Thimble: Design-time Analysis of Multi-threaded System Behavior

11th Annual High Performance Embedded Computing Workshop
September 2007

Daniel Waddington
Advanced Technology Laboratories
email: dwadding@atl.lmco.com

LOCKHEED MARTIN

Outline

- **Motivation**
- **Solution & Benefits**
- **Tool Overview**
  - Reverse Engineering & Model Generation
  - Systematic Model Execution
  - Behavioral Analysis
  - Data Presentation
- **Currently Supported Design Metrics**
  - Examples
- **Status**
- **Further Work**

Motivation

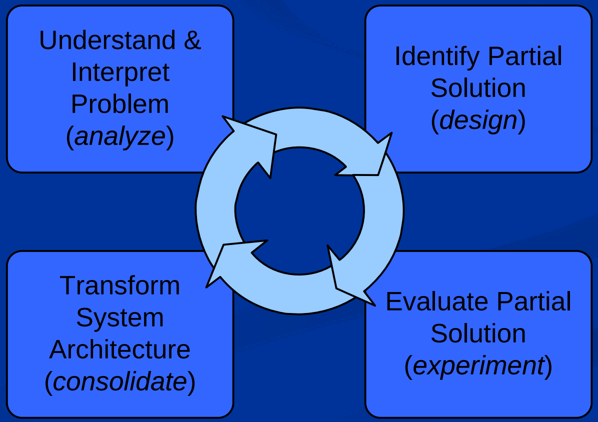
- **Parallel processing is feeding this decades demand for increased performance – commodity processors are increasingly multi-core**
  - CMP, CBE, GPU
- **Software for these new platforms must be explicitly designed to be concurrent**
  - Parallelizing compilers are typically limited to fine-grained parallelism (e.g., loop unrolling)
  - Multi-threaded programming is today's principal approach to implementing concurrency
- **Understanding good and bad design (with respect to concurrency) is inherently difficult**
  - No experimental feedback
- **In large-scale systems development, the ramifications of design decisions are often not understood until late in the development cycle (testing and integration)**

