Improving the Performance of Your NI LabVIEW Applications

Dan Hedges
Senior Software Engineer
LabVIEW Performance Group
Agenda

• How to find performance problems
  ▪ Benchmarking
  ▪ Profiling
• Understanding LabVIEW under the hood
  ▪ Memory usage
  ▪ Execution system
Optimization Cycle

Benchmark
- Evaluate Performance
- Identify Problem Areas

Optimize
- Improve efficiency
- Improve Speed
Benchmarking Code Execution

“Timing Template (data dep)” – LabVIEW Shipping Example
Benchmarking Code Execution

Calibration

Code

“Benchmark Project” – LabVIEW Real-Time Shipping Example
Tools for Measuring Resource Usage (Windows)

- Task Manager
- Perfmon
Windows Task Manager

- Gives user a rough idea of whether memory or CPU is the bottleneck

- Can be helpful in identifying memory leaks

- View » Select Columns … allows you to add additional stats
Perfmon

• Allows you to monitor
  • Processors
  • Disk I/O
  • Network Tx/Rx
  • Memory/Paging

• Access by typing “perfmon” into the Windows Run dialog
Why Should You Profile Your VIs?

The 80/20 rule of software performance

• 80 percent of the execution time is spent in 20 percent of the code

• Performance improvements are most effective in the 20 percent
• Guessing which 20 percent is difficult
VI Profiler

- Tools >> Profile >> Performance and Memory…
Demo – VI Profiling
LabVIEW Desktop Execution Trace Toolkit

- Detailed execution traces
- Thread and VI information
- Measurement of execution time
# Profiling and Benchmarking Summary

<table>
<thead>
<tr>
<th>To answer this question:</th>
<th>Use these tools:</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is my current performance?</td>
<td>Benchmark VIs</td>
</tr>
<tr>
<td>What are my limiting resources?</td>
<td>Task Manager, Perfmon</td>
</tr>
<tr>
<td>How much time are each of my VIs taking?</td>
<td>VI Profiler</td>
</tr>
<tr>
<td>In what order are events occurring?</td>
<td>LabVIEW Desktop Execution Trace Toolkit</td>
</tr>
</tbody>
</table>
Under LabVIEW’s Hood

Memory Management

Execution System
What Is In Memory?

Panel

Diagram

Compiled Code

Data
VIs in Memory

• When a VI is loaded into memory
  ▪ We always load the data
  ▪ We load the code if it matches our platform (x86 Windows, x86 Linux, x86 Mac, PowerPC Mac)
  ▪ We load the panel and diagram only if we need to (for instance, we need to recompile the VI)
Panel and Diagram Data

- How many bytes of memory does this VI use?
- The answer depends on:
  - Is the panel in memory?
  - Is the environment multi-threaded?
Execute, Operate and Transfer Data

4K Execute Data
- Populated by Code

4K Transfer Data
- Temporary Buffer

4K Operate Data
- Copy for Indicator
Avoid Loading Panels, Save Memory
Wire Semantics

• Every wire is a buffer
• Branches typically create copies
Optimizations by LabVIEW

The theoretical 5 copies become 1 copy operation

Output is “in place” with input
The “In Place” Algorithm

• Determines when a copy needs to be made
  ▪ Weighs arrays and clusters higher than other types

• Algorithm runs during compilation, not execution
  ▪ Does not know the size of an array or cluster

• Relies on the sequential aspects of the program
  ▪ Branches may require copies
Bottom Up

In-place information is propagated bottom up.

Copy because of increment

No copies required

Increments array in place
Showing Buffer Allocations
The In-Place Element Structure

Allows you to explicitly modify data “in place”
Example of In Place Optimization

Operate on each element of an array of waveforms
Make the First SubVI “In Place”

changes into…
SubVI 2 Is Made “In Place”

Changes into ...

ni.com
SubVI 3 is Made “In Place”

Changes into …
Final Result: Dots Are Hidden
Building Arrays

There are a number of ways to build arrays and some are better than others

Bad

- Reallocates array memory on every loop iteration
- No compile time optimization
Building Arrays

There are a number of ways to build arrays. Try to minimize reallocations.

Best

- Memory preallocated
- Indexing tunnel eliminates need for copies
Demo – Effects of Memory Optimization
Under LabVIEW’s Hood

Memory Management

Execution System
VIs Are Compiled
VIs Are Compiled: "Clumps"
VIs Are Compiled: “Clumps”

Start of diagram: Reads controls, then schedules Clumps 1 and 2
Then sleeps ...

Completion of diagram: Divide nodes, display of indicators, then VI exit

Top for loop indicator is updated
Clump 0 Scheduled
Sleep ...

Bottom for loop indicator is updated
Clump 0 Scheduled
Sleep ...

Clump 1 Sleeping

Clump 2 Sleeping
Single-Threaded LabVIEW

CPU

Thread

User Interface Loop

Coroutines

Code Execution
Multithreaded LabVIEW

CPU

Thread messages Thread Thread Thread

UI Loop Exec Exec Exec
LabVIEW on a Multicore Machine

CPU0

UI Loop

Thread

Exec

messages

CPU1

Thread

Exec

Thread

Exec
Some Operations Require the UI Thread

- Front Panel Control References
- Call Library Nodes
- Control/Indicator Property Nodes
Execution Properties

Priority:
- normal priority

Allow debugging
- Reentrant execution
  - Share clones between instances (reduces memory usage)
  - Preallocate clone for each instance (maintains state for each instance)

Preferred Execution System:
- same as caller

Enable automatic error handling
- Run when opened
- Suspend when called
- Clear indicators when called
- Auto handle menus at launch

Inline subVI into calling VIs
Reentrant VIs

- Reentrancy allows one subVI to be called simultaneously from different places
  - Requires extra memory for each instance
- Use reentrant VIs in two different cases
  - To allow a subVI to be called in parallel
  - To allow a subVI instance to maintain its own state
LabVIEW 2010 Compiler

- Generates code that runs faster, ~30%
- Takes longer to run (~5x-7x)

- SSE Instructions
- SubVI Inliner
- Register Candidates
- Dead Code Elimination
- Loop Invariant Code Motion
- Common Subexpression Elimination
- And more
Demo – Effects of Execution Optimization
Next Steps

In LabVIEW

• LabVIEW Help

On the Web

• ni.com/multicore
• ni.com/devzone