ITDevCon

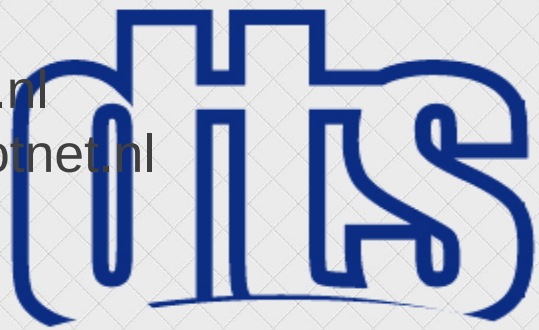
European Delphi Conference

14, 15 november 2013 - VERONA (Italy)

Profiling techiques for Delphi

André Mussche, The Netherlands
andre.mussche@gmail.com

dts.nl
4dotnet.nl



Contents

- Introduction
- Using Delphi IDE
- Sampling profiling
- Instrumenting profiling
- Optimization tips
- Other tools
- Intel vTune

Introduction

“Premature optimization is the root of all evil”
So you should not do any optimization at all when you start programming?
I don’t like black or white, truth is somewhere in between:

- Know what you are doing (no cowboy programming)
- Quickly estimate how fast it should be:
  - server side function that is used a lot?
  - button click on about box that is rarely used?
- Use default Delphi structures (string, TDictionary, etc)
  - can be changed easily (Agile)
- But don’t fully optimize it already (no pointers, assembly, etc => no easy porting to 64bit, Mobile/ARM, etc) unless you are making e.g. a cryptographic function

Introduction (2)

- Think about architecture
  - cannot be changed easily afterwards
  - no “chatty” interfaces, but more in one call
  - abstraction has a price (more overhead)
    - KISS: Keep It Simple
    - Don’t make it technically perfect
- Think about quality metrics
  - Speed
  - Maintainability
  - Scaleability
  - Flexibility/abstraction
  - Readability

=> you can’t have them all!

Introduction (3)

- When it gets slow…
  - Don’t be afraid: sooner or later it will (changes over time, different usage, etc -> don’t know the future)
- Measure before restructure! Don’t only look at code...
  - mostly in least expected place
  - mostly small changes with big effect
- Examples:
  - SAP sync was slow (>8h) => optimize sql?
  - No: connect+fetch single field+disconnect => single connect, no disconnect => 45min!
  - Slow import => too many objects? No: slow Excel OLE => TMS Flexcel (native file load) => 4x faster
  - Still too slow => too many single sql’s? => No: background debug form with memo =>3x faster

Introduction (4)

- Too lazy programming:
  - Load order object (BO) with all orderlines and details, only to get ordernumber in logging…
  - => selective load (array of ORM fields) + lazy load of details
  - => make special (class) function to get one field

Delphi IDE

- Just pause the debugger
  - slow functions have more chance to get paused
  - (so pause a couple of times)
  - Watch the stack (mainthread)
- Technique learned from Andreas Hausladen
- (DelphiSpeedUp, IDE Fix pack, DDevExtensions, etc)
- Tip: use the F12 key (Windows XP)
  - Vista/7/8+: Delphi F12 Debug Hotkey support
  - Direct pause when app hangs (e.g. button click)

Sampling profiling

- Seperate tool
- Takes up to 1000 stack traces per second
- No interruption/modification of application
- Shows all used functions, also dll's
- (drawing, database, etc)
  - only when catched in stack trace (not every tiny function)
- Estimation of function durations
- Common data shown
  - function name (unit, class)
  - Hit or sample count
  - Duration percentage
    - .own time: e.g. calculations in function itself
    - .child time: calls to other functions

Sampling profiling: tools

- Sampling Profiler
  - Free
  - Small, easy to use, no installation (xcopy)
  - Online processing (webserver), only shows top function
- AQTime
  - Commercial
  - Needs installation (not suited for production server)
- AsmProfiler (sampling mode)
  - Open source
  - No installation
  - Detailed: every stack frame can be seen, time per thread
    - but sometimes partial stack tracing, no 100% correct results
  - Less easier to use?

