4.2 Incorporate Autoscheduled Parallel Testing for Optimized Throughput

*Improving Test Throughput with Emerging Technologies*

This topic will address increasing throughput in test and will specifically introduce a new concept in parallel testing called autoscheduling.
Why Parallel Testing?

1. Prevent bottleneck in test and streamline manufacturing processes
   - Most existing test systems cannot keep up with throughput requirements
   - Medium-to-high volume environments

2. Maintain quality standards
   - Avoid removing critical tests for sake of performance

3. Increase device utilization
   - Lowers cost to test products

There are three main reasons to implement parallel testing. The most common reason is to streamline your manufacturing process and to prevent a bottleneck in test. This is primarily due to the fact that as devices and products increase in functionality, the testing required increases proportionally. This does not always correlate with the rest of the manufacturing process however. Assembly time, as an example, is proportional to the number of components on a device; and more and more functionality is being packaged into the same component, reducing the total number of components needed, speeding up assembly.

Another reason to incorporate parallel test is to maintain quality standards by not pruning necessary tests in-order to increase throughput. This can sometimes be a common occurrence when test is feeling increasing pressure to improve throughput.

The third main reason is to increase device utilization of the instruments you have. Typically, your test system can include thousands of dollars worth of instruments that are being used less than 50% of the time. This cost is spread over the lifetime of the instrument and by increasing the number of devices tested during that period of time, you are effectively reducing the cost of test.
Options for Increasing Test Throughput

• Duplicate or fan out entire test systems
  – Cons: Expensive, large footprint, more operators

• Distribute tests among sub-systems
  – Cons: Large footprint, additional handling, more operators

• Develop a shared-resource parallel test system
  – Concurrently test multiple functions and/or UUTs in parallel
    • Share or duplicate hardware based on performance requirements
  – Pros: Optimizes device utilization, minimizes footprint, single operator
  – Cons: Some additional software development required

Traditional options for increasing throughput such as fanning out test systems or distributing sub-tests come at the expense of large investments in equipment, more personnel, and additional handling required. Shared-resource parallel testing, however, typically involves a single test station with some or all the equipment being shared by parallel processes testing separate UUTs. This type of parallel testing is the preferred choice for increasing throughput because it requires minimal investment, minimal footprint, and delivers substantial gains.

In general there are two types of shared-resource parallel testing:

1. Performing parallel tests on a single unit under test (UUT)
2. Testing multiple UUTs in parallel

The latter option is the most common form of parallel testing found in high-volume manufacturing environments.
Increasing Throughput without Adding Hardware using a Shared-Resource Approach

<table>
<thead>
<tr>
<th>Sequential</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UUT 1</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
<tr>
<td>UUT 2</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
<tr>
<td>UUT 3</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
<tr>
<td>UUT 4</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parallel</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UUT 1</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
<tr>
<td>UUT 2</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
<tr>
<td>UUT 3</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
<tr>
<td>UUT 4</td>
<td>Test 1 Test 2 Test 3</td>
</tr>
</tbody>
</table>

The timing diagram above compares the flow of four units under test (UUTs) tested sequentially and in parallel (sharing instruments per test).

Now the majority of test systems today operate sequentially, completing an entire suite of tests on one UUT before removing and beginning tests on the next UUT. The parallel execution of the UUTs takes advantage of idle instruments in the system to execute the UUTs in parallel. Parallel testing typically results in a 30–60% increase in throughput and instrument utilization.
Considerations for Architecting a Parallel Test System

• Instrument utilization rates
• Optimizing the number of parallel UUTs and tests
• UUT and test requirements (instruments, fixtures, drivers, and so on)
• Switch hardware and switch programming
  And
• Programming a parallel execution engine.

There are many questions you should ask to determine the ideal parallel test configuration for your application. In many cases it is not necessary to purchase additional hardware to develop a parallel test system. Many instruments in traditional test systems sit idle for more than 50% of the time during a test cycle. A parallel test system can leverage this idle time to increase throughput without sharing the hardware among the multiple UUTs. It is also important to consider the number of tests and/or sockets you wish to execute in parallel. This can vary widely from system to system depending on your throughput requirements and available hardware. You will also need to document which hardware devices are required to run for each test in the system so that you can prevent resource conflicts when performing tests in parallel. Another consideration is the necessary switching required to automate your parallel test system as well as the switch programming. Most importantly, how do you plan on developing your parallel execution engine to execute your test sequences, communicate with your operator interface and handle resource sharing and synchronization?
Common Parallel Test Development Pitfalls

- Low-level programming of multithreaded tasks and execution engine
- Thread synchronization and communication routines encapsulated in test code
- Lack of reuse or scalability for added test coverage or future test needs
- Error handling, results collection, and reporting per thread

Requires programming expertise!

