Tips and Tricks to Speed LabVIEW Performance

Robert Berger

Sr. District Sales Manager

Agenda

- Recap of favorite tips and tricks from years past
- Benchmarking techniques
- Programming techniques
- Algorithm selection

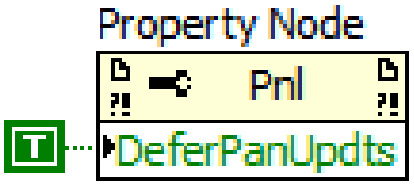
- New LabVIEW 2011 usability features

“Best of” Recap

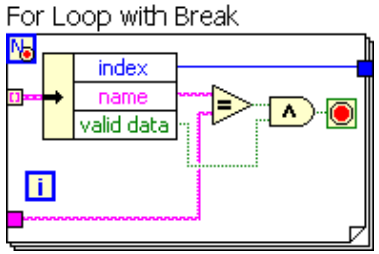
A rundown of some favorites from my Performance Tips and Tricks presentations at past NIWeek conferences

“Best of” Recap

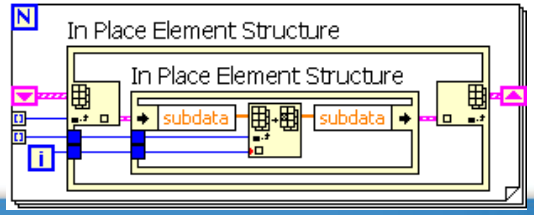
#1 – Defer Panel Updates



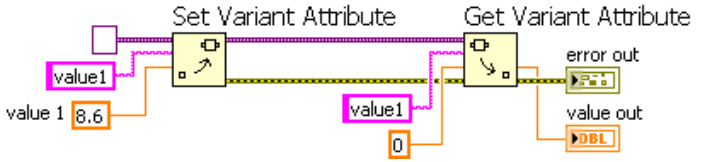
#2 – For Loop with Break



#3 – In Place Element Structure

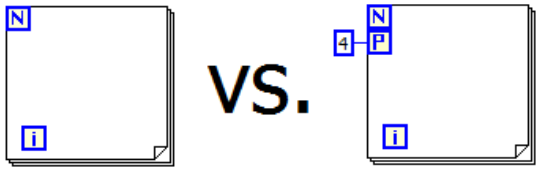


#4 – Variant Attributes



#5 – Build Array Ordering

#6 – Parallel For Loop

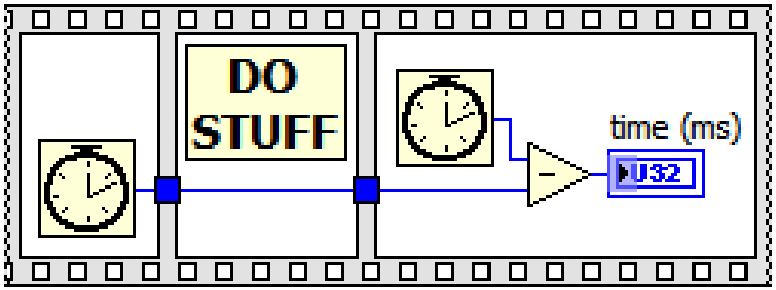


Benchmarking Techniques

How do we figure out which programming techniques are faster?

Benchmarking Techniques

- Good



- Better

High Resolution Relative Seconds.vi

(located in vi.lib\Utility)