Solution & Benefits

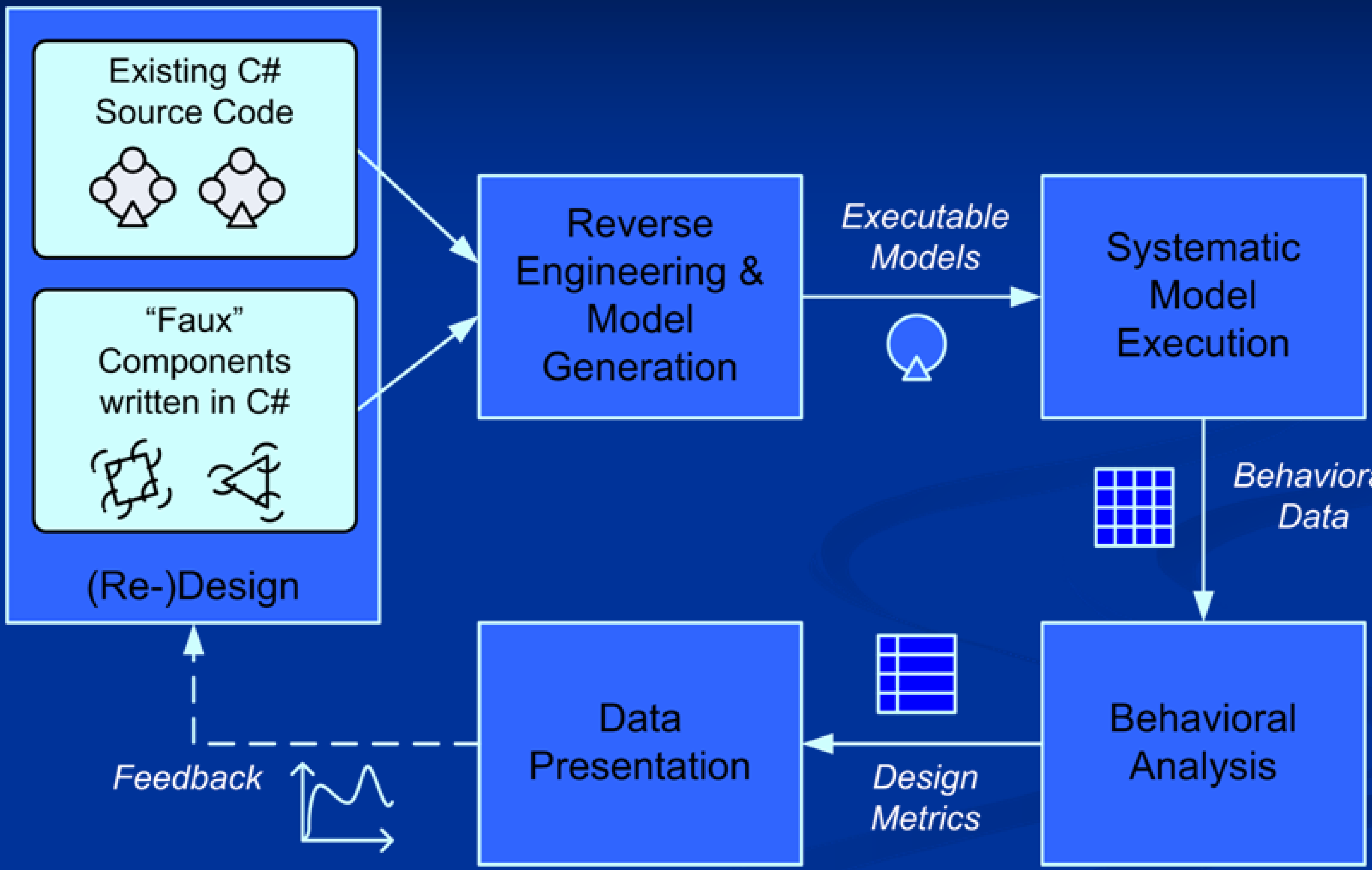
- **Provide tools (Thimble) that will allow multi-threaded systems designers and developers to rapidly explore the design space (with respect to concurrency and synchronization) and understand the ramifications of design decisions**
  - Are threads contending? How much contention exists?
  - Are the cores saturated over time? Will increasing the number of cores lead to increased performance?

- **Thimble will enable rapid evaluation of design decisions and selection of effective architecture early in the development cycle**
  - Help optimize performance and avoid late-stage integration problems

Software and systems development is a ***wicked*** problem that demands rapid fluctuation between problem and solution understanding.



Thimble Tool Overview



Reverse Engineering & Model Generation

- **Reverse engineering : structured interpretation of existing code**
  - Existing C# source code is parsed
    - *custom built parser implemented in ANTLR*
  - Symbol tables, scope relationships, etc., are build from the ASTs
    - *custom program analysis engine written in Stratego* functional programming language*
  - Visual Studio 2005 project files are interpreted for built dependencies and cross-references
    - *provides a complete program view across compilation units*
- **Model generation : building executable models from program**
  - Program analysis engine constructs executable models that accurately represent the analyzed C# code for specific aspects of concern
  - *Bogor* (a model checking framework from Kansas State University) provides a guarded-transition language for specifying multi-threaded systems
    - *explicit support for thread & lock constructs*
    - *no object-oriented support (other that virtual function tables)*
    - *explicit support for non-deterministic choice*

** E.Visser, "Stratego: A Language for Program Transformation based on Re-writing Strategies", System Description of Stratego, RTA `01, LNCS pp.357-361, Springer Verlag May 2001.*

Example Model Generation

LOCKHEED MARTIN

C# Code

```
public void Start()
{
    ThreadStart ts = new ThreadStart(WorkLoop);
    mActiveThread = new Thread(ts);
    mActiveThread.Start();
}
```

