SmartSpice HPP: High Performance Parallel with SPICE Accuracy

1. Introduction

As technology advances, the complexity of circuit designs is continuously growing, whereas the design cycles become even shorter. Consequently, circuit simulation can easily become the bottleneck for design verification. An analog simulator, therefore, must deal with a larger number of advanced devices, while still maintaining the same level of accuracy for a given simulation time.

In order to cope with this increasing pressure on the simulator's performance, SmartSpice has introduced a new simulation engine: HPP (High Performance Parallel). SmartSpice HPP takes advantage of the modern multi-core hardware platforms to speed up all internal aspects of transient simulations of analog circuits.

One of the main differences of SmartSpice in HPP mode is its partition-based simulation, where partitions will be processed in parallel and the matrix loading stage, linear solver operations, as well as general transient simulation steps are sped up. Further improvements can also be achieved when the block isomorphism and block latency features are enabled. Finally, one can extract the most out of SmartSpice HPP for post-layout simulation when it is used in conjunction with Jivaro for parasitic reduction.

In this application note, we describe SmartSpice HPP, how it works, and how to use it in order to provide fast and accurate transient simulations on a variety of designs, from medium- to large-size circuits, either pre- or post-layout. We also demonstrate in this application note multiple usage modes, so one can see that SmartSpice HPP can not only be up to 8x more scalable than regular SmartSpice, but it can also be up to 40x faster, while still keeping acceptable accuracy.

The rest of this document is organized as follows. In Section 2, we introduce the concept of Bordered Block-Diagonal form, an important topic to better understand SmartSpice HPP and its advantages. Section 3 introduces HPP's circuit partitioning approaches, whereas Section 4 focuses on its matrix partitioning strategies. In Section 5, we present two other important features from HPP: block isomorphism and block latency. Section 6 briefly introduces Jivaro as an RCL-reduction engine to speed up simulation time in SmartSpice. In Section 7, we present how one can use SmartSpice HPP and all the features presented in this application note. Section 8 presents some experimental results, whereas Section 9 presents our final remarks. A quick summary is presented in Section 10.

2. Bordered Block-Diagonal (BBD) Form

In order to enable parallelizable methods to solve linear systems, one can reorganize the system matrices into a Bordered Block-Diagonal (BBD) form. Figure 1 illustrates how a Global Matrix can be reordered in a BBD form.

To enable the desired parallelization, the BBD form follows a divide-and-conquer approach. For this, from a "divide" perspective, it splits the problem of solving a large matrix into smaller submatrices that can be solved independently. Then, from a "conquer" perspective, it relies on linking these submatrices together by using the so called "border matrices", which are used to complete the solution of the original large matrix.

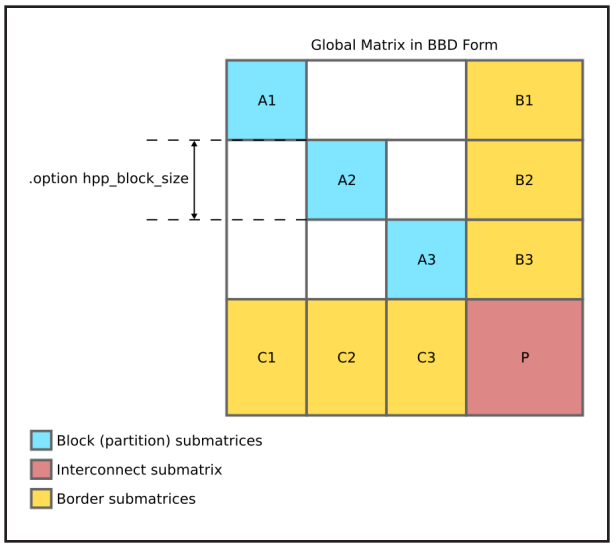

Figure 1. A Global Matrix represented in BBD Form.

From a SPICE simulation point of view, the global matrix is reorganized into submatrices representing circuit partitions (A1-An in Fig. 1), its interconnections (P in Fig.1), and border submatrices (B1-Bn and C1-Cn in Fig. 1).

In the next sections, you can understand how SmartSpice HPP takes advantage of BBD forms to speed up simulation time.

3. Circuit Partitioning on SmartSpice HPP

A distinguishing characteristic of SmartSpice HPP is its partition-based simulation. Circuit partitioning is currently available through two different methods:

1. **Graph circuit partitioning**: topology-oriented partitioning, automatically computed based on the circuit topology
2. **Hierarchical circuit partitioning:** topology-oriented partitioning, guided by the circuit's user-defined hierarchy.