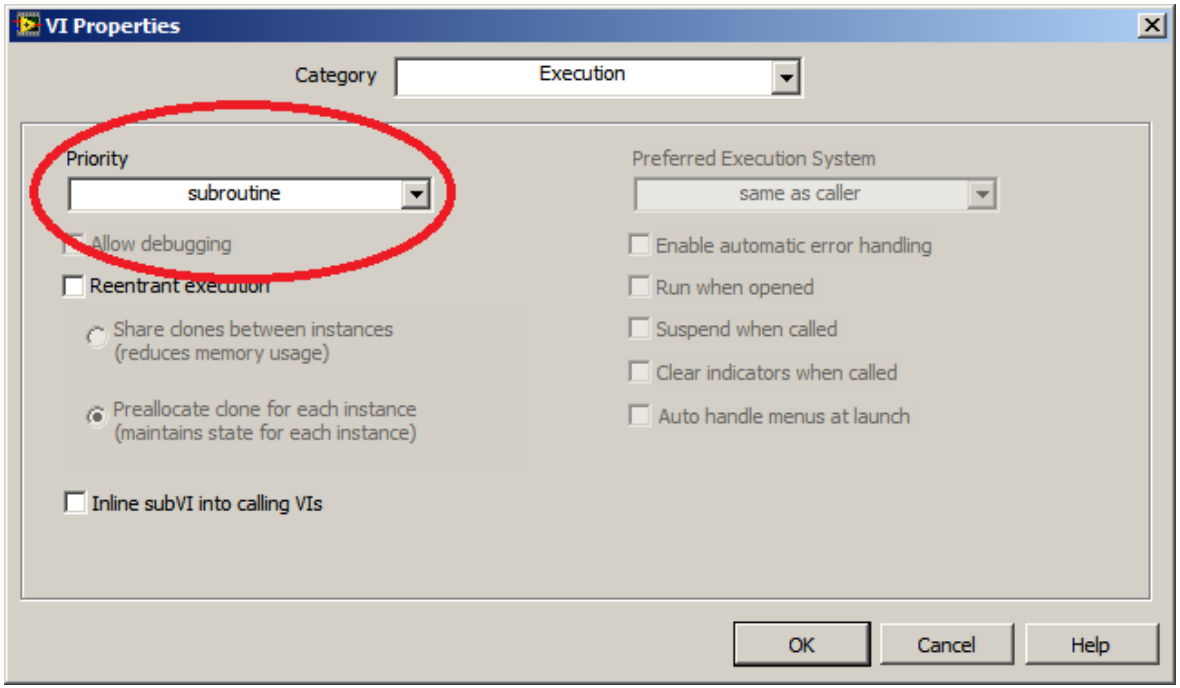
- Best

??? – Google '*timing probe idea*' to help us figure it out!

Programming Techniques

Three specific suggestions on how to improve the performance of your VIs without having to drop a single node or wire!

#1. Subroutine Priority



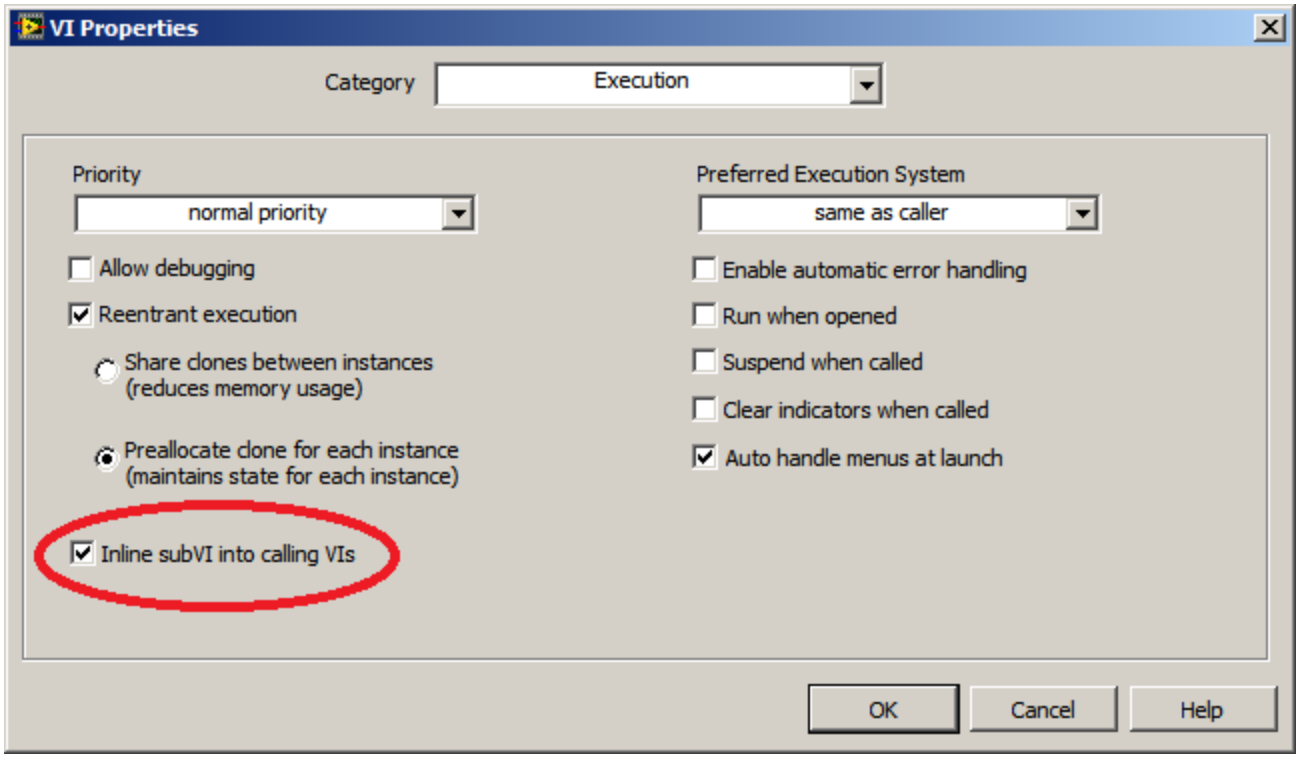
The **subroutine** priority setting on a VI causes that VI to take control of the thread in which it is executing. This allows it to run as efficiently as possible.