Generated Bogor Model Code

```
function {|ThreeWayActors.Actor.Start.()|}((|ThreeWayActors.Actor|)[|this|])
{
    (|System.Threading.ThreadStart|) ts;

    /* var initializer assignment to new expression Program.cs:67 */
    loc loc0: do { ts := new (|System.Threading.ThreadStart|); } goto loc1;

    /* implicit ctor call for var initializer new expression Program.cs:67 */
    loc loc1:invoke {|Ctor.System.Threading.ThreadStart.(string,System.Object)|}
                    (ts,"{|ThreeWayActors.Actor.WorkLoop.()|}",[|this|])
    goto loc2;

    /* assignment to new expression Program.cs:68 */
    loc loc2: do { [|this|].mActiveThread := new (|System.Threading.Thread|); } goto loc3;

    /* implicit ctor call for new expression statement Program.cs:68 */
    loc loc3: invoke {|Ctor.System.Threading.Thread.(System.Threading.ThreadStart)|}([|this|].mActiveThread,ts)
    goto loc4;

    /* invocation on method via member accessor Program.cs:69 */
    loc loc4: invoke {|System.Threading.Thread.Start.()|}([|this|].mActiveThread) goto loc5;
    return;
}
```

Reverse Engineering & Model Generation

- **Model cut-off points**
  - Bogor models are only generated for "visible" source code; cut-off points define the limits of the modeled system (e.g., invocations on system calls that are not directly concerned with concurrency and synchronization are omitted)
  - System libraries are either:
    - a.) implemented manually in Bogor modeling language
    - b.) left as empty stubs (cut-off points)
    - c.) simulated directly in Java code
- **Thimble models are abstract – only details that are pertinent to synchronization and concurrency are retained**
  - Storage (and persistent data) is not modeled
  - Interaction with the environment must be simulated
- **Challenges of deriving "representative" behavior**
  - Traditionally model-checking performs exhaustive searching of the state space and therefore does not care about time *per se* (only ordering)
  - Thimble must imitate wall-clock time by scaling the number of quanta needed to perform external functions (timings collected from run-time profiling)

Systematic Model Execution

- **Bogor models of the system are model-checked**
  - Model-checking allows controlled state exploration
- **Pluggable search strategies control how state space is explored**
- **Currently implemented strategies**
  - Exhaustive (takes a long time even with partial-order reduction)
  - Random (comparable to simulated execution)
    - *complete execution paths are randomly selected*
  - Pathological
    - *path selection is based on the variance of data on candidate paths; representatives of dissimilar-path groups are searched first*
    - *approach allows worst-case scenarios to be identified*
- **Support for N-core abstract machines**
  - Model-checker effectively simulates an abstract machine
  - Number of cores is selectable through tool
    - *collapsing N scheduling decisions into one*
    - *supporting frame-based scheduling and thread core-affinities*
- **Distributed execution**
  - Model checking can be distributed to multiple nodes (this processing requires a lot of horsepower)

