



Quick! Drop Your VI Execution Time!

General-purpose techniques to speed up your VIs

Darren Nattinger

Chief Technical Support Engineer, CLA

NI

Before we get started

All of my presentations (including this one) are available at:



dnatt.org

(slides, demos, and links to video recordings)

This presentation's link: <https://bit.ly/slowvis>

[Download link for ZoomIt](#)

Outline

- Why I'm talking about this stuff
- Stuff I'm not going to talk about
- Stuff I'm going to talk about
- **Real-world** demos that show the stuff I talked about

Why am I giving this presentation?

- About once a month, somebody comes to me with a slow VI and asks me to make it run faster.
- These slow VIs reside in a wide variety of LabVIEW applications.
 - ...but are usually of the type “do something with a big chunk of data”.
- Over the years I have accumulated a toolbox of **simple, general-purpose** techniques for improving VI execution time.
- I am sharing those techniques with you today.

Stuff I'm not going to talk about

- Desktop Execution Trace Toolkit
- Show Buffer Allocations
- Profile Buffer Allocations
 - The coolest LabVIEW feature you've never heard of
- Benchmarking techniques
 - <http://bit.ly/brainlesslabview>
- How the LabVIEW compiler works
 - [Introduction to the LabVIEW Compiler](#)
 - [LabVIEW Compiler Under the Hood](#)
- Real-Time/FPGA

Stuff I'm going to talk about

- **VI Profiler**
 - The good, the bad, and the ugly
- VI Settings
 - Enabled debugging, Priority, Inlining, etc.
- Parallel For Loops
- Programming Patterns for Performance
- Sets and Maps

- Illustrative **real-world** demos

Disclaimer

- There are times when we have to do **silly things** to eke out more performance from our VIs.
- If code readability and maintainability is our #1 goal, we shouldn't do these things.
- If code performance is our #1 goal, we may have to.
- Items marked as “!” in this presentation denote these situations.
- *“Make it work, make it right, make it fast.” – Kent Beck*
...then make sure it still works.

VI Profiler

VI Profiler

- Official name: “Profile Performance and Memory”
- *Tools > Profile > Performance and Memory*
- Has been around forever
- Gives information on execution time of VIs, along with optional info on memory usage

The screenshot shows the 'Profile Performance and Memory' window for a project named 'Board Testing - Benefits of Object-Oriented Design.lvproj'. The interface includes several configuration options and a table of profile data.

Configuration Options:

- Timing statistics
- Profile memory usage
- Timing details
- Memory usage
- Time unit: milliseconds
- Size unit: kilobytes
- Application Instances: My Computer

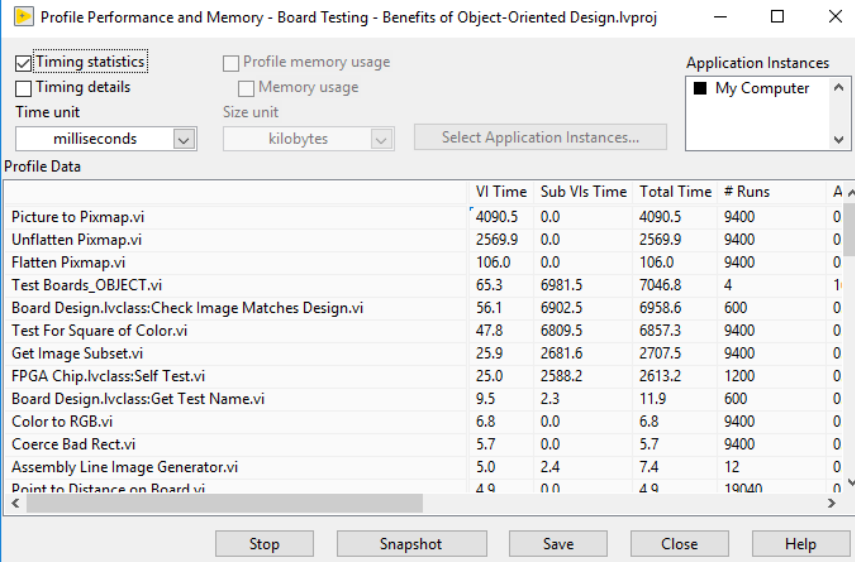
Profile Data Table:

	VI Time	Sub VIs Time	Total Time	Project Library	A
Picture to Pixmap.vi	4090.5	0.0	4090.5		M
Unflatten Pixmap.vi	2569.9	0.0	2569.9		M
Flatten Pixmap.vi	106.0	0.0	106.0		M
Test Boards_OBJECT.vi	65.3	6981.5	7046.8		M
Board Design.lvclass:Check Image Matches Design.vi	56.1	6902.5	6958.6	Board Design.h	M
Test For Square of Color.vi	47.8	6809.5	6857.3		M
Get Image Subset.vi	25.9	2681.6	2707.5		M
FPGA Chip.lvclass:Self Test.vi	25.0	2588.2	2613.2	FPGA Chip.lvcl	M
Board Design.lvclass:Get Test Name.vi	9.5	2.3	11.9	Board Design.h	M
Color to RGB.vi	6.8	0.0	6.8		M
Coerce Bad Rect.vi	5.7	0.0	5.7		M
Assembly Line Image Generator.vi	5.0	2.4	7.4		M
Print to Distance on Board.vi	4.9	0.0	4.9		M

Buttons: Stop, Snapshot, Save, Close, Help

VI Profiler – Simple Usage Procedure

1. Launch *Tools > Profile > Performance and Memory*
2. Check 'Timing statistics'
3. Click *Start*
4. Run your code
5. Click *Snapshot*
6. Interpret Results



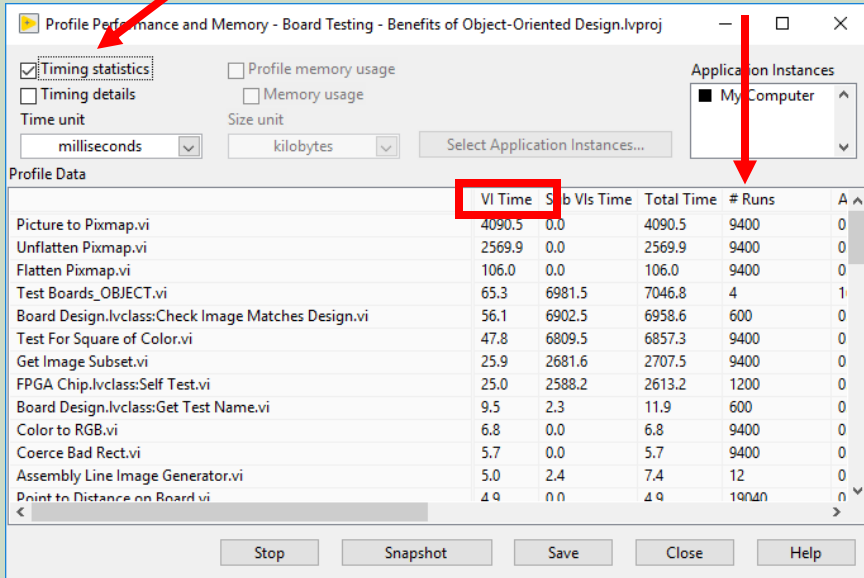
The screenshot shows the 'Profile Performance and Memory' window for a LabVIEW project. The 'Timing statistics' checkbox is checked, and the 'Timing details' checkbox is unchecked. The 'Time unit' is set to 'milliseconds' and the 'Size unit' is set to 'kilobytes'. The 'Application Instances' list shows 'My Computer'. The 'Profile Data' table is displayed below.

