EECS Berkeley EE249
LabVIEW Framework

Hugo A. Andrade
Principal Architect
LabVIEW Platform & Real Time Group
Agenda

• Overview of NI Tools
• LabVIEW
  – Intro
  – Framework
  – Parallel programming
    • From multi-core to many-core
National Instruments Profile

- Leaders in Computer-Based Measurement and Automation
- Strong Track Record of Growth and Profitability
- $740 MM Revenue in FY2007
- Past nine consecutive years voted Fortune’s 100 Best Companies to Work For in America
- Headquartered in Austin, Texas
- More than 4,000 employees; operations in 40+ countries
Diversity of Customers

- Customers are mainly domain experts, scientists and engineers
- Top 100 customers ≈ 35% of revenue
- More than 25,000 customers in more than 90 countries
- 95% of Fortune 500 manufacturing companies have adopted Virtual Instrumentation
Diversity of Industries

- Telecom
- Automotive
- Semiconductors
- Electronics
- Computers
- ATE
- Military/Aerospace
- Advanced Research
- Petrochemical
- Food Processing
- Textiles
Virtual Instrumentation Applications

- **Analysis and Design**
  - Simulation
  - Signal and Image Processing
  - Embedded System Programming
    - (PC, DSP, FPGA, Microcontroller)
  - Prototyping
  - And more...

- **Control**
  - Automatic Controls and Dynamic Systems
  - Mechatronics and Robotics
  - And more...

- **Measurement/Test**
  - Circuits and Electronics
  - Measurements and Instrumentation

A single graphical development platform

ni.com
LabVIEW Graphical Development System

• Graphical Programming Environment
• Compile code for multiple OS and devices
• Useful in a broad range of applications
LabVIEW

- LabVIEW Background and Intro
- Facets
  - Measurement and Automation
    - (Instrument Control, I/O, DAQ, etc.
  - General purpose computing environment for scientists and engineers
  - Modeling Environment
  - Framework for Domain Experts
  - Embedded Development Environment
  - Framework for Parallel Programming Techniques
LabVIEW 8.6

Distributed Multi-core Systems

Multiple Programming Models
LabVIEW Virtual Instrument

Front Panel

Block Diagram
Introduction to LabVIEW and Dataflow
The G (LabVIEW) Language Model

- Homogenous dataflow language
  - Structured case (switch, select), loops, timed specs
    - “Structured dataflow”
- Turing complete
- Run-time scheduling
  - Inherently parallel language
- Synthesizable language
  - Direct compilation to software binaries on host processors
  - Embedded Processors (via C)
  - To FPGAs (via VHDL)
Section Outline: LabVIEW and Dataflow

1. Introduction to the LabVIEW environment
2. Functions and hierarchy
3. Dataflow and parallelism
What is LabVIEW?

- Programming language: graphical programming based on structured dataflow
- Complete development environment: project explorer, debugging tools, source code control, and more
Parts of a LabVIEW Application

• Block diagram shows graphical source code

• Front panel contains graphical user interface (GUI) items (connected to source code)
Block Diagram: Source Code

• Block diagram can contain functions, structures, and constants

• Analogous to functions, subroutines, flow control elements, and constants in text-based languages

• Terminals represent inputs and outputs
3 Types of Functions (from the Functions Palette)

Express VIs: interactive VIs with configurable dialog page (blue border)

Standard VIs: modularized VIs customized by wiring (customizable)

Functions: fundamental operating elements of LabVIEW; no front panel or block diagram (yellow)
What Types of Functions are Available?

• Input and Output
  – Signal and Data Simulation
  – Acquire and Generate Real Signals with DAC
  – Instrument I/O Assistant (Serial & GPIB)
  – ActiveX for communication with other programs

• Analysis
  – Signal Processing
  – Statistics
  – Advanced Math and Formulas
  – Continuous Time Solver

• Storage
  – File I/O
Cross-Platform Graphical Programming

Graphical Source Code → Compiled Code (binary)
LabVIEW Deployed

- Wireless Sensors
- Smart Cameras
- Embedded (FPGA)
- Handheld
- Wireless
- Networked I/O
- PC Boards
- Industrial Computer (PXI)
- Touch Panel
- Tektronix Open Windows Oscilloscopes
- Portable
- Workstation

ni.com
Block Diagram: Functions Palette

- Contains programming structures and functions
- Contains specialized IP, including math libraries built on Intel Math Kernel Library (MKL)
Front Panel: Controls Palette

• Controls (inputs): menus, text entry, selectors, buttons, etc.