#1. Subroutine Priority

Subroutine Caveats

- A subroutine VI can only call other subroutine VIs
- A subroutine VI cannot call any blocking functions (Wait, One Button Dialog, VISA calls, etc.)
- Front Panel controls and indicators are not updated during execution
- No other VI in the calling VI's thread can run while a subroutine VI is running
- DEMO!!!

#2. Inlining SubVIs



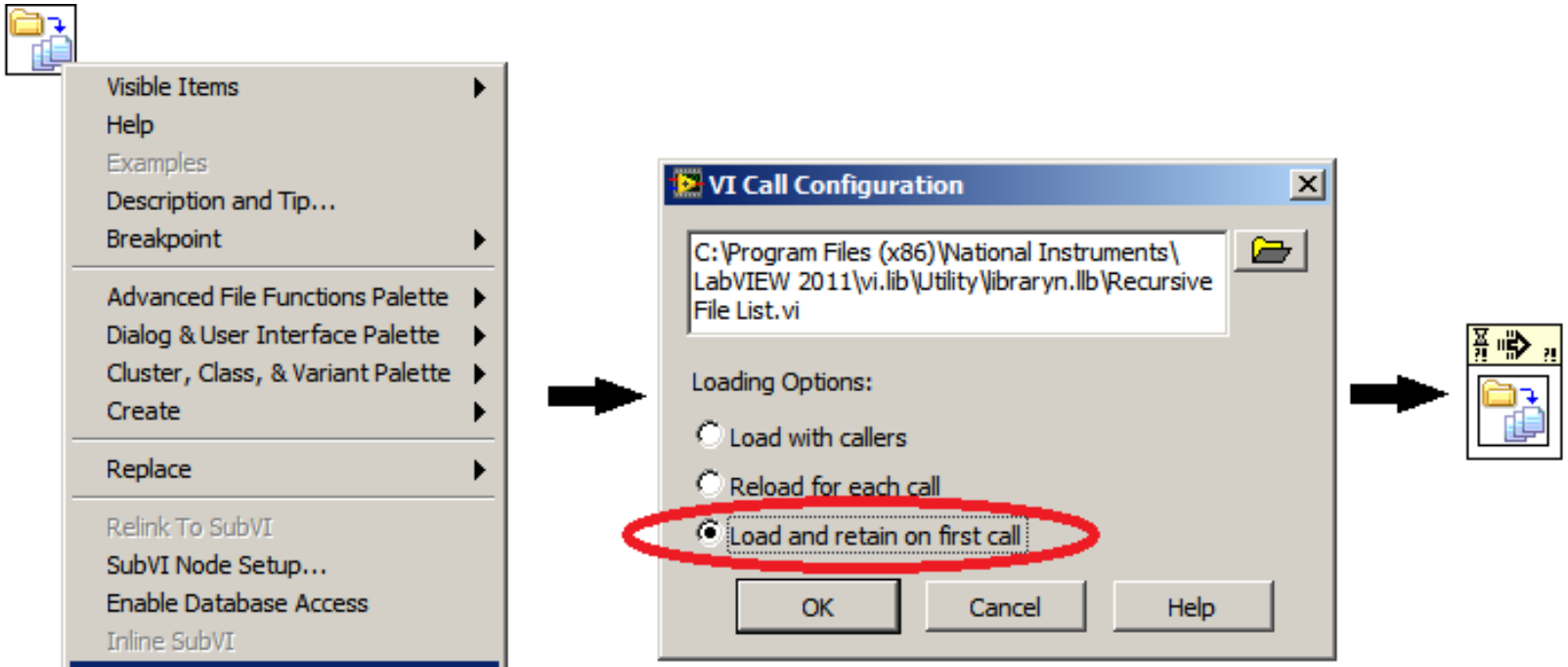
Introduced in LabVIEW 2010, **subVI inlining** eliminates the overhead of calling subVIs by telling the compiler to act as if the subVI code resides directly on the owning diagram.

