Overview
We show a technique for creating and using an executable MPI runtime prototype generated by a semantics engineering tool inside an algorithm used by ISP, a state-space exploration testing tool for MPI programs.

Motivation
Here is a snippet from a program that implements the Jacobi algorithm for PDEs using MPI and CUDA:

```c
void RunJacobi(...) {
    ...
    while (...) {
        residue = CallJacobiKernel(...);
        ...
        MPI_Allreduce(...);
    }
    ...
    MPI_Reduce(...);
}
```

Analyzing the execution of such a program is difficult:
- hundreds of interacting components
- complicated semantics and nondeterminism

How does one verify such a program is bug free?

Accurate, fine-grained model checking and active testing tools for such programs are more complex: The tool needs the runtime semantics to check for
- deadlocks
- data races
- buffer overflows

ISP² is an active testing tool for exploring non-equivalent execution paths of an MPI program. A simple example is shown at the right.

ISP needs the semantics of the MPI runtime in some of its algorithms.

**Problem:** How can we accurately model the MPI runtime and use the model in an ISP algorithm?

Design
We used the K framework for the MPI runtime model and the ISP algorithm.

The MPI runtime model is responsible for
- parsing the MPI program
- determining which actions are possible
- carrying out a chosen action
- reporting whether it’s deadlocked

The ISP module explores the possible execution paths of the MPI program using the executable MPI model. The entire ISP-MPI package can be executed using K.

Rule Example
This rule describes the non-overtaking requirement for MPI sends:

```
rule (instruction, (send, Dest1:Int),
    CallingProc1:Int, PC1:Int)
    overtakes
    (instruction, (send, Dest2:Int),
    CallingProc2:Int, PC2:Int)
=>
    (CallingProc1 ==Int CallingProc2) andBool
    (Dest1 ==Int Dest2) andBool
    (PC1 >Int PC2)
```

Observations
- Found the model was accurate but executed slowly. Could be due to our inexperience: One rule we wrote initially lead to exponential evaluation behavior! A simple change fixed the problem.
- Initially tried representing the state of the MPI runtime using K ‘configurations’, but found it easier for our design to use syntactic lists, maps, and other structures, with a LISP style (see above example).
- With K configurations, we found it difficult to ◊ calculate all possible transitions the MPI model could take and use this -inside- the model ◊ eliminate unwanted nondeterminism

Conclusions & Related Work
Tools for describing the complicated runtime semantics of high performance systems will continue to be valuable to designers, programmers, and analysts.

Morse and colleagues demonstrated a similar technique for MCAPI programs using PLT-Redex⁵.

References