Notice that we are referring to circuit partitioning (which is tied to its physical interpretation), as opposed to matrix partitioning (which is tied to its mathematical interpretation). Even so, SmartSpice HPP performs circuit partitioning (either hierarchical- or graph-based) in a BBD-like structure. In this case, each partition will be processed separately by the HPP engine and can speed up the matrix loading stage, linear solver operations, as well as general transient simulation steps.

4. Matrix Partitioning on SmartSpice HPP

In SmartSpice HPP, the user can also control which matrix partitioning approach is used. Two partitioning options are supported on SmartSpice HPP:

1. **Metis-based matrix partitioning**: the solver activates the Metis nested dissection algorithm on its preordering step and uses the information from Metis to create a suitable matrix in BBD form
2. **Circuit-based matrix partitioning:** the solver reuses the information about the BBD structure from the circuit partitioning, if available, to create a matrix in BBD form.

5. Block Isomorphism and Block Latency

SmartSpice HPP can also take advantage of block isomorphism and block latency to speed up simulation. These two techniques are briefly explained in the following.

Multiple blocks are said to be isomorphic if they have the same size, internal states, and external stimuli, considering the given tolerances. During a transient simulation, the HPP engine dynamically checks the potentially isomorphic blocks if this feature is enabled. Then, for each group of isomorphic blocks, only one block is simulated, and its results are reused to its other isomorphic blocks.

A block is said to be latent if its internal states stay within the given tolerances for some consecutive iterations during a transient analysis. If a block is found to be latent for the time point being calculated, this block will not be recomputed. Instead, its internal data is reused from the previous time point.

6. SmartSpice HPP Along with Jivaro

When running post-layout simulation, another way to best make use of SmartSpice HPP and speed up simulation time while keeping accuracy under control is to employ Jivaro, provided with SmartSpice, for parasitic reduction. In this case, even if the user does not have access to Jivaro as a standalone tool, the patented mathematical approaches from the Jivaro engine sit inside SmartSpice and perform Model Order Reduction (MOR) to reduce the parasitic complexity while preserving high accuracy.

7. How to use SmartSpice HPP

SmartSpice HPP can be enabled by running "smartspice -hpp". By default, other general-purpose SPICE options are also set from this flag, as follows:

- hsimspeed=3
- bypass=2
- expbypass=1e-3
- cflflag=1

SmartSpice HPP is available on both Windows and Linux. However, as a general limitation, it only supports transient analyses. Also, only a limited number of SPICE models are supported. (A list of supported models in HPP mode can be found in SmartSpice User Manual.)

In the following, you will find more details about solver compatibility in SmartSpice HPP, as well as how to use circuit partitioning, matrix partitioning, isomorphism, latency, and Jivaro reduction.

Solver Compatibility in SmartSpice HPP

SmartSpice HPP supports all the solvers currently implemented in SmartSpice, namely XMS, SPEEDS, and SPS. However, in order to extract the most out of the HPP functionality, we strongly recommend the SPS solver to be used. This can be achieved either by using a netlist option:

.option solver = sps

or by a command-line argument (which forces the SPS solver and overrides any ".option solver" statement set in the netlist):

-forcesolver sps

The SPS solver is required for the following features: matrix partitioning, isomorphism, and latency. Other solvers do not currently support these features.

Using Circuit Partitioning

In SmartSpice HPP, graph partitioning is set by default. If the user would rather use hierarchical partitioning, then ".option user_hier_level" needs to be set. The list of options relevant for each partitioning mode can be found in Table 1. The referred circuit partitioning modes will only work if the following lists of requirements are met.

Graph partitioning requirements:

- None

Hierarchical partitioning requirements:

- User-defined hierarchy in the circuit

Using Matrix Partitioning

In SmartSpice HPP, Metis-based matrix partitioning is set by default. In order to control matrix partitioning, ".option matrix_partitioning_mode" should be used, as follows:

- **.option matrix_partitioning_mode=0**

  Enables Circuit-based matrix partitioning. (Only this mode supports isomorphism and latency.)

- **.option matrix_partitioning_mode=2 (default)**

  Explicitly enables Metis-based matrix partitioning.

- **.option matrix_partitioning_mode=3**

  Completely disables matrix partitioning and builds one single global matrix. (It can still use multiple threads.)