Sampling profiling: requirements

- Debug symbols
  - everything supported by jclDebug.pas
    - detailed .map & .jdbg files
    - JCLDEBUG & TD32 sections
    - dll exports, .pdb files
- Compiler settings
  - stack frames
- Tip: include debug symbols in .exe
  - MakeJclDbg.exe (JCL)
  - For example, when making build with FinalBuilder
  - Always correct debug symbols (otherwise 2 files to deploy: project.exe and project.map)

Sampling profiling: demo

Demo

Instrumenting profiling

- Source profilers
  - modifies all your source code!
  - optimizations by compiler?
  - Also 64bit and ARM possible, MacOS(?)
  - http://www.prodelphi.de/ (Freeware + Pro versions)
- Binary profilers
  - Modifies binary code
    - .on-disk or only runtime (in-memory)
  - CPU specific
  - small functions (<5bytes) can't be changed
    - .no room for assembly JMP <address>
  - Tools:
    - .AQTime
    - .AsmProfiler (instrumenting mode)

Instrumenting profiling (2)

- Cons
  - Only specific (marked) functions
  - Real execution is changed
    - Little overhead per call
      - Be aware: many small functions gives false/unbalanced tree (high child execution times)
    - CPU is interrupted, less optimized (cpu cached) execution
- Pros
  - Exact function duration times
  - Number of executions (call count)
  - Trace tree is possible

Instrumenting profiling: AsmProfiler

- AsmProfiler.dll:
  - Binary profiler, using assembly for detouring functions and modifying stack (to get start and end times)
  - Stores every call, not only average (so high memory usage!)
    - Can make exact call trace
    - Duration for each call (for trends)
  - Only 32bit
  - Usage:
    - API to load and start/stop in your code
    - Dll injection
  - Less user friendly?

Instrumenting profiling: approach

- Approach:
  - First use sampling profiler to get quick view
    - .export found functions for instrumenting
  - Look at highest durations (incl child times)
    - .90% function / 2x faster = 45% improvement
    - .10% function / 10x faster = only 9% improvement
  - Look at call count: function called too much?
    - .e.g. 3x loading same data
  - Iterate till no more possible (without big changes)
  - Re-architecture (less abstraction, specialized single direct but fast function, etc)

Instrumenting profiling: Demo

Demo

Optimization tips

- Tips:
  - const strings/arrays/variants/interface parameters
    - see CPU Window!
  - begin+endupdate, disable+enablecontrols
  - class functions (no object creation)
  - inline for small functions
  - no variants, use simple types as much as possible
  - use SQL params (no query compile each time because no unique sql's anymore)
  - direct low level ADO instead of TAdoQuery for single rows/values (20x faster!)
  - test with real (lots of) data

Optimization tips (2)

- Tips (continued):
  - use multi threading
  - short (finegrained) locks (no processing within lock but copy data)
  - no datamodules in threads due to global RTL locks
  - use a scaling memory manager (ScaleMM2 or Google's TCmalloc) for memory intensive multithreaded applications
  - data alignment: most used data in front, fixed arrays last (in records)
  - no "packed" records/arrays or use fillers for 4/8/16 byte alignment

Other tools

- Live stack view:
  - Process Explorer (threads tab in detail popup)
    .needs tds2pdb
  - Stack viewer of AsmProfiler Sampler

- Minidump reader
  - When e.g. mainthread (watchdog thread) -> make minidump
  - Or with Process Explorer or Win7 Task manager -> make minidump
  - Offline stack viewer of each thread

Intel vTune

- Intel vTune Amplifier XE
  - CPU/GPU times
  - Threading, locks & waits
  - Intel hardware Performance Monitoring Unit (PMU)
    - .CPU instructions retired/executed/stalled, L2 cache hits and misses, branch misprediction, partial address alias, split load/store, data alignment, etc
    - .Note: CPU waits when result is fetched! (async)
  - Report with tuning advise/help
  - Big and slow...
  - Delphi not supported
    - .mixed results with tds2pdb (but no source line support)

Intel vTune: Demo

Demo?