Behavioral Analysis

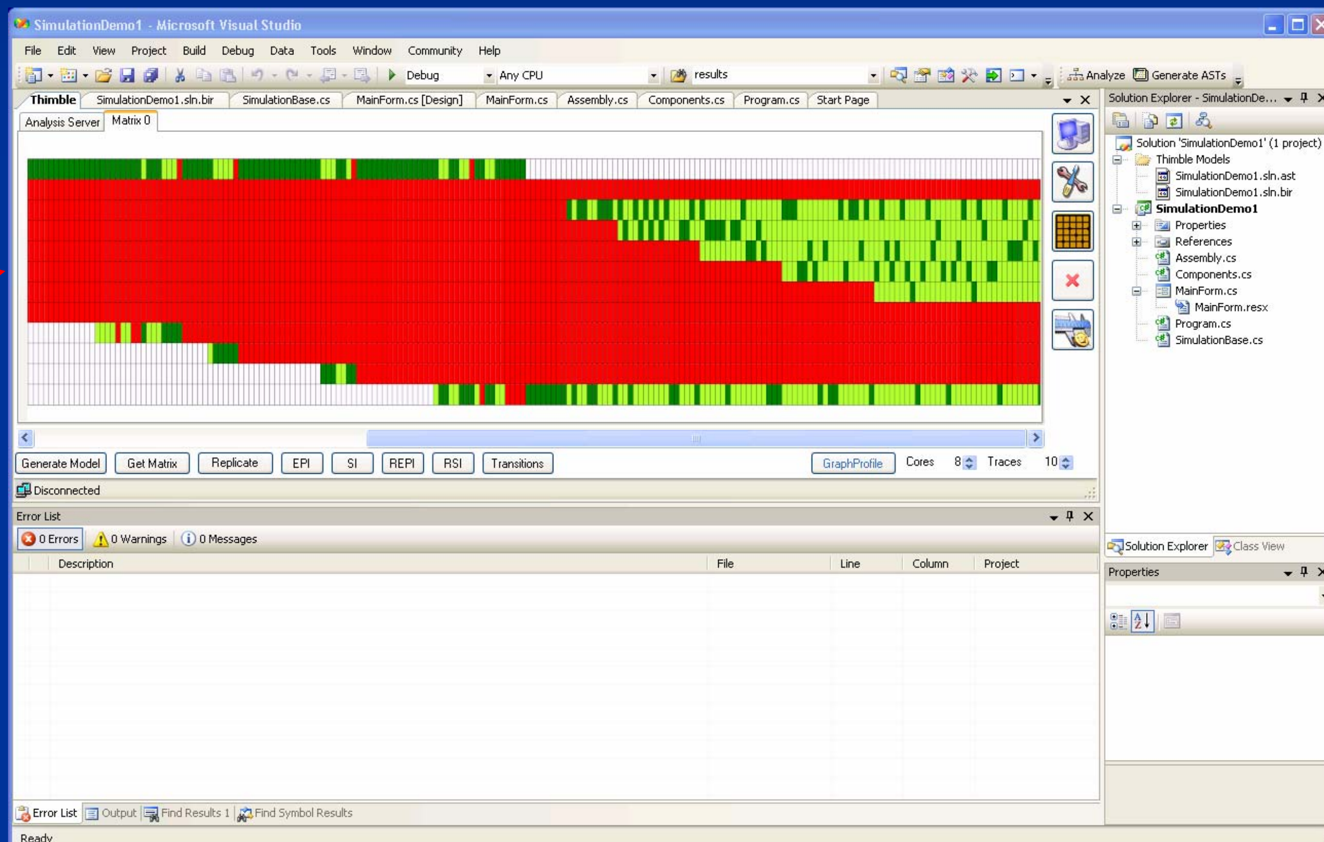
- **Raw data collected from model checker**
  - Scheduling matrices
    - *thread state (running, ready, block, doesn't exist) over time*
    - *one matrix for each inspected inter-leaving (execution path)*
    - *N-core scheduling states collapsed into one*
  - Potentially large amounts of data O(100Mb)
    - *HDF5 data format*

- **Data is distilled in Mathematica**
  - Simple statistical analysis
  - Efficient matrix manipulation (e.g., sum)
  - Powerful analysis libraries (e.g., cluster analysis)
  - Off-the-shelf data visualization