Notice that in the Metis-based matrix partitioning, the interconnect matrix created by the solver can be substantially smaller than the one provided by the circuit-based matrix partitioning, so performance could be better. However, Metis-based partitioning does not support isomorphism and latency.

The referred matrix partition modes will only work if the following list of requirements is met.

Matrix partitioning requirements:

- SPS Solver

Using Isomorphism and Latency

Block isomorphism is only available in HPP if the hierarchical circuit partitioning and solver SPS are used. Isomorphism is not supported in the graph partitioning approach. Table 2 summarizes the list of options that are applicable for isomorphism and latency. The list of requirements is presented in the following.

Option Name	*Option Description*
Graph Partitioning Options	
hpp_partition_count=<N>	Sets desired partitions count to <N> for graph partitioning. By default, the number of partitions is determined automatically for better performance.
Hierarchical Partitioning Options	
user_hier_level=<L>	Specifies the hierarchy level to split the partitioning into interconnect and block partitions. All the cells from top level to level <L-1>, will compose the interconnect partition (P in Fig. 1). All the cells from level <L> on, will compose the block partitions (A1-An in Fig. 1). If not defined, an auto detection algorithm will calculate <L> to minimize interconnect network and maximize number of separate blocks.
hpp_block_size=<S>	Specifies the desired size in terms of nodes of block partitions (A1-An in Fig. 1). The default value is 100.
hpp_hiersim	Enables automatic detection of array blocks (useful for memory designs, flat-panel displays, etc.).

Table 1. The list of options that are applicable for Graph- and Hierarchical-Based circuit partitioning.

Option Name	*Alias Name*	*Option Description*
Isomorphism Options		
hpp_block_isomorphism	iso	Activates isomorphic blocks detection algorithm.
hpp_block_isomorphism_reltol	iso_reltol	Isomorphism relative voltage error tolerance for blocks. Default is 1e-3.
hpp_block_isomorphism_abstol	iso_abstol	Isomorphism absolute current error tolerance for blocks. Default is 1e-9.
hpp_block_isomorphism_vntol	iso_vntol	Isomorphism absolute voltage error tolerance for blocks. Default is 5.e-5.
Latency Options		
hpp_block_latency	lat	Activates latent blocks detection algorithm.
hpp_block_latency_reltol	lat_reltol	Latency relative voltage error tolerance for blocks. Default is 1e-3.
hpp_block_latency_abstol	lat_abstol	Latency absolute current error tolerance for blocks. Default is 1e-9.
hpp_block_latency_vntol	lat_ vntol	Latency absolute voltage error tolerance for blocks. Default is 5.e-5.

Table 2. The list of options that are applicable for Isomorphism and Latency.

Isomorphism requirements:

- Solver SPS
- Hierarchical circuit partitioning
- Circuit-based matrix partitioning

Latency requirements:

- Solver SPS
- Either graph or hierarchical circuit partitioning
- Circuit-based matrix partitioning

Using Jivaro Reduction

In SmartSpice HPP, no parasitic reduction techniques are set by default. If the user is running post-layout simulation, we strongly recommend Jivaro for parasitic reduction be configured from inside SmartSpice. Table 3 summarizes the list of options that are applicable for Jivaro reduction. The list of requirements is presented in the following.

Jivaro reduction requirements:

- None

8. Experimental Results

In order to demonstrate the benefits of adopting SmartSpice HPP on a simulation flow, we present in this section some results from experimenting it on a design. For this, we used a post-layout, synchronous 2MB (256 64-bit words) SRAM, designed in a 45nm technology based on BSIM4 models. We simulate common timing and power measurements using the entire memory design, i.e., we did not apply any design optimization to reduce the circuit only to its critical path. The experiments were performed on a server with an Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz.

Experimental Setup

For these experiments, we exercised the characterization of a single memory cell from the entire design. The intention was to illustrate how SmartSpice HPP can be used when a designer is trying to investigate both timing and power behaviors of potentially critical paths from a memory design.

