - Also stop loading & de-compressing images to measure their pixel size & then discard again ...

- Nice, isolated change.
What next? VCL / Windows

- LibreOffice:
  - thread-safe widget / scripting API.
- Windows:
  - Thread affine Window event queue, GDI resource allocation etc. < set of ~10 things>
  - Horrors to push events across threads
    - Into contended osl conditions (and Mutex?)
    - Un-controlled re-enterancy nightmare
    - Frequent cause of bugs.
  - Have a GDI / Window handling thread …
    - Always responsive and running: have the IPC already.
What next? VCL / render thread

- Partition: split calculation & layout from render threads.
- Meta-files:
  - Re-factor & split out back-compat / file pieces
  - Extend to ~full VCL API set.
  - Stream the meta-file to a render-thread:
    - needs immutable / safe VCL / tools primitives: bitmaps, polygons etc.
- Render meta-files separately ...
  - In that Windows Window / GDI thread?
  - Reduce OpenGL / acceleration threading & context woes too.
- Always responsive progress-bar thread too?
  - Remove another un-controlled re-entrancy point ... ‘Yield’
What next? Little bits?

- Specific simpler pieces?
  - OpenMP - able?
  - Rust?
    - Hmm → its a big problem.
    - Adding annotations? Already have ~100 clang plugins.
      - Good C++ extensions will come surely !?
    - Current C++ extensions – functionality not validation [!]
- Word-count:
  - currently in small chunks ...
  - Little chunks of time → do 16 times more? ...
- Speculative shaping & text measuring (?)
What next? Calc Core …

- Lots of parallelism – but only with OpenCL
  - No threading in the non-CL case
    - Much less tested → bad …
  - Have rich CPU – available to everyone
  - Dependency graph pieces → badly needed
    - avoid the horrors of ‘calc-chain’ nonsense.
- Design, validation, dependency, testing vital
- Pivot-Tables
  - Lots of fun possible here …
Summary

• Threading: its scary but necessary
• We need new ways to make it safer in C++
• Big opportunities to speed up slow things.
• By thinking about, and pipelining algorithms:
  • Make them smaller, easier to read & understand, and sometimes more efficient – even if not threaded.
• Questions?

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