LabVIEW Memory Management and Code Optimisation

Jacques Cilliers
Applications Engineering
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

Place the functions or VIs you want to time here.

Initial Tick Count

Final Tick Count

\( \text{Elapsed Time (sec)} \)

\( \text{< disable indexing} \)

\\(1000.0\\)
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…

#### Profile Performance and Memory - Tetris Demos.ivproj

<table>
<thead>
<tr>
<th>Application Instances</th>
<th>Timing statistics</th>
<th>Timing details</th>
<th>Time unit</th>
<th>Profile memory usage</th>
<th>Size unit</th>
<th>Select Application Instances…</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Profile Data

<table>
<thead>
<tr>
<th>VI</th>
<th>Sub VI Time</th>
<th>Total Time</th>
<th># Runs</th>
<th>Average</th>
<th>Shortest</th>
<th>Longest</th>
<th>Diagram</th>
<th>Display</th>
<th>Draw</th>
<th>Tracking</th>
<th>Locals</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - Check Collision.vi</td>
<td>0.0</td>
<td>1513.2</td>
<td>40352</td>
<td>0.0</td>
<td>0.0</td>
<td>15.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 - Board Ranking.vi</td>
<td>0.0</td>
<td>62.4</td>
<td>2579</td>
<td>0.0</td>
<td>0.0</td>
<td>15.6</td>
<td>62.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 - Test Drop in Place.vi</td>
<td>1450.8</td>
<td>1497.6</td>
<td>2579</td>
<td>0.0</td>
<td>0.0</td>
<td>15.6</td>
<td>46.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 - Main.vi</td>
<td>1684.8</td>
<td>1716.0</td>
<td>1</td>
<td>31.2</td>
<td>31.2</td>
<td>31.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 - Clear Completed Rows.vi</td>
<td>15.6</td>
<td>46.8</td>
<td>2579</td>
<td>0.0</td>
<td>0.0</td>
<td>15.6</td>
<td>31.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 - Find Optimal Move.vi</td>
<td>1669.2</td>
<td>1684.8</td>
<td>101</td>
<td>0.2</td>
<td>0.0</td>
<td>15.6</td>
<td>15.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 - Delete Rows.vi</td>
<td>15.6</td>
<td>15.6</td>
<td>48</td>
<td>0.3</td>
<td>0.0</td>
<td>15.6</td>
<td>15.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 - Test Rotate.vi</td>
<td>0.0</td>
<td>0.0</td>
<td>303</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

#### Buttons
- Start
- Snapshot
- Save
- Close
- Help
LabVIEW Desktop Execution Trace Toolkit

- Detailed execution traces
- Thread and VI information
- Measurement of execution time
- Included with LabVIEW Professional
### LabVIEW Desktop Execution Trace Toolkit

<table>
<thead>
<tr>
<th>#</th>
<th>Time</th>
<th>VI</th>
<th>Event</th>
<th>Thread ID</th>
<th>CPU ID</th>
<th>Highlight</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>97059</td>
<td>10:06:37.0515112</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97060</td>
<td>10:06:37.0515122</td>
<td>0 - Check Collision.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97061</td>
<td>10:06:37.0515837</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97062</td>
<td>10:06:37.0515842</td>
<td>0 - Check Collision.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97063</td>
<td>10:06:37.0516552</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97064</td>
<td>10:06:37.0516557</td>
<td>0 - Check Collision.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97065</td>
<td>10:06:37.0517263</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97066</td>
<td>10:06:37.0517272</td>
<td>0 - Check Collision.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97067</td>
<td>10:06:37.0517978</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97068</td>
<td>10:06:37.0517988</td>
<td>0 - Check Collision.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97069</td>
<td>10:06:37.0518693</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97070</td>
<td>10:06:37.0518703</td>
<td>0 - Check Collision.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97071</td>
<td>10:06:37.0519413</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97072</td>
<td>10:06:37.0519423</td>
<td>0 - Check Collision.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97073</td>
<td>10:06:37.0520129</td>
<td>0 - Check Collision.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1189</td>
</tr>
<tr>
<td>97074</td>
<td>10:06:37.0520176</td>
<td>0 - Clear Completed Rows.vi</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1265</td>
</tr>
<tr>
<td>97075</td>
<td>10:06:37.0520248</td>
<td>0 - Clear Completed Rows.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1265</td>
</tr>
<tr>
<td>97076</td>
<td>10:06:37.0520253</td>
<td>0 - Test Drop in Place.vi</td>
<td>VI Return</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:900</td>
</tr>
<tr>
<td>97077</td>
<td>10:06:37.0520267</td>
<td>0 - Board Ranking.vi:1770001</td>
<td>VI Call</td>
<td>5</td>
<td>2</td>
<td>Object</td>
<td>subVI UID:1249</td>
</tr>
</tbody>
</table>
# 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
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

Branched wire

Copy

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 …
SubVI 3 Is Made “In Place”

changes into ...

ni.com
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
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 ...

Clump 0

Clump 1
Top for loop indicator is updated
Clump 0 Scheduled
Sleep ...

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

Clump 1 Sleeping

Clump 0 Sleeping

Clump 0

Clump 2
Bottom for loop indicator is updated
Clump 0 Scheduled
Sleep ...

Clump 2 Sleeping
Multi-threaded LabVIEW

CPU

Thread

UI Loop

messages

Thread

Thread

Thread

Exec

Exec

Exec
LabVIEW on a Multicore Machine
Some Operations Require the UI Thread

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

Checkbox for Enable automatic error handling is marked.
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
Effects of Execution Optimization
Next Steps


Relevant Courses

• LabVIEW Performance