	VI Time	Sub VIs Time	Total Time	# Runs	A
Picture to Pixmap.vi	4090.5	0.0	4090.5	9400	0
Unflatten Pixmap.vi	2569.9	0.0	2569.9	9400	0
Flatten Pixmap.vi	106.0	0.0	106.0	9400	0
Test Boards_OBJECT.vi	65.3	6981.5	7046.8	4	1
Board Design.lvclass:Check Image Matches Design.vi	56.1	6902.5	6958.6	600	0
Test For Square of Color.vi	47.8	6809.5	6857.3	9400	0
Get Image Subset.vi	25.9	2681.6	2707.5	9400	0
FPGA Chip.lvclass:Self Test.vi	25.0	2588.2	2613.2	1200	0
Board Design.lvclass:Get Test Name.vi	9.5	2.3	11.9	600	0
Color to RGB.vi	6.8	0.0	6.8	9400	0
Coerce Bad Rect.vi	5.7	0.0	5.7	9400	0
Assembly Line Image Generator.vi	5.0	2.4	7.4	12	0
Print to Distance on Board.vi	4.9	0.0	4.9	19400	0

Buttons at the bottom: Stop, Snapshot, Save, Close, Help.

VI Profiler – The Good

- **Very low barrier to entry**
- **Very easy to interpret results**
- Automatically sorts by VI Time
 - Sortable columns (but VI Time is almost always what I want to sort by)
- Enabling “Timing statistics” shows the “# Runs” column
 - Useful when deciding if inlining makes sense



Profile Performance and Memory - Board Testing - Benefits of Object-Oriented Design.lvproj

Timing statistics Profile memory usage
 Timing details Memory usage

Time unit: milliseconds Size unit: kilobytes Select Application Instances...

Application Instances: My Computer

Profile Data

	VI Time	Sub VIs Time	Total Time	# Runs	A
Picture to Pixmap.vi	4090.5	0.0	4090.5	9400	0
Unflatten Pixmap.vi	2569.9	0.0	2569.9	9400	0
Flatten Pixmap.vi	106.0	0.0	106.0	9400	0
Test Boards_OBJECT.vi	65.3	6981.5	7046.8	4	1
Board Design.lvclass:Check Image Matches Design.vi	56.1	6902.5	6958.6	600	0
Test For Square of Color.vi	47.8	6809.5	6857.3	9400	0
Get Image Subset.vi	25.9	2681.6	2707.5	9400	0
FPGA Chip.lvclass:Self Test.vi	25.0	2588.2	2613.2	1200	0
Board Design.lvclass:Get Test Name.vi	9.5	2.3	11.9	600	0
Color to RGB.vi	6.8	0.0	6.8	9400	0
Coerce Bad Rect.vi	5.7	0.0	5.7	9400	0
Assembly Line Image Generator.vi	5.0	2.4	7.4	12	0
Print to Distance on Board.vi	4.9	0.0	4.9	10000	0

Stop Snapshot Save Close Help

VI Profiler – The Bad

- C-based feature
 - No LabVIEW-based extensions ☹️
- Inline VIs do not show up
- Lots of mostly distracting info

Profile Performance and Memory - Board Testing - Benefits of Object-Oriented Design.lvproj

Timing statistics

Profile Data

	VI Time	# Runs
Picture to Pixmap.vi	4090.5	9400
Unflatten Pixmap.vi	2569.9	9400
Flatten Pixmap.vi	106.0	9400
Test Boards_OBJECT.vi	65.3	4
Board Design.lvclass:Check Image Matches Design.vi	56.1	600
Test For Square of Color.vi	47.8	9400
Get Image Subset.vi	25.9	9400
FPGA Chip.lvclass:Self Test.vi	25.0	1200
Board Design.lvclass:Get Test Name.vi	9.5	600
Color to RGB.vi	6.8	9400
Coerce Bad Rect.vi	5.7	9400
Assembly Line Image Generator.vi	5.0	12
Print to Distance on Board.vi	4.0	10000