For this, we performed a transient analysis, as follows (Pi should be interpreted as the i-th period of interest):

P0: Bring the target cell to a known state by writing a 1 logic to it

P1: Write a 0 logic to the target cell

P2: Clean the buses by writing a 1 logic to a different memory cell

P3: Try to read from the target cell and confirm if the 0 logic was successfully written

P4: Clean the buses with an idle period

P5: Write a 1 logic to the target cell

P6: Clean the buses by writing a 0 logic to a different memory cell

P7: Try to read from the target cell and confirm if the 1 logic was successfully written

Then, once the simulation is finished, we performed the following post-processing measurements:

delay_hl: measure on P3 the clock-to-output delay (50% to 50%), when the target signal is transitioning from the 1 logic to the 0 logic

delay_lh: measure on P7 the clock-to-output delay (50% to 50%), when the target signal is transitioning from the 0 logic to the 1 logic

Option Name	*Option Description*
rcl_reduction_type=val	Selects RCL netlist reduction engine. 0 - Reduction disabled (default) 2 - “Jivaro”
rcl_reduction_reltol=val	Specifies maximal relative error for transient responses of reduced and original circuits. (Default: 0.05.)
rcl_reduction_max_freq=val	Defines frequency range for which RCL reduction guarantees consistency of transient responses of reduced and original circuits. The default is 5.0e9 Hz.
rcl_reduction_flow_mode_hpp=val	Controls RCL-reduction in Graph-based circuit partitioning. 0 - Reduction before graph partitioning 1 - Reduction after graph partitioning (default)
rcl_reduction_minc=val	Defines a threshold for filtering capacitor values. All capacitor values less than <val> are replaced by an open circuit. Default is -1 (inactive).
rcl_reduction_minr=val	Defines a threshold for filtering resistor values. All resistor values less than <val> are replaced by short circuit. Default is 1E-15.
rcl_reduction_maxr=val	Defines a threshold for filtering Resistor values. All resistor values greater than <val> are replaced by open circuit. Default is 1E15.

Table 3. The list of options that are applicable for Jivaro reduction.

slew_hl: measure on P3 the slew rate (90% to 10%) on the output, when the target signal is transitioning from the 1 logic to the 0 logic

slew_lh: measure on P7 the slew rate (10% to 90%) on the output, when the target signal is transitioning from the 0 logic to the 1 logic

write0_pw: measure the average power from P1 to P2, where the 0 logic is being written to the target cell

write1_pw: measure the average power from P5 to P6, where the 1 logic is being written to the target cell

read0_pw: measure the average power from P3 to P4, where the 0 logic is being read from the target cell

read1_pw: measure the average power from P3 to P4, where the 0 logic is being read from the target cell

Finally, we performed some comparisons between regular SmartSpice using its default settings against SmartSpice HPP default settings using SPS solver, graph circuit partitioning, and Metis-based matrix partitioning. Neither block isomorphism nor block latency were used.

Scalability Analysis

In these experiments, we were investigating how much more scalable SmartSpice HPP can be when compared to regular SmartSpice. For this, we observed regular SmartSpice scalability by running it with multiple threads ("-P <n>" command-line argument, where <n> is the number of threads) and comparing its performance against its single-thread run. Then, we did the same with SmartSpice HPP and extracted the speedup curve of both simulators. The results are presented in Fig. 3.

As one can see, SmartSpice HPP with 16 threads can be 13.3x faster than SmartSpice HPP single thread. This shows that it is up to 8x more scalable than regular SmartSpice, which was only 5.3x faster with 16 threads compared against its single-thread run.

These results demonstrate how HPP algorithms and its simulation strategy can take advantage of the modern multicore hardware platforms to speed up all internal aspects of transient simulations of analog circuits.

Performance and Accuracy Analyses

In these experiments, we were investigating how much faster SmartSpice HPP can be when compared to regular SmartSpice, as well as its impact in terms of accuracy. For this, we experimented SmartSpice HPP along with two other compatible features: (1) Turbo mode, which plays with the runtime/accuracy trade-off and (2) Jivaro, for RCL reduction. All simulations were run with 16 threads (-P 16). The obtained results are presented in Fig. 4.

As it can be seen, just by enabling HPP, the simulation can already be about 5x faster with about 2% maximum error for slew measurements, less than 0.1% for delay measurements, and less than 1% for power measurements. By bringing Jivaro in, the speedup goes up to more than

