

Verification and Profiling tools

Dissemination Event – September 2014

Nick Lay
Dave George

Rapita Systems Ltd.

■ Introduction to Rapita Systems



www.rapitasystems.com

■ Rapita Systems Ltd.

Founded in January 2004

- Based in York, UK
- Spin-off from the University of York
- Approximately 30 employees

Specialise in *On-target Software Verification*

Products

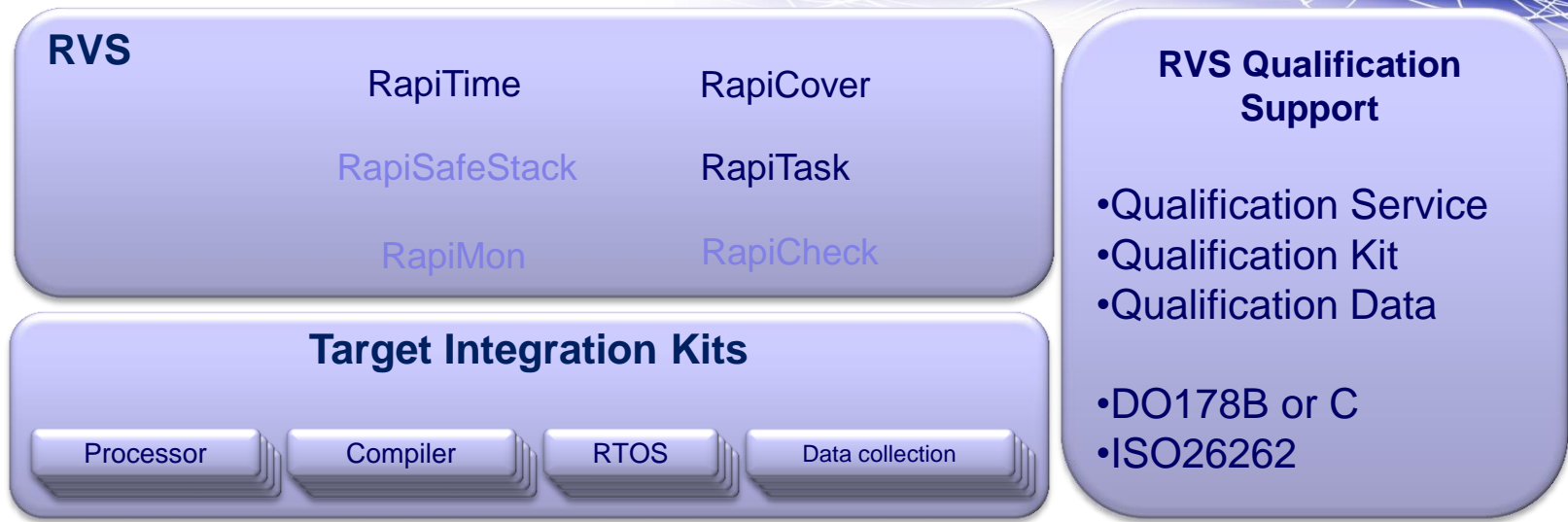
- RVS: Rapita Verification Suite
 - RapiTime: On-target Timing Analysis tool
 - RapiCover: On-target Code Coverage tool
 - RapiTask: RTOS schedule visualisation
 - RTBx: Hardware tracing solution.
- Consultancy services, support and training

Some customers and projects:

- BAE Systems (UK), GE Avionics (UK), Airbus (France), Liebherr (Germany), MTU (Germany), Honeywell (US), North West Industry University (China), Chinese Helicopter R&D Institute (China), Infineon (Germany), European Space Agency, ...



■ RVS in context



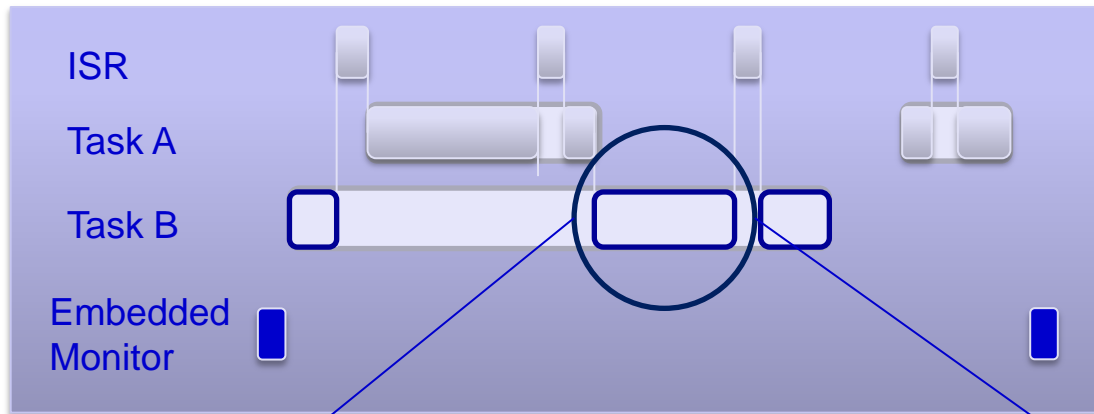
■ Code Level:

- **RapiTime:** Timing analysis, worst-case hotspots, WCET analysis, High Watermarks, average and worst-case profiler, worst-case path optimisation.
- **RapiCover:** On-target code coverage. Statement, Functions, Decision, Condition, MC/DC
- **RapiSafeStack (prototype):** Stack analysis. Static and dynamic.
- **RapiCheck (prototype):** Constraint checker. Dynamic verification of constraints.