• Indicators (outputs): graphs, LEDs, charts, and other displays
Hierarchy in LabVIEW

• A LabVIEW subVI is a VI that is called inside of another VI

• A subVI’s inputs and outputs (parameters) are exposed and accessed on the block diagram

• Similar to a function or subroutine in text-based programming
Dataflow Programming

• Block diagram execution
  – Dependent on the flow of data
  – Block diagram does NOT execute left to right

• Node executes when data is available to ALL input terminals

• Nodes supply data to all output terminals when done
Dataflow and Parallel Execution

• Dataflow programming means inherent parallelism in your application

• Parallel code sections in a dataflow program can be easily visualized
Dataflow and Parallel Execution

- Program massively parallel hardware with LabVIEW FPGA
- Same approach as programming CPUs, only code executes directly in hardware
Simple LabVIEW G Demo
Increasing Levels of Hardware Abstraction

- Vacuum Tubes
- Transistors
- Integrated Circuits (IC)
- Very Large Scale ICs (VLSI)
- System-on-Chip (SoC)
Increasing Levels of Software Abstraction

System complexity

Abstraction

Machine code
Assembly language
Procedural Lang. (C)
Object Oriented Lang. (C++)
System design platform
LabVIEW Targets

- Scalable from distributed network to sensors
LabVIEW on cRIO - HW

CompactRIO

Real-Time Processor

Reconfigurable FPGA

Industrial I/O Modules

• **Industrial I/O Modules** with built-in signal conditioning for direct connection to sensors/actuators

• **Reconfigurable FPGA** for high-speed and custom I/O timing, triggering, control

• **Real-Time Processor** for deterministic, stand-alone operation and advanced analysis
LabVIEW on cRIO – Logical Diagram
LabVIEW on cRIO – SW and Some Ideas
High-Level Development Tools

Data Flow

C Code

Textual Math

Modeling

Statechart

LabVIEW™
Graphical System Design Platform

Linux®
Macintosh
Windows

Real-Time
FPGA
Micro

Desktop Platform
Embedded Platform
LabVIEW as a Framework

- Demo Express VI
- Demo Vision Assistant
  - Metal example
- Demo Motion Assistant
Textual Math in LabVIEW

• Integrate existing scripts with LabVIEW for faster development
• Interactive, easy-to-use, hands-on learning environment
• Develop algorithms, explore mathematical concepts, and analyze results using a single environment
• Freedom to choose the most effective syntax, whether graphical or textual within one VI

Supported Math Tools:
- MathScript script node
- Mathematica software
- Maple software
- MathSoft software
- MATLAB® software
- Xmath software

MATLAB® is a registered trademark of The MathWorks, Inc.
Math with the MathScript Node

- Implement equations and algorithms textually
- Input and Output variables created at the border
- Generally compatible with popular m-file script language
- Terminate statements with a semicolon to disable immediate output

Prototype your equations in the interactive MathScript Window.
The Interactive MathScript Window

- Rapidly develop and test algorithms
- Share Scripts and Variables with the Node
- View /Modify Variable content in 1D, 2D, and 3D

(LabVIEW » Tools » MathScript Window)

ni.com
Developing Applications with the NI LabVIEW Statechart Module
What are Statecharts?

Statecharts are visual representations of reactive (event-based) systems.
Differences between Statecharts and FSMs

Both contain the same basic concepts:
  – States
  – Transitions

Statechart adds additional concepts:
  – Hierarchy
  – Concurrency
  – Event-based paradigm
  – Pseudostates & Connectors

Based on the UML statechart diagram specification
Reactive Systems

- Communication systems
- Digital protocols
- Control applications
  - Sequential logic
  - Batch processing
  - Event response
  - Non-linear control
- User-interface implementation
- System modeling for virtual prototyping (simulation)
Statechart Benefits

• Abstraction
  – Simple semantics to represent complex systems
  – System-level view
  – Self-documenting
Machine & Process Control

- **Concurrency**
- **Hierarchy**
Statechart Benefits

• Abstraction
  – Simple semantics to represent complex systems
  – System-level view
  – Self-documenting

• Scalability
  – Easily extend applications
  – Open software platform

• Automatic Code Generation
  – LabVIEW Embedded Technology
LabVIEW Simulation Module