There are also many common development pitfalls that one can encounter when building a parallel test system without the proper development tools and planning. Developing a parallel test system with general purpose programming tools is very difficult because of the low-level multithreaded programming. Managing the thread synchronization and execution can easily consume more than 50% of your total test development time and adds a significant level of complexity to your test code. This complexity and low-level programming often leads to rigid code that cannot be easily reused. There are also many other system level challenges associated with developing a parallel test application including separate error handling, result collection, and reporting for each thread. Implementing a parallel test system without modern test software development tools requires a high level of programming expertise and extended development time.
National Instruments TestStand test management software includes features designed specifically for use in architecting parallel test systems. Test engineers with moderate programming knowledge can use NI TestStand to develop sophisticated multithreaded test applications in a short amount of time. TestStand provides an off-the-shelf parallel test framework consisting of a multithreaded execution engine, asynchronous and batch execution models, built-in resource allocation, synchronization and test autoscheduling, and multithread safe operator interfaces for use in production environments. All of these features essentially turn the development of your parallel and multithreaded test system into simply the configuration of such a system.
Now the following timing diagram includes the concept of autoscheduling. Autoscheduling is essentially the reordering of your test execution to take advantage of idle resources and equipment. This not only increases your device and instrument utilization but it will decrease your overall test time. In the diagram illustrated, you are saving an additional 15–20% over parallel testing when compared to the sequential case.
When using the autoschedule step types, the execution flow is as follows: When the tests are launched in parallel, the first UUT acquires the resources to perform the first test. As opposed to standard parallel testing, the other UUT tests in an autoschedule group do not remain idle waiting for the resources used in the first test to become available. Instead, the other UUT executions “search” for test steps with available resources to start testing the other tests in the sequence.

When the Autoschedule DMM Test step executes, the corresponding resource will be acquired. If the resource is busy, then the sequence jumps to the next Autoschedule Scope Test step in another autoschedule section. If no resources are available, the execution will jump to the autoschedule section with the least threads waiting for the resource. There is also the option to include a specified timeout for the autoschedule group.

For a sequence to use autoscheduling, it needs to include at least one autoscheduled group plus one or more autoscheduled sections. For ease of use, the Auto Schedule step type automatically inserts the necessary steps that are needed.
Demo – Autoscheduled Parallel Testing

Compare test throughput and utilization rates
- Sequential
- Parallel
- Autoschedule

This TestStand demo displays an operator interface developed to test four UUTs in parallel. The demo executes testing both a filter test and a component test on each of the four UUTs using a sequential testing strategy as well as a parallel and autoschedule testing strategy.
Sample Results for a Parallel PXI Test System

- Software: TestStand, LabVIEW, NI Switch Executive
- Hardware: DMM, Arb (2x), Digitizer, Matrix Switch

<table>
<thead>
<tr>
<th></th>
<th>Execution Time</th>
<th>DMM Utilization Rate</th>
<th>Digitizer Utilization Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential Testing</td>
<td>43s</td>
<td>53%</td>
<td>44%</td>
</tr>
<tr>
<td>Parallel Testing</td>
<td>23s</td>
<td>78%</td>
<td>65%</td>
</tr>
<tr>
<td>Autoscheduling</td>
<td>25s</td>
<td>92%</td>
<td>76%</td>
</tr>
</tbody>
</table>
In summary, shared-resource parallel testing increases throughput, reduces instrumentation costs and reduces system footprint. Also, parallel test software technology, such as NI TestStand, greatly reduces the complexity, development time and can even optimize multithreading. Incorporating the concept of autoscheduling can also optimize your parallel testing by increasing instrument utilization rates and therefore throughput. All of this parallel testing can be implemented on a wide range of hardware configurations.