Data Presentation

- **The Thimble front-end is fully integrated into Visual Studio 2005**

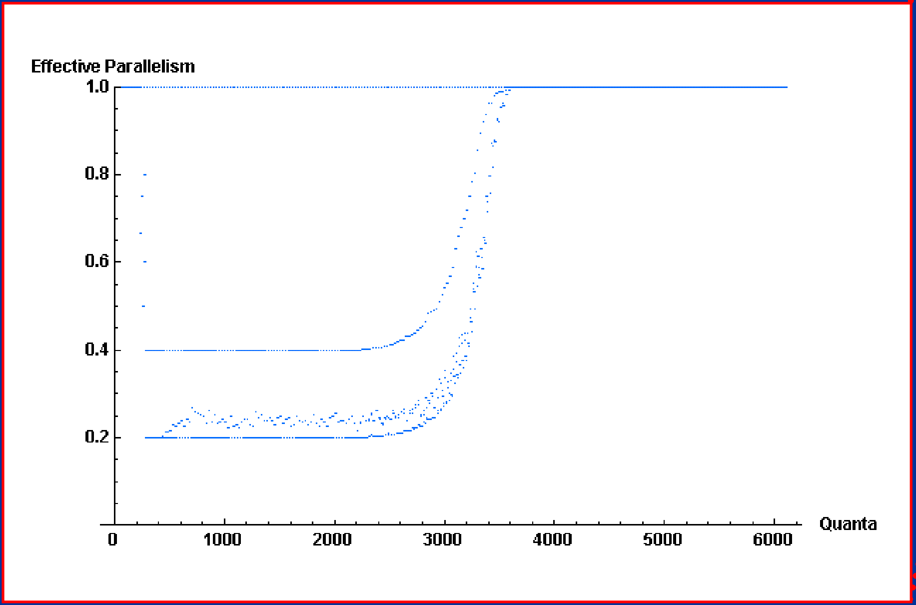
Example scheduling matrix

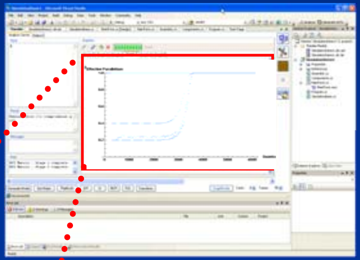


Currently Supported Metrics

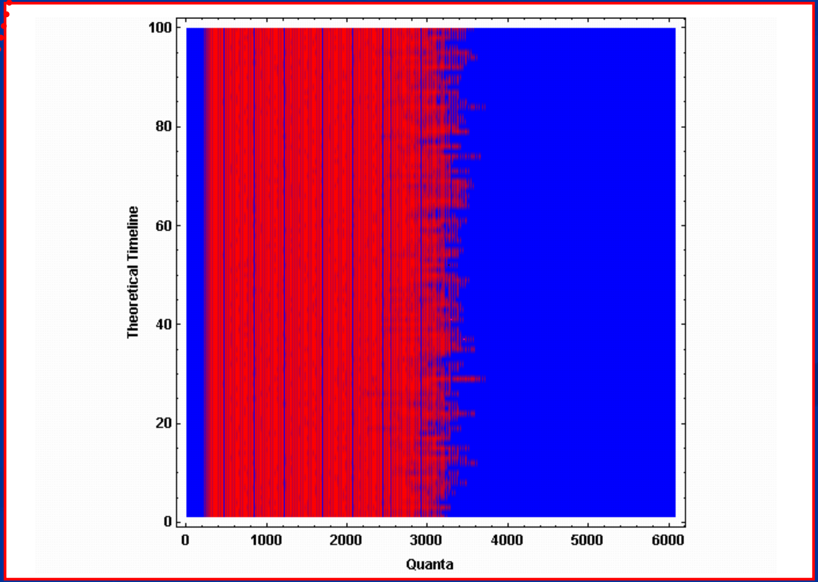
- ***Effective Parallelism Index*** **(EPI) – over time, how many of the threads that have been created are able to perform work concurrently**