• Develop dynamic systems such as motor controllers and hydraulic simulators with LabVIEW
• Implement your dynamic systems with real-time I/O using built-in LabVIEW data acquisition functions
• Simulate linear, nonlinear, and discrete systems with a wide array of solvers
• Deploy dynamic systems to real-time hardware with the NI LabVIEW Real-Time Module
• Translate models from The MathWorks, Inc. Simulink® into LabVIEW with built-in utility
The Design Process

1. **Modeling** – Identify a mathematical representation of the plant
2. **Control Design** – Choose a control method and design a controller
3. **Simulation** – Employ a point-by-point approach to simulate the system timing with a solver
4. **Tuning and Verification** – Introduce real-world nonlinearities, tune, and verify the control algorithm
5. **Deployment** – Implement the finalized control system
The Simulation Loop

- Built in Differential Equation Solver allows continuous-time system
- Similar to a While Loop with a predefined time period
- Installed with Simulation Module
- Double-click Input Node to configure simulation parameters
- Create an indicator on the Output Node to display Simulation errors
Simulation Loop Parameters

- Drag left node to show current parameters and provide inputs for run-time simulation configuration.

- Double-click Input Node to configure simulation parameters.
High-Level Development Tools

Data Flow

C Code

Textual Math

Modeling

Statechart

LabVIEW™

Graphical System Design Platform

Linux®

Macintosh

Windows

Real-Time

FPGA

Micro

Desktop Platform

Embedded Platform
Targets and Deployment

LabVIEW Real-time

LabVIEW FPGA

LabVIEW Microprocessor SDK

LabVIEW PDA and related
Evolution of LabVIEW Backend Technologies

Intermediate Code

- None (Machine Code)
- VHDL
- None (Object Library)
- C

Compiler

- LabVIEW Real-Time
- OEM Synthesis PAR
- LabVIEW DSP
- Any

Hardware Target

- Wintel PowerPC
- FPGA
- DSP
- Any 32-bit MPU
LabVIEW for Parallel Programming

- Overview of LabVIEW Multithreading
- Parallel Programming Techniques
- Real-Time Considerations
- Resources
Impact on Engineers and Scientists

Engineering and scientific applications are typically run on dedicated systems (i.e. little multitasking).

Diagram:
- Measurement or Control Application
  - Data Acquisition
  - User Interface
  - Network Comm.
  - Logging
- Operating System
  - CPU Core
  - CPU Core
Creating Multithreaded Applications

Engineers and scientists **must** use threads to benefit from multicore processors.

```
Measurement or Control Application

| Data Acquisition | User Interface | Network Comm. | Logging |

Operating System

| THREAD |

| CPU Core | CPU Core |

ni.com
```
Multithreading Programs in LabVIEW

• Automatic threading
  – LabVIEW clumping algorithm automatically assigns code to threads based on parallelism
  – Most existing programs will run faster on a Multicore system with no alteration

• Manual threading
  – Force section of code to execute in a single thread
LabVIEW divides a program into multiple threads (originally introduced in 1998 with LabVIEW 5.0)

Oversimplification shown below; parallel code paths execute in separate threads to run on parallel hardware
1. LabVIEW compiler analyzes diagram and assigns code pieces to “clumps”
2. Information about which pieces of code can run simultaneously are stored in a run queue
3. If block diagram contains enough parallelism, it will simultaneously execute in all system threads

# of threads scales based on # of CPUs
## Multithreaded Software Stack Support

<table>
<thead>
<tr>
<th>Software Stack</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development tool</strong></td>
<td>Support provided on the operating system of choice; tool facilitates correct threading and optimization</td>
<td>Example: Multithreaded nature of LabVIEW and structures that provide optimization</td>
</tr>
<tr>
<td><strong>Libraries</strong></td>
<td>Thread-safe, re-entrant libraries</td>
<td>Example: BLAS libraries</td>
</tr>
<tr>
<td><strong>Device drivers</strong></td>
<td>Drivers designed for optimal multithreaded performance</td>
<td>Example: NI-DAQmx driver software</td>
</tr>
<tr>
<td><strong>Operating system</strong></td>
<td>Operating system supports multithreading and multitasking and can load balance tasks</td>
<td>Example: Support for Windows, Mac OS, Linux® OS, and real-time operating systems</td>
</tr>
</tbody>
</table>
Manual Threading with Timed Structures