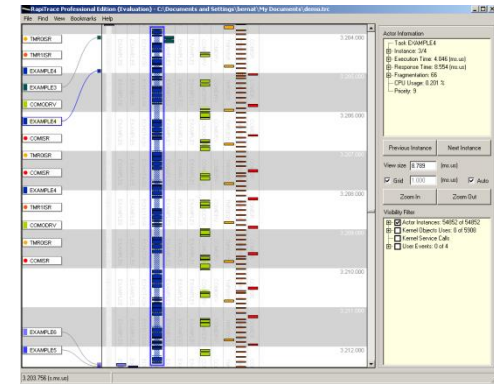
■ System Level:

- **RapiTask:** Visualisation of RTOS events.
- **RapiMon (prototype):** Embedded Monitor. On-the-fly monitoring of timing.
- **RapiCheck (prototype):** Constraint checker. Dynamic verification of constraints

Scope of RVS tools

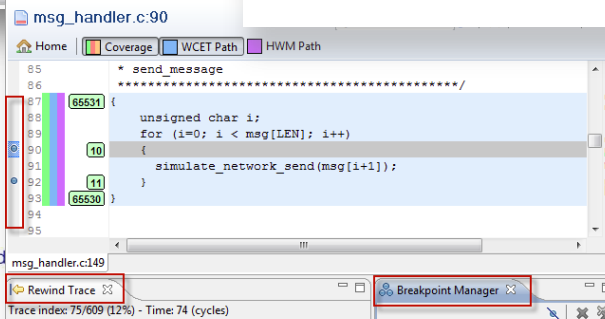
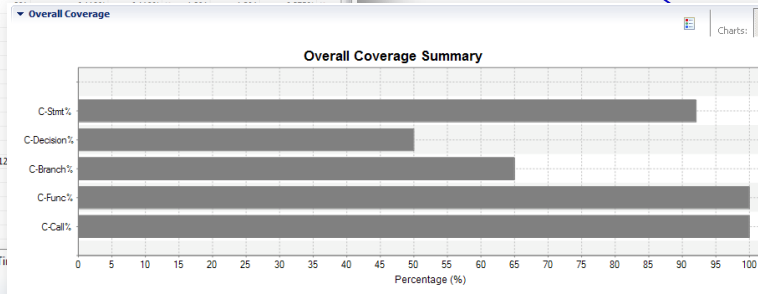


RapiTask -System level visualisation
RapiMon (P) - Embedded Monitor



Summary (109 items)

Name	Location	W-Freq	W-SetET	W-SetCT	W-SetCT%	W-SetCC%	W-SubET	W-SubCT	W-SubCT%	W-O
AirSpeed.Cycle	airspeed.adb:66-103	1	391							
AirSpeed.Extrapolate_Speed	airspeed.adb:31-60	1	226							
BC1553_Is_Fresh	bc1553.adb:86-91	12	78							
BC1553_Is_Valid	bc1553.adb:96-99	12	78							
BC1553_Read_Word	bc1553.adb:104-114	30	101							
BC1553_Write_Word	bc1553.adb:63-73	1	99							
BIT_Machine.Change_State	bit_machine.adb:24-30	12	52							
BIT_Machine.Phase	bit_machine.adb:33-36	12	0							
BIT_Machine.Step	bit_machine.adb:55-84	12	197							
Barometer.Cycle	barometer.adb:86-127	1	464							
Barometer.Extrapolate_Height	barometer.adb:46-80	1	250							
Bus.Cycle	bus.adb:531-591	1	123,190							
Bus_Is_BC_Fresh	bus.adb:430-453	12	78							
Bus_Is_BC_Valid	bus.adb:455-478	12	78							
Bus_Is_RT_Fresh	bus.adb:305-328	7	77							
Bus_Is_RT_Valid	bus.adb:330-353	7	78							
Bus_Read_BC_Word	bus.adb:481-504	30	84							
Bus_Read_RT_Word	bus.adb:355-378	7	85							
Bus_Write_BC_Word	bus.adb:379-402	7	85							



RapiTime – Timing analysis
RapiCover – Code Coverage

RapiSafeStack (P) – Stack Analysis
RapiCheck (P) – Constraint Checker

■ RVS Features

Multi-language

- C
- C++
- Ada

Multi-target

- Bespoke *Target Integration Kit* provides CPU/RTOS support
- Supports a wide range of data collection methods

Works with both simple and complex systems

- From single thread to complex MultiCore

Qualifiable

- DO-178B/C and ISO26262

■ RVS Benefits



Reduce Verification Costs

- Unified method, process and tool environment for on-target verification activities.
- Reduce unit-test costs by gathering evidence based on systems tests instead.
- Better evidence for verification than with other approaches.