100 Execution Samples for 5-Threaded System (4 Active, 4 Passive) Matrix Work





Raster images allow variance across potential executions to be quickly assessed

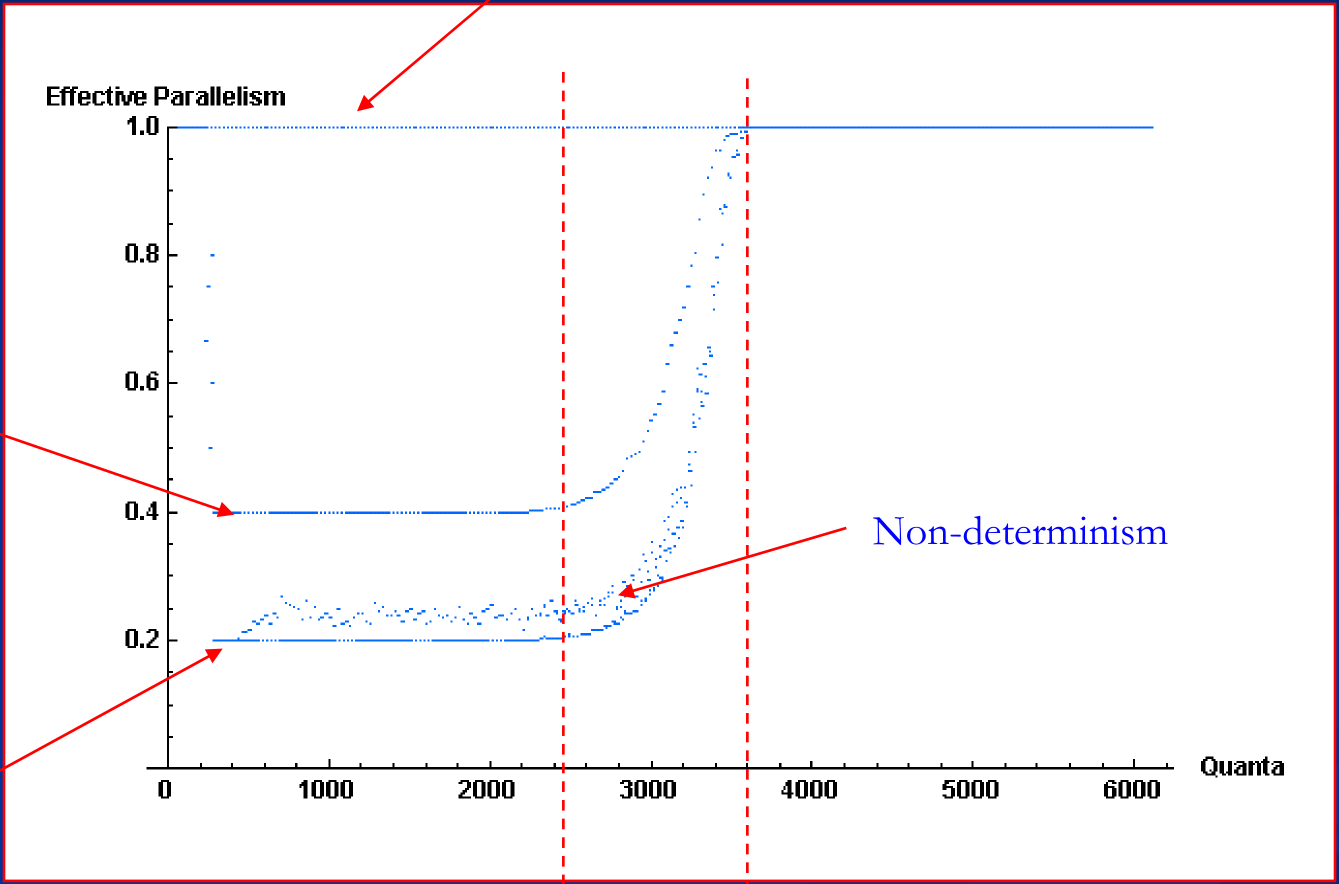


Example Graph: Interpreting EPI Graphs

Handovers

Handover overlaps (two threads runnable)

Single thread holding resources



Non-determinism

Single thread (zero contention)