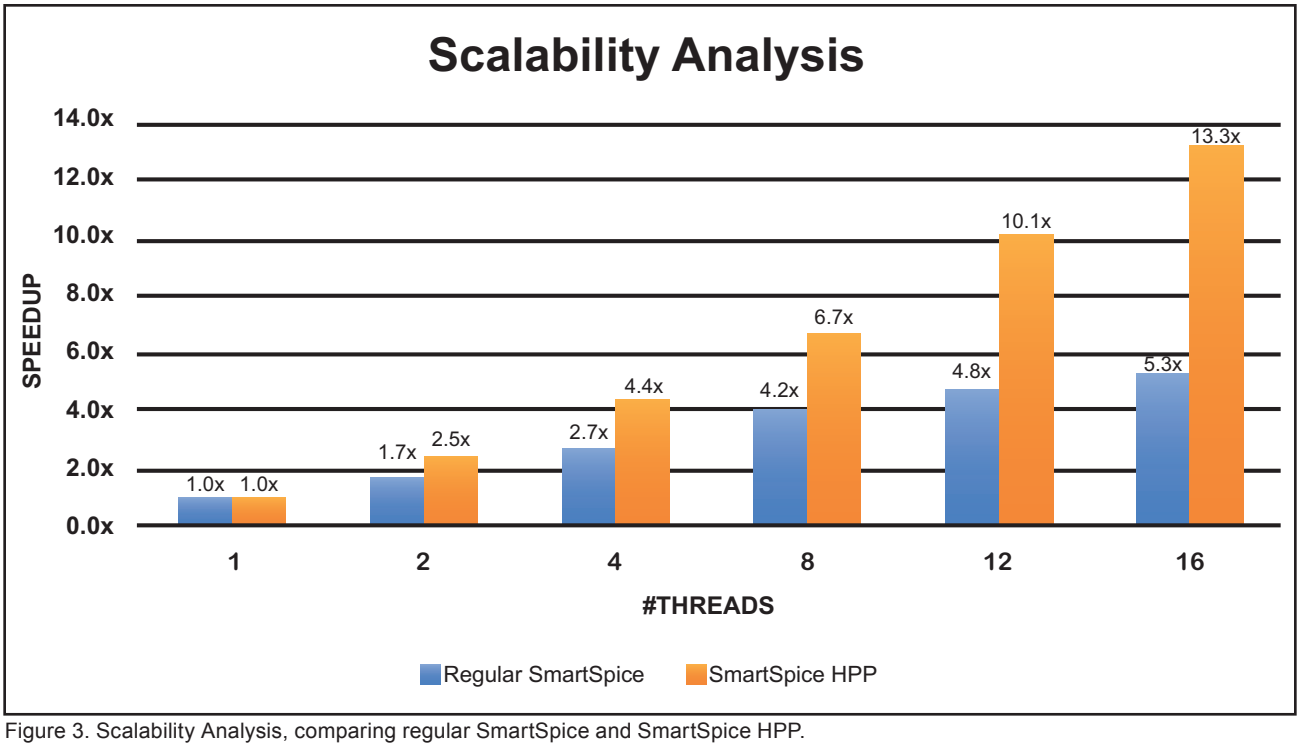

Figure 3. Scalability Analysis, comparing regular SmartSpice and SmartSpice HPP.

8x, while maintaining nearly identical accuracy. Finally, by bringing both Jivaro and Turbo together, HPP can be about 40x faster than regular SmartSpice, whereas the maximum errors for the measurements are kept with less than 9% for slew figures, around 2% for delay measurements, and still less than 1% for power figures.

These results demonstrate how effectively SmartSpice HPP can integrate with other SmartSpice features (e.g. Turbo mode and Jivaro RCL reduction). It also shows how much faster HPP can be while keeping an acceptable accuracy for timing measurements and very high accuracy for power measurements. Using SmartSpice HPP brought the simulation time from around 90 hours down to 2.2 hours. This represents a dramatic increase in productivity from a designer perspective.

Notice that, with power becoming one of the most important characteristics for circuit designers, followed by delay measurements, having a circuit simulator that can deliver 40x speedup with highly accurate power and delay figures can dramatically increase designers' productivity. This is something that FastSPICE engines cannot deliver.

9. Final Remarks

In this section, we present a few recommendations for how one can better use SmartSpice HPP. Find more details in the following subsections.

Circuit Partitioning Recommendations:

A. Hierarchical structure of the circuit is unknown

1) Make sure you are using Graph circuit partitioning. For this, DO NOT use ".option user_hier_level".
2) Use ".option hpp_partition_count" to control the number of partitions.
3) Try ".option hpp_block_latency" to increase speed of model evaluation.

B. Circuit hierarchy is known and contains many cells with the same size (e.g. TFT, SRAM)

1) Make sure you are using Hierarchical circuit partitioning. For this, DO use ".option user_hier_level".
2) You can try ".option user_hier_level=<n>" to set the hierarchy level, but SmartSpice HPP can also handle it automatically.
3) ".option hpp_block_size" might help to protect simulation from