#2. Inlining SubVIs

Inlining Caveats

- The inlined VI must be reentrant, meaning it cannot hold state information
- You cannot debug inlined VIs
- Inlining may decrease performance on large VIs
- Inlined VIs cannot contain recursive calls
- Inlined VIs cannot contain Property Nodes or Invoke Nodes
- DEMO!!!

#3. Easy Dynamic Calls



The “Call Setup” feature (introduced in LabVIEW 8.0) makes it very easy to change a static subVI call into a dynamic call to improve load time performance

#3. Easy Dynamic Calls

Dynamic Call Caveats

- If the calling VI is in edit mode, all dynamic VIs will be in memory
- The “VI Call Configuration” dialog displays an absolute path, but the calling VI stores a relative path
- “Reload for each call” should only be used if you need to release the memory allocated for each subVI call
- DEMO!!

Algorithm Selection

There are multiple ways to write this VI…can you figure out the *fastest* solution?

Jumble Solver

I have a list of Jumble words, and an open-source dictionary. I want to write a VI that will solve the Jumbles for me.

Jumble Example:

VABWIEL → LABVIEW

New LabVIEW 2011 Usability Features

Edit>Create SubVI Improvements

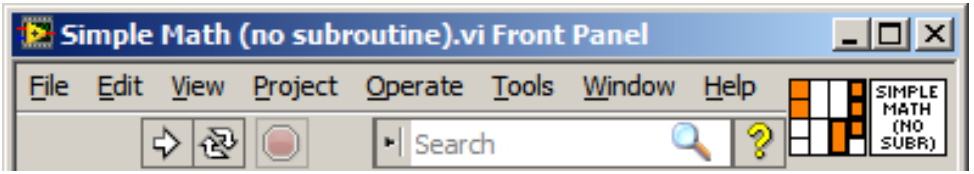
- **Changes to created VI**
  - 4x2x2x4 connector pane (or another default pattern that you specify)
  - Error terminals in lower corners (and named properly)
  - Refnum/class terminals in upper corners (and named properly)
  - Clean front panel
- **Plugin Architecture through LabVIEW Scripting**
  - If you like the way we create the subVI, but you want to do something extra, you can write a plugin VI that will perform further modifications on the subVI
  - If you don't like the way we create the subVI, you can completely replace our scripting code with your own

New LabVIEW 2011 Usability Features

Quick Drop Launch Time

- **Quick Drop is now instantly usable on first launch**
  - …provided you don't try to use it immediately as soon as you launch LabVIEW

Connector Pane Always Visible



Boolean Functions Accept Error Clusters