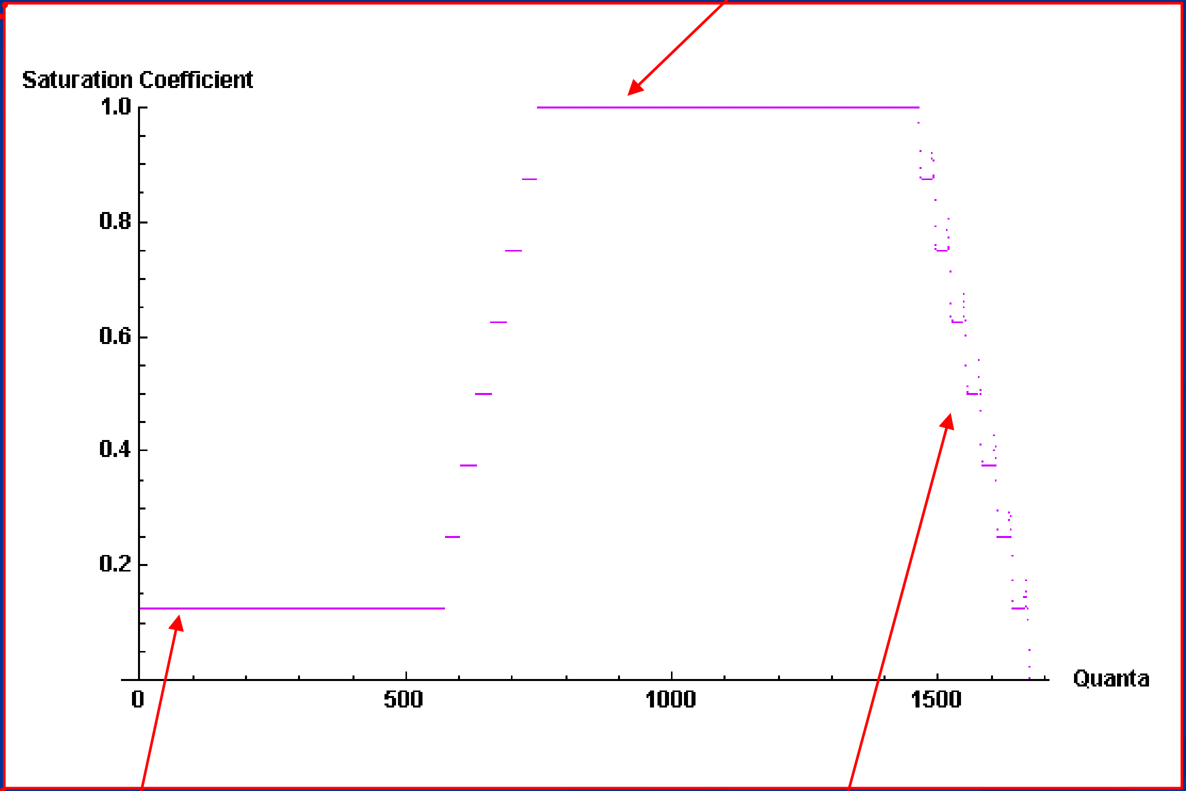
Threads begin to complete

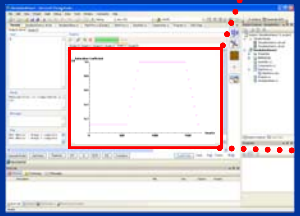
Currently Supported Metrics

- ***Saturation Index*** **(SI) – shows how threads that have been created induce load on the system**

All cores saturated

Saturation Index calculated for 8 cores and an 8 actor system.





1 thread running at this point

Threads completing

Status

- **Project started January 2006 as part of the Lockheed Martin Software Technology Initiative (STI)**

- **Team**
  - Lockheed Martin ATL
  - Kansas State University (Prof. John Hatcliff & Prof. Robby)
  - Vanderbilt (Prof. Doug Schmidt)

- **Proof-of-concept prototype implementation expected to completed by EOY 2007**

- **Current status**
  - 70% C# version 2.0 supported
  - Only supports round-robin scheduler (systems with multi-priority threads are not currently accurately modeled)
  - Support for random and exhaustive searching (pathological in development)
  - MDD-tool in development

Further Work

- **Technology piloting**
  - Deployment of tool on Lockheed Martin Astraeus test bed (1Q08)
    - *experimental facility to allow evaluation of different multi-core platforms*
  - Piloting tools with LM IS&GS Horizon satellite ground station framework
    - *partnering with sponsor of work*