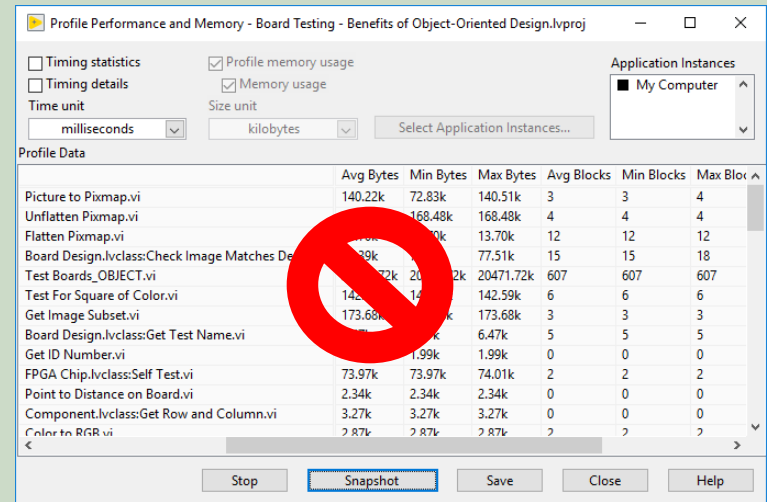
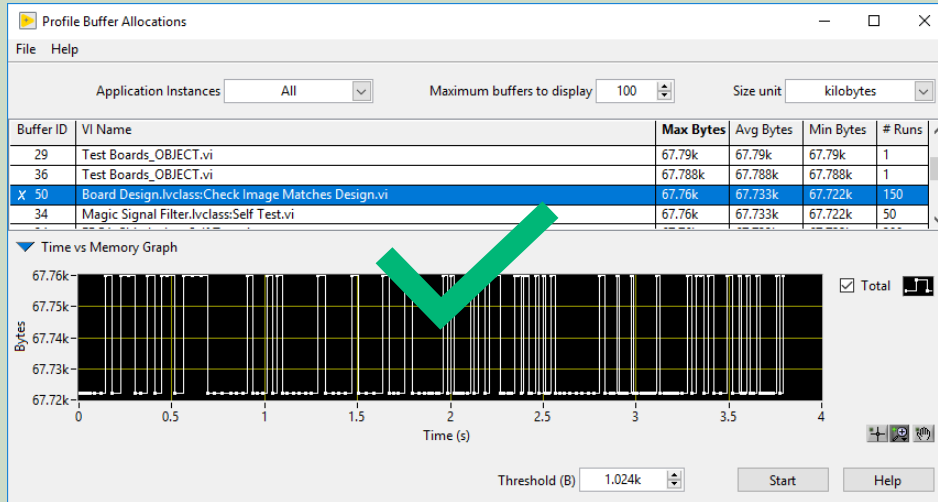
Stop Snapshot Save Close Help

VI Profiler – The Ugly

- Absolute time values are often **unexpected**
 - A VI that takes 10 seconds to run might show ‘VI Time’ values that sum to something completely different
 - One reason is that parallel operations are summed
 - A VI with two parallel loops that run within 1 second will show a profile time of 2 seconds
 - Another reason is because “LabVIEW-friendly sleep time” is not included
 - LabVIEW-friendly sleep: Wait functions, Event Structure, TCP, Queues
 - LabVIEW un-friendly sleep: OS-level (e.g. driver functions, DLL calls)
- Use VI Time value as a **relative metric**
 - Focus on the big numbers
 - Ignore the small numbers
 - You’re making progress if the big numbers get smaller and your VI execution time decreases

VI Profiler – What about Memory?

- The VI Profiler gives memory usage info on a **per-VI** basis
- **Profile Buffer Allocations** gives memory usage info on a **per-node** basis
 - (most of the time)



VI Profiler – More Granular Information

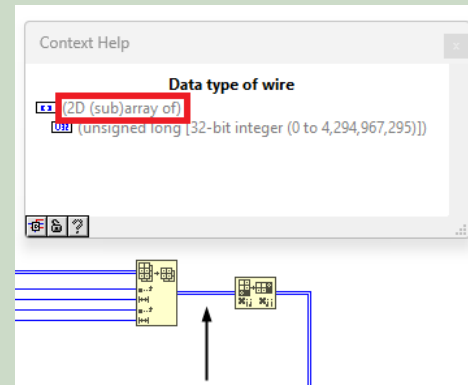
- Use **Edit > Create SubVI** to create temporary subVIs of suspect code (!)
 - Workaround for the lack of per-node execution time
- These subVIs will appear in the VI Profiler to help you narrow down issues

Profile Data				
	VI Time	Sub VIs Time	Total Time	# Runs
Waveform Time to Date Time String.vi	8014.5	0.0	8014.5	2500000
WriteToCSV.vi	5719.9	8014.5	13734.4	1

VS.