- Code within timed structure executes in a single thread
- Threads can be assigned a relative priority
- Set processor affinity
Parallel Programming Techniques to Improve Performance on Multicore Systems

- Task Parallelism
- Data Parallelism
- Pipelining
Task Parallelism

1) Look for tasks that can be run in parallel
2) Architect code to reflect this parallelism
   • Eliminate data dependencies
   • LabVIEW automatically identifies parallel code and can split into multiple threads!
Task Parallelism

- Multiple tasks, same data
- Multiple data-independent tasks
Task Parallelism Demo
Data Parallelism

1) Look for large data sets that can be processed in two or more “chunks” independently

2) Architect code:
   • Split the data apart
   • Process the data in parallel
   • Combine the individual results into one overall result
Data Parallelism

Data Set → Signal Processing → Result
Data Parallelism

You can speed up processor-intensive operations on large data sets on multicore systems.

Data Set

Subset 1
Subset 2
Subset 3
Subset 4

Signal Processing
Signal Processing
Signal Processing
Signal Processing

Combine Results
Data Parallelism Demo
Pipelining Strategy

• Many applications involve sequential, multistep algorithms that are executed over and over again

• Applying pipelining can increase performance
Pipelining Strategy

- **CPU Core**
- **Acquire**
- **Filter**
- **Analyze**
- **Log**

Time:
- $t_0$
- $t_1$
- $t_2$
- $t_3$
Pipelining in LabVIEW

Sequential

Pipelined

Note: Queues may also be used to pipeline data between different loops
Pipelining Demo
Multicore Challenges and Debugging

- Reentrancy and execution highlighting
- Thread synchronization
- Execution tracing
- Performance counters
- Data transfer considerations
Reentrancy

- All VIs in LabVIEW can be set to be reentrant
- Allows for each subVI to use separate memory space

All subVIs that will be executed multiple times in parallel need to be set as reentrant
Thread Synchronization

- No guarantee that an OS will schedule threads in the correct sequence without synchronization primitives.
- Order of events may change at each execution due to the way the threads are scheduled.

First execution: Thread 1 → Thread 2 → Thread 3
Second execution: Thread 2 → Thread 3 → Thread 1
Third execution: Thread 3 → Thread 1 → Thread 2
Code Synchronization with Dataflow

- Dataflow paradigm ensures synchronization

Parallel code paths are synchronized and order of execution is determined by LabVIEW wires

Dataflow is a key enabler for Multicore programming
Synchronization in LabVIEW

- When low-level synchronization is required, use synchronization mechanisms including:
  - Queues
  - Notifiers
  - Semaphores
  - Rendezvous
  - Occurrences
Execution Tracing

- On real-time systems, trace debugging can show thread activity at the OS level
- Thread activity on each core is displayed by selecting a particular CPU
Performance Counters

- Performance counters provide detailed system information (CPU usage, memory usage, and cache hits/misses)
- LabVIEW can call into Windows performance counters
- Example utilities:
  - Windows Perfmon
  - Intel’s VTune
Data Transfer Considerations

- Carefully consider system memory architecture when programming for Multicore
- Avoid transferring overly large data sets between cores (relative to cache size)
- Very small memory transfers can cause repeated cache coherency updates
Physical Processor Layout

- Distance between processors and quality of processor connections can have a large effect
- Transferring data over system bus is much slower than accessing a shared cache

**Shared Cache Example:**
Dual-Core, Dual Processor System

Source: Tian and Shih, Software Techniques for Shared-Cache Multi-Core Systems, Intel Software Network
Deterministic Real-Time Systems

LabVIEW 8.5 adds Symmetric Multiprocessing (SMP) for real-time systems.
Assigning Tasks to Specific Cores

In LabVIEW 8.5, users can reserve a core and then assign time critical code to run on it.
Affinity, pools, threads, CPUs, etc.

System Threads
- Auto load balancing
- CPU Pool

Timed-Structure Threads
- Thread-Structure Affinity Mask
- Auto load balancing
- CPU Pool

Pool Affinity Mask

Up to 32 CPUs
Execution Trace Toolkit

- LabVIEW Real-Time Only
- Thread view
- VI view (function view)
- Multicore support
- Debugging flags (memory manager collisions, mutex collisions, priority inheritance, ISRs, etc.)
- Trace Activity Sorting
Resources

www.ni.com/multicore