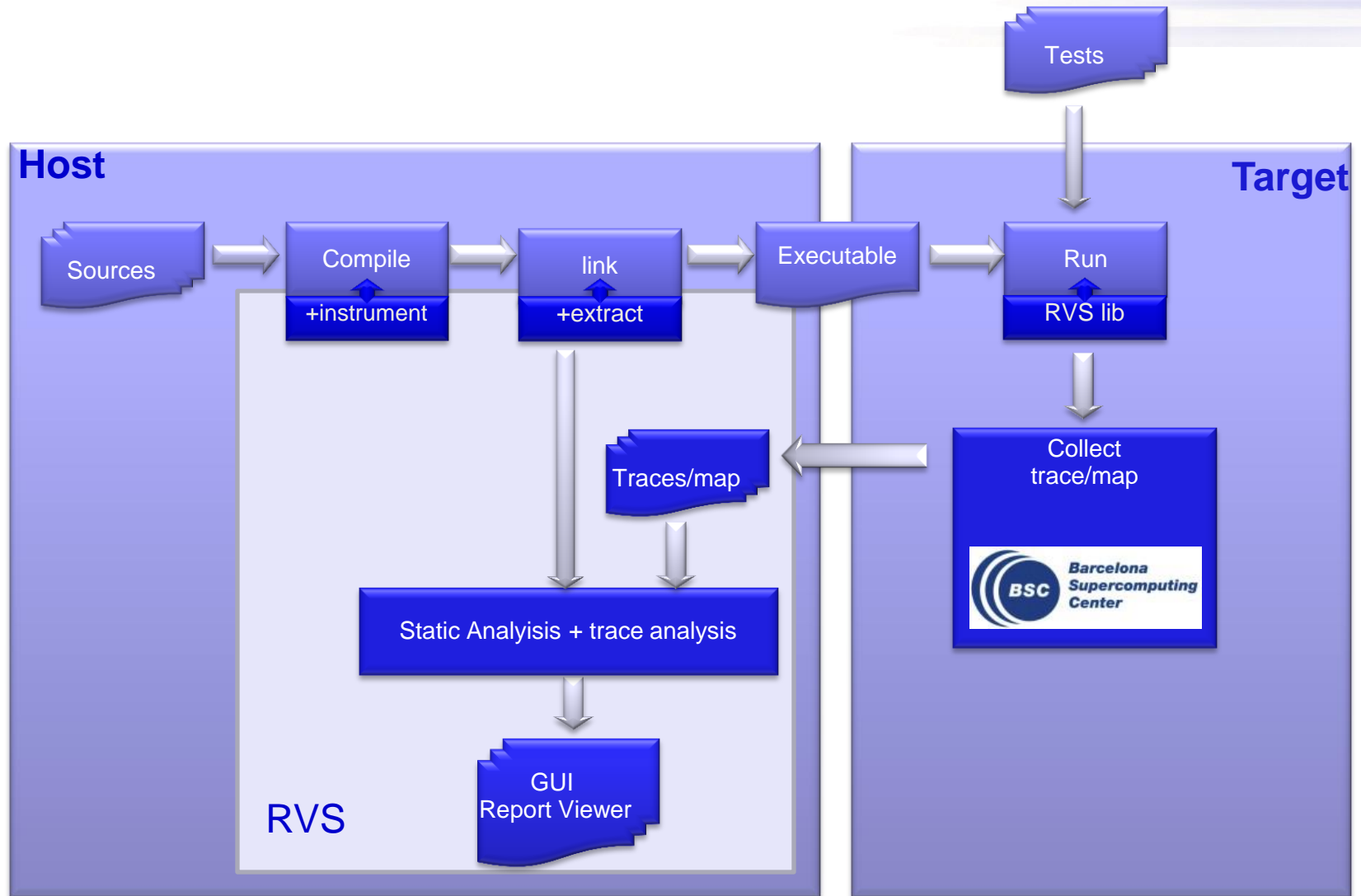
Reduce time and effort

- Simplify learning curve across different lanes, targets, RTOS and compilers
- Provides observability “inside the box”. Find issues of behaviour of code on target
- Specifically built to analyse embedded systems “on-target”
 - Lightweight instrumentation (specific Target Integration Kit)
 - Highly configurable so that it “fits” on the target
 - Integrates into customer specific build process
 - No more *Ad-hoc* methods
 - Can also be used “on-host”

Meet standards

- Supports DO178-B and ISO26262 verification requirements

■ Software testing with RVS



- RapiTime
- RapiTask
- RapiCheck
- RapiCover
- Rapita Dependency Tool

- Part of the **Rapita Verification Suite**
 - Code Coverage
 - Trace Visualisation
 - Timing Analysis
 - Constraint Checking
- *Measurement Based* Timing Analysis
- The best model of the machine is the machine itself!

- Fundamental part of using RVS.
- Since we *instrument* the source code, we need to *integrate* our tools with the build system ...
- ... Which we've done with the parMERASA platform.
- Since RVS instruments at the source level, anything with source code available can be analysed, including unique platforms (such as the parMERASA simulator)!
- We're not constrained to a set of known platforms. If you can build for it, we can probably analyse it!

```
int main (int argc, char** argv)
{
    int i;

    switch (argc)
    {
        case 0:
            break;
    }

    return 0;
}
```




```
int main (int argc, char** argv)
{RVS_I(65535);{
    int i;

    switch (argc)
    {
        case 0:
            RVS_I(10);break;
    }

    {RVS_I(65534);return 0;}
}}
```


- Barrier waiting time:
 - Time spent waiting at the barrier.
- Barrier execution time:
 - Execution time since last synchronisation at a barrier OR start of program.
- Lock waiting time:
 - Time spent waiting for the lock.
- Lock execution time:
 - Execution time for code between the lock and unlock.

```
mutex_lock (&mutex) ;  
atomic  () ;  
atomic2  () ;  
mutex_unlock (&mutex) ;
```



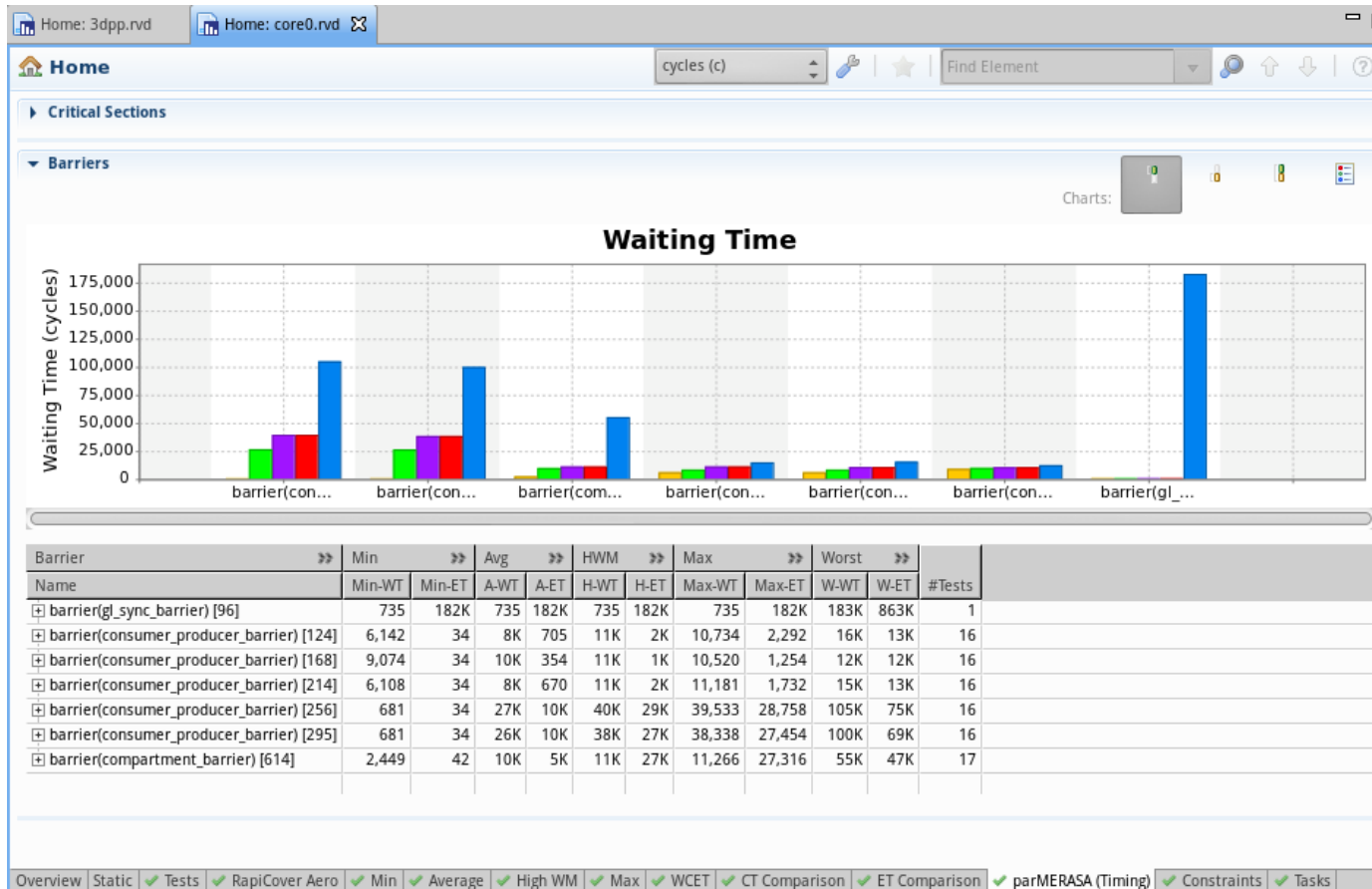
Lock
WCET

```
barrier_wait (&barrier) ;  
work  () ;  
more_work  () ;  
barrier_wait (&barrier) ;
```

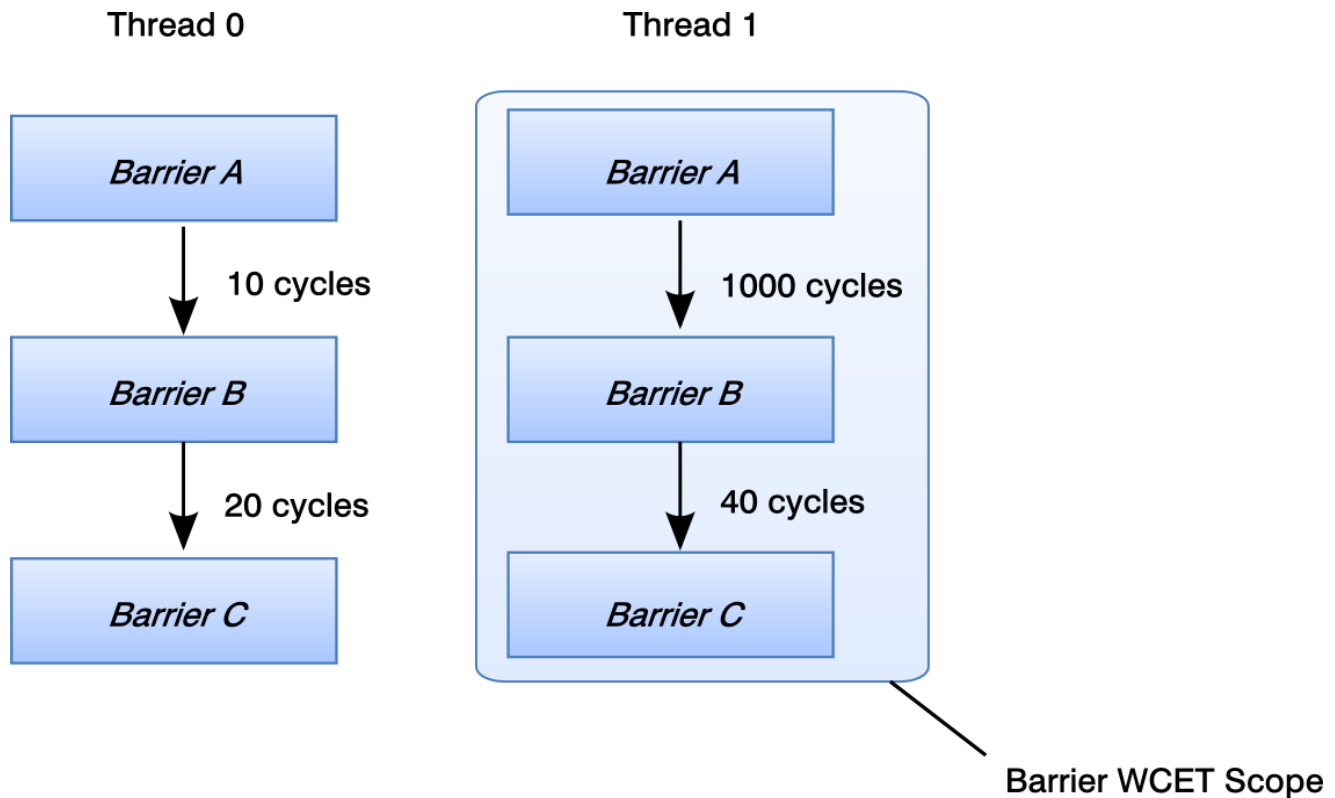


Barrier
WCET

- Support for measured time and WCET for multi-core synchronisation primitives (barriers and locks).

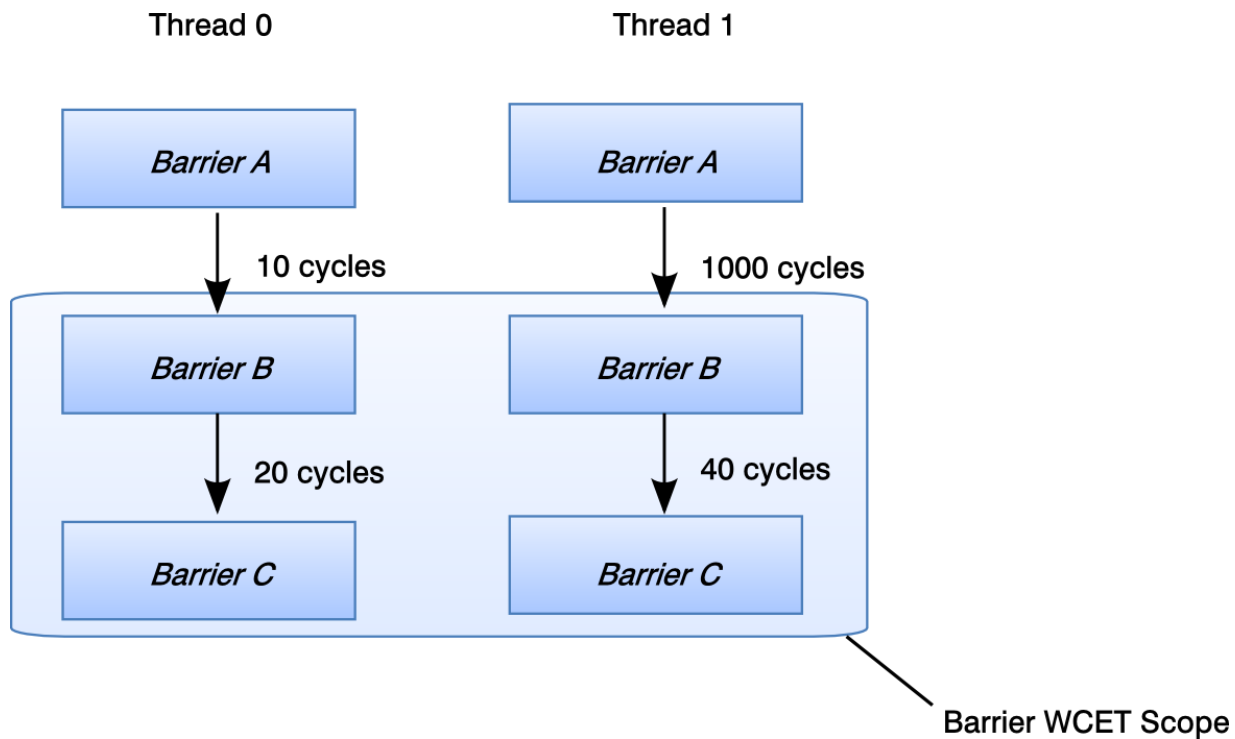


- Initial barrier timing similar to how we do sequential worst case.
- Consider worst case of a barrier to be *worst case time it could take any thread to get to the barrier from **any point***.



- Very pessimistic.
- **But ...** this method works for complicated multi-core applications with diverse barrier sequencing and synchronisation dependencies.
- So, it's useful in some situations.

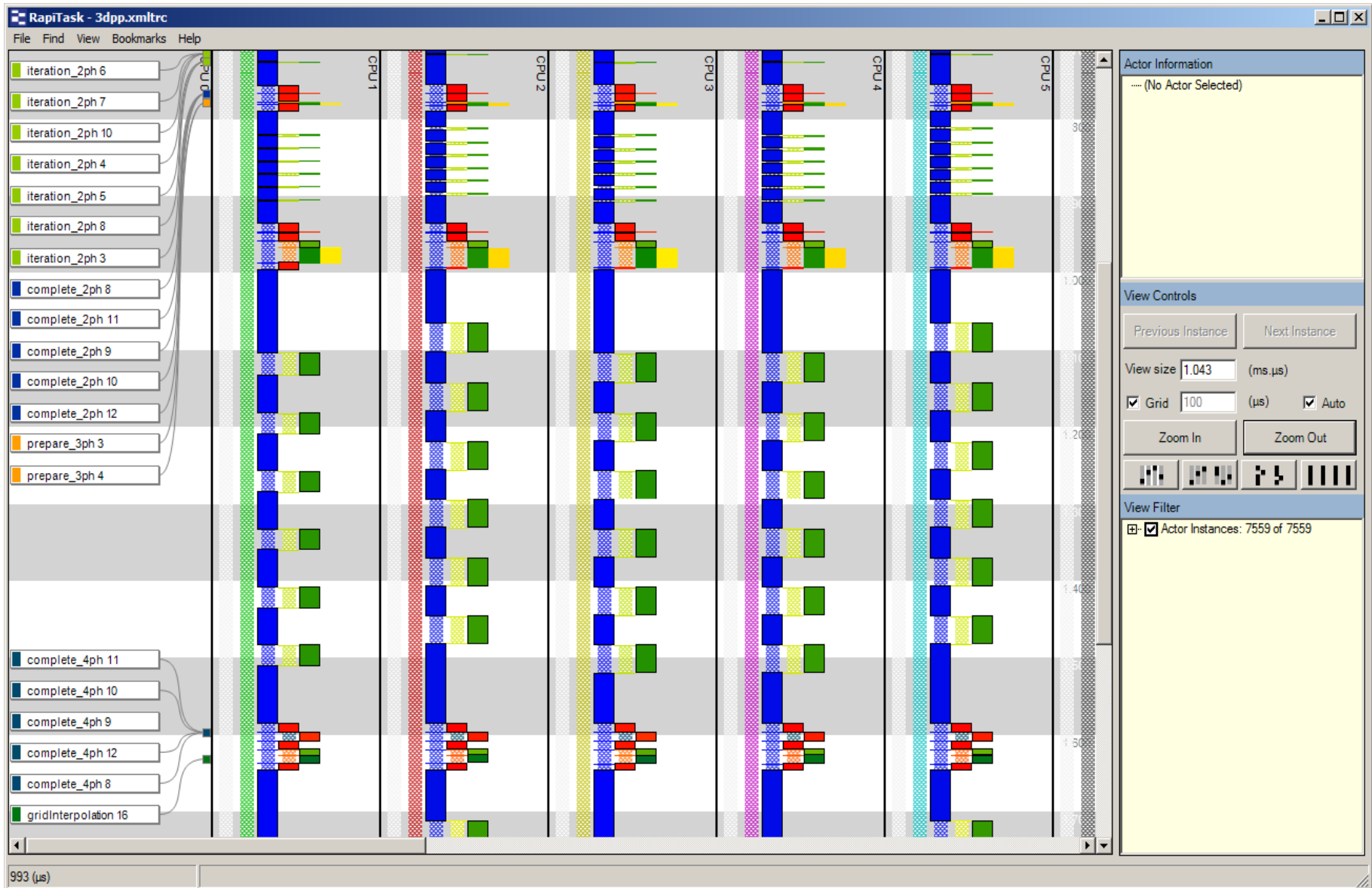
- If we constrain our usage of the primitives slightly we can get much, much tighter WCET values.
- Now we analyse timing per transition with the requirement that barrier sequences are the same across threads.

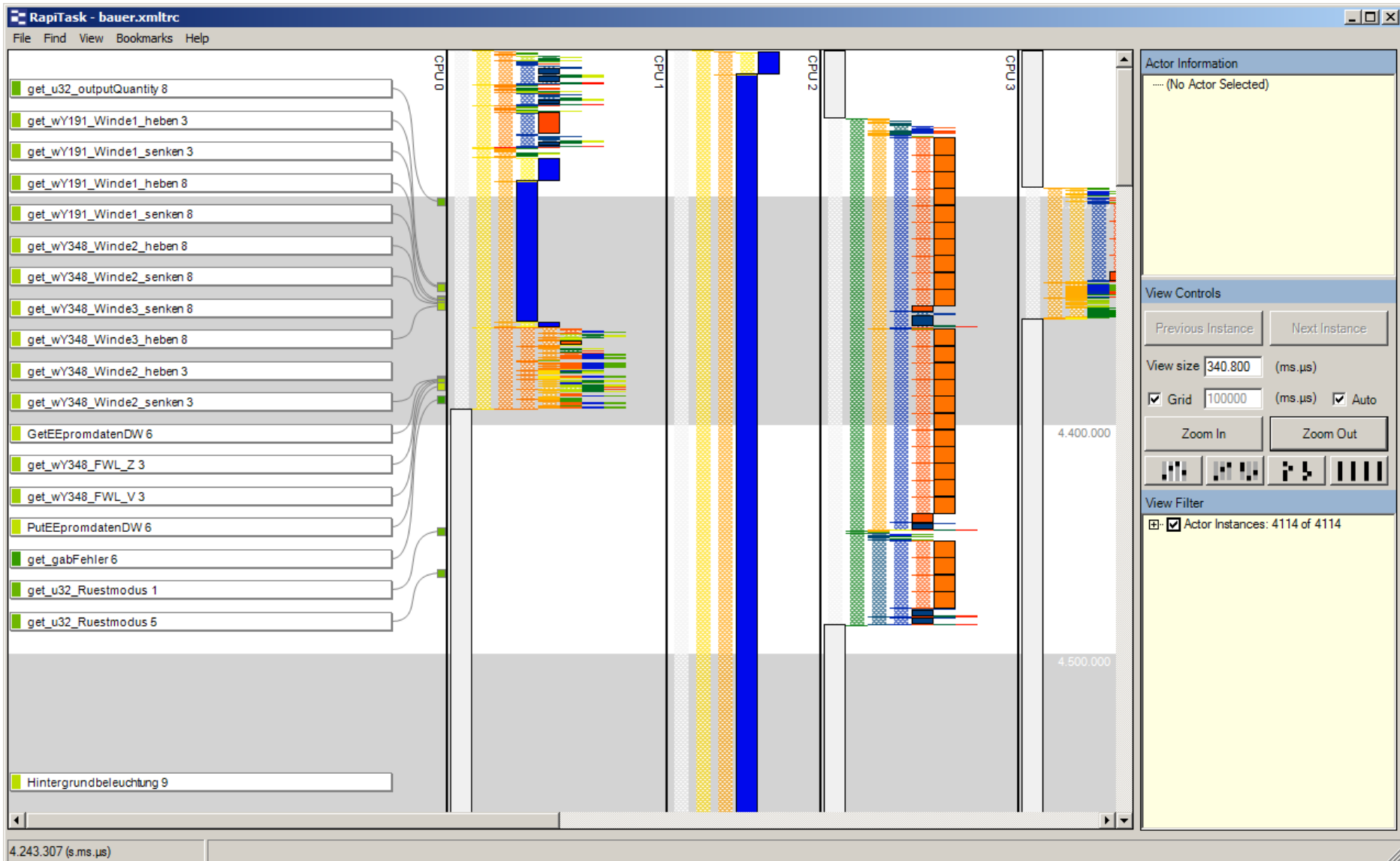


- What did we learn?
- We can analyse timing for complex usage of barriers and locks, but it will be pessimistic.
- Constrain synchronisation usage to barriers in lockstep and your pessimism drops significantly.
- Sensible and careful usage of constructs provides tighter and more controlled WCET!

- Allows WCET timing analysis for multi-core synchronisation without significant extra effort.
- Helps to identify possible problematic areas of synchronisation.
- Standard timing analysis when using locks/barriers can be either optimistic or pessimistic depending on tests seen.
- Low analysis overhead (time and memory).

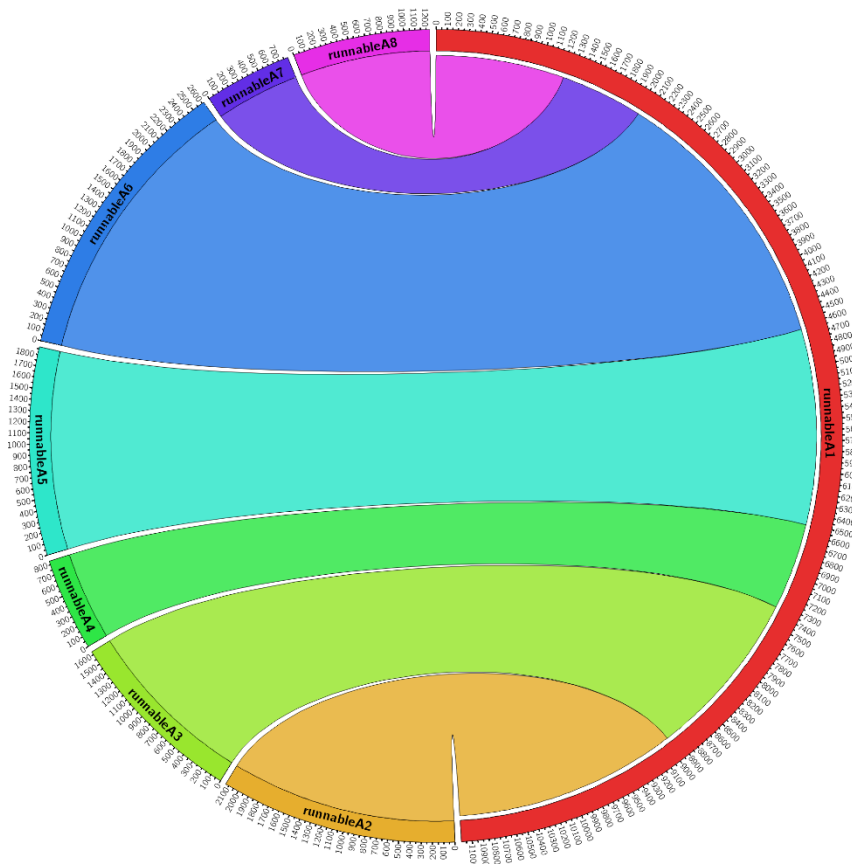
- RapiTask Updated for Multi-Core trace view support.
- Easy to jump to identified failure or timing issue in trace visualiser across *all* cores.
- Helpful for identifying unexpected patterns of execution.
- Very useful when used alongside timing for synchronisation primitives, allowing a visualisation of what other cores are doing at stall time.



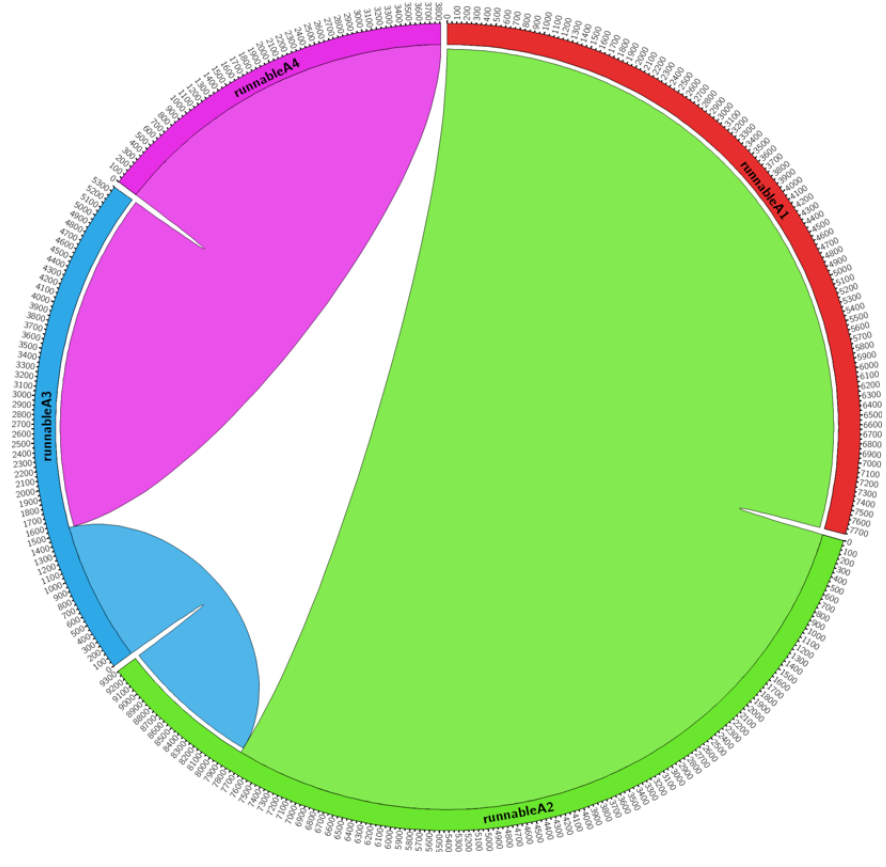


- Code coverage updated to support multi-core execution.
- Including statement, loop, branch and MC/DC coverage.
- New multiple core view, allowing users to see which statements executed on which core.
- Full coverage reports for each core used in the application.

- Used for analysing sequential code when attempting to parallelise legacy applications.
- Identifies 'dependencies' between elements of code based on global variables.
- Read after Write and Write after Read dependencies.
- Aggregate elements of code can be identified (e.g. AUTOSAR runnables).
- XML and experimental CIRCOS output ...



Producer/Consumer



Pipeline

- Intended to be used as part of equivalency checking between sequential and parallelised applications.
- Constraint/equivalency checking in RapiCheck is highly configurable.
- Used to prove that a particular set of desired behaviour in a program held at execution time.
- Detailed reports of failures that occurred.
- Now extended to support multi-core applications.

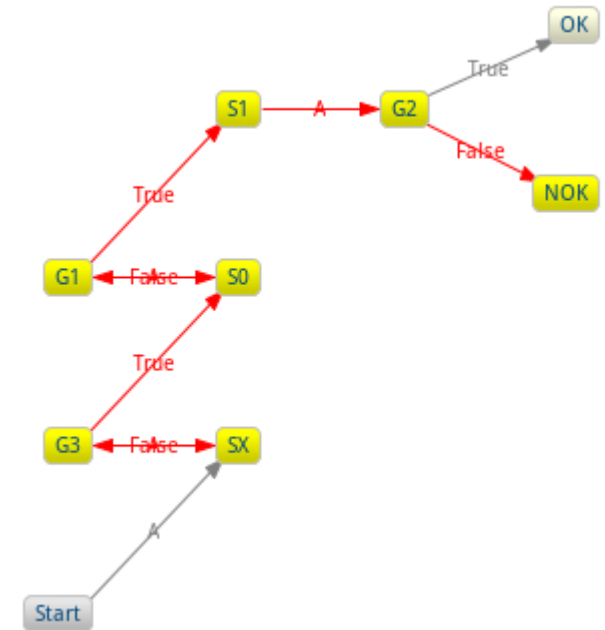
▼ Constraints

Overview of the constraints in the report

Constraint >>			Pass / Fail >>	
Name	Constraint Type	High Level Definition	Pass %	
- thread_start_bound	timing		100.000%	
- count_iterations	timing		-	
- check_iteration_counts	timing		0%	

▼ Failure

Test #	Symbols	Description	To State	Timestamp	Duration
- 1	.AAAAAAAAAAAAAAAAAAAA	-	-	556,289	149,950



- Extension of barrier WCET analysis to support more dynamic and complex barrier usage.
- Further reduction of pessimism.
- More pre-built multi-core focused constraints to be provided with RapiCheck.
- Further extension of Rapita Dependency Tool.