Profile Data				
	VI Time	Sub VIs Time	Total Time	# Runs
Waveform Time to Date Time String.vi	8160.7	0.0	8160.7	2500000
Untitled 3 (SubVI)	4803.7	8160.7	12964.4	25
Untitled 1 (SubVI)	1374.8	0.0	1374.8	25
Untitled 2 (SubVI)	353.3	0.0	353.3	25
WriteToCSV.vi	194.5	14692.6	14887.1	1

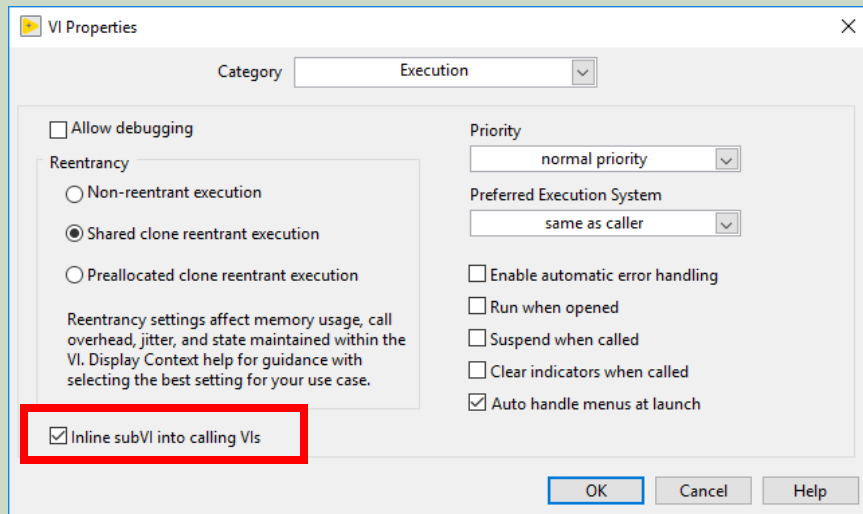
Watch out for sub-arrays!



VI Settings

VI Settings

- **Inline VIs** that run a lot
 - Removes subVI overhead
 - Opens up potential optimizations when subVI boundaries are removed
 - Dead code elimination, Constant folding, etc.
- Don't worry about *Priority* or *Preferred Execution System*
- Save copies of vi.lib VIs to inline and optimize them (!)
 - Give them a different icon
 - Document the caller VI



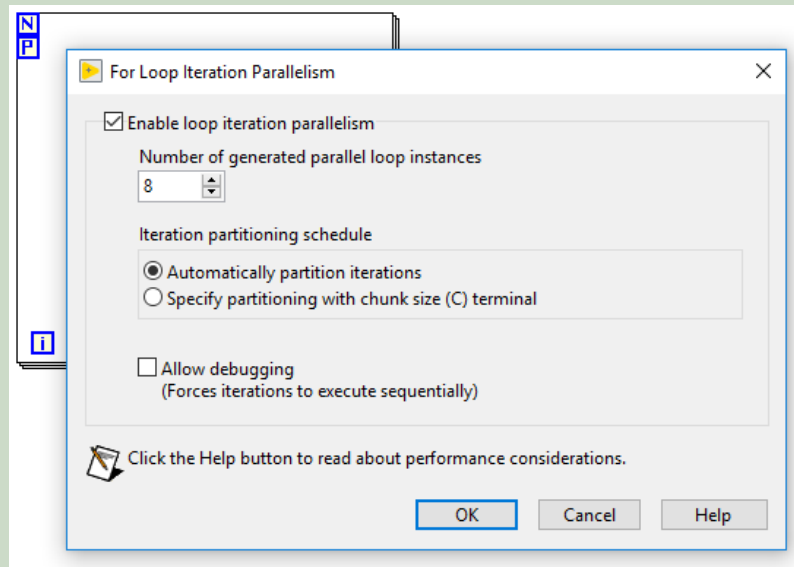
VI Settings – When to apply them

- Inline VIs don't show up in the VI Profiler 😞
- Mark as inline after you're done profiling to get that last speed boost
- **Turn off debugging** on non-inline VIs after you're done profiling
 - (Debugging setting doesn't matter for non-debug outputs like EXEs and PPLs)

Parallel For Loops

Parallel For Loops

- **Easiest** way to speed up existing For Loop code
 - The **first thing** I look for when I get a “slow VI”
- Parallelize the outer-most loop
 - Don't parallelize nested loops
 - (with rare exception)
- VI will become broken if the loop cannot be parallelized



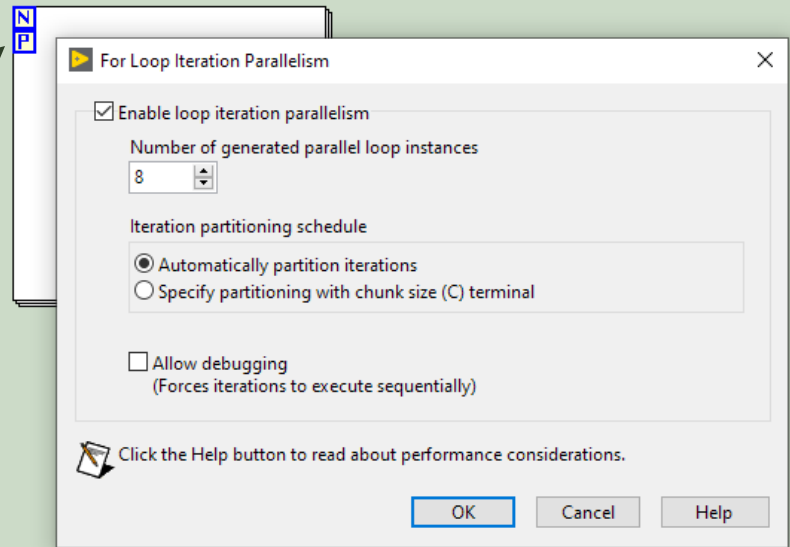
Parallel For Loops – How many loop instances?

- Don't wire 'P' (see guidance below)
- 'Number of generated parallel loop instances' specifies the **maximum** number of parallel instances the LabVIEW compiler will generate
- "Just use 8"
 - (unless you know for sure you'll need more)

-1: Use value in dialog

0 (unwired): Use the most available logical processors (up to configured value)

1 or greater: Use wired value (up to configured value)



Programming Patterns for Performance

Programming Patterns for Performance part 1

- Control and Indicator terminals always on the top-level diagram (of subVIs)
- Remove decision points from diagrams if you can
 - Like error case structures
- Basic string primitives vs. “newer” stuff like JSON (!)
- Consolidate class accessors in tight loops (!)
 - ...or get the data out of classes before the tight loop starts (!)

Programming Patterns for Performance part 2

- Modifying cluster and array elements
 - If you need the original element value, use In Place Element Structure
 - If you don't, use Bundle By Name or Replace Array Element
 - NEVER delete/index from array then rebuild
- If you see multiple branches of a (large) array wire, you **may** need a DVR
 - Or if you have the large array in a promiscuous functional global variable
 - When refactoring for performance, DVRs should be a last resort

Sets and Maps

Good/Gooder/Better/Betterer/Best/Bester

- Good – Search 1D Array
 - Gooder – Search Unsorted 1D Array.vim
- Better – Custom binary search
 - Betterer – Search Sorted 1D Array.vim
- Best – Variant Attributes
- Bester – **Sets and Maps**

Performance Benefits of Maps

Maps **eliminate the data type conversion** required to store variant attribute keys as strings and values as variants. Plus, they're an actual API and not a hack. 😊

Variant attributes are comparably performant **if** your keys are already strings and your values are already variants. (!)

If you find yourself dropping a Search 1D Array or a Build Array, ask yourself if you should be using Sets or Maps instead.

See my **All About Collection Data Types** presentation for more info:
<https://bit.ly/dnattcollections>

Real-world Demos



Thanks for attending!

Remember, you can download the presentation here:

bit.ly/slowvis

Parallelize your loops. Inline your subVIs. Profile your VIs. Write fast code.



dnatt.org