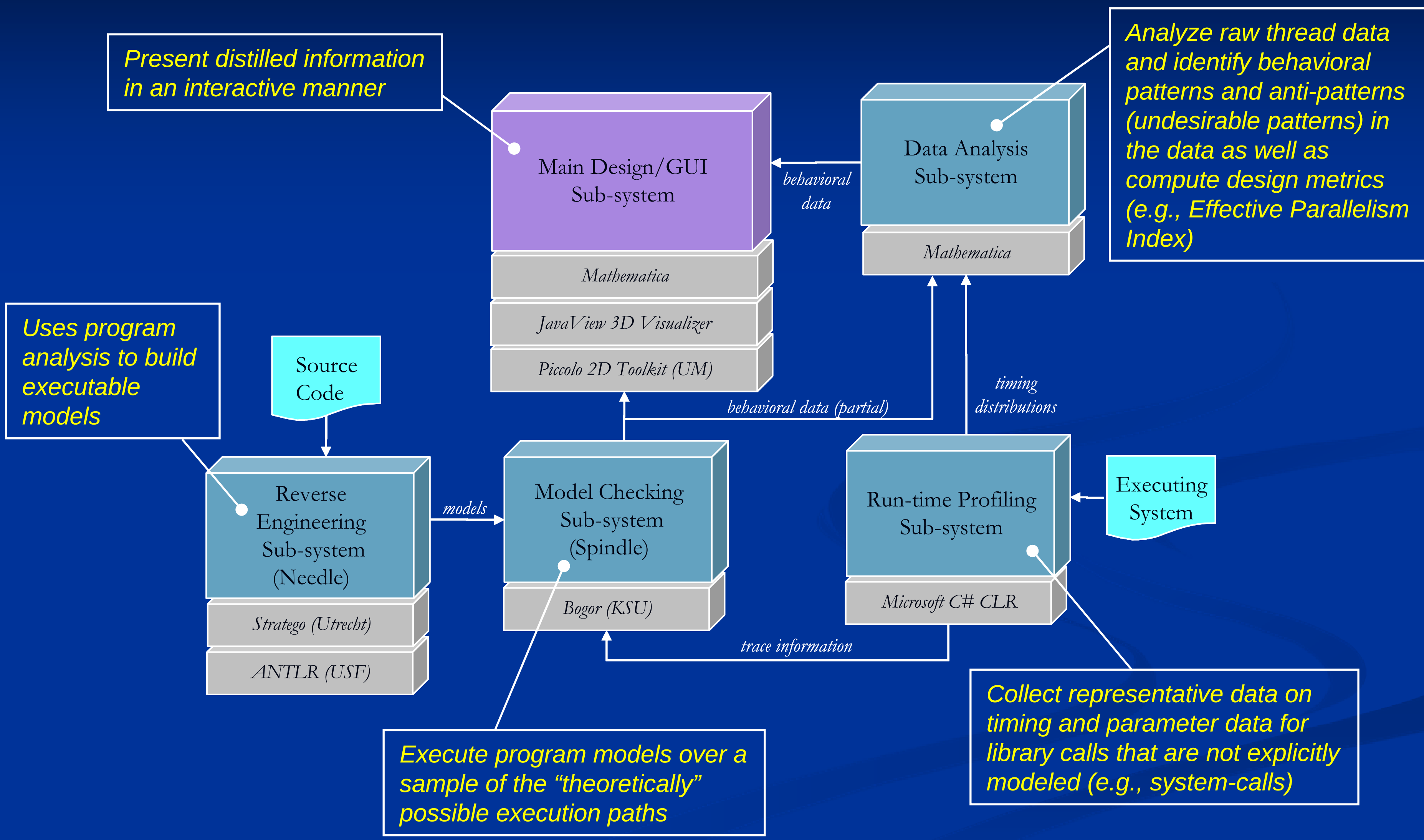
- **Possible future avenues**
  - Coupling with Model-Driven Development tools (domain specific models of execution and concurrency supporting round-trip engineering)
  - Extensions to support Java
    - *consider a subset of C# language features*
  - Support for behavioral data collection from actual execution – modification of OS kernel scheduler to collect scheduling matrices.
    - *allow experimental quantification of model accuracy*
  - Support for multiple task schedulers beyond round-robin
    - *e.g., simulation of dynamic priority queues, RMS, EDF*
  - Isolation and selection of execution segments to support larger code bases

- **Extension of existing design metrics**
  - Thread Coupling Index – to quantify inter-dependencies across threads
  - Logical flow analysis to help identify hidden causal chains

Questions?

Backup Slides

Thimble Solution Architecture



Key Innovations

1. **Use of model-checking in a sampling mode to mine representative behavior patterns**
2. **Integration of behavioral signatures collected from run-time profiling with statically derived models**
3. **Definition of design metrics that can be used to formally quantify the behavior of a program with respect to concurrent execution**
   - *Effective Parallelism Index (EPI)* – how effectively threads are being used; indirectly gives a measure of lock-step caused by blocking
   - *Saturation Index (SI)* – actual versus potential processor utilization over time
   - *Thread Coupling (TC)* – measure of level of cross-thread dependencies
4. **Modification of abstract machine to perform "what-if" analyses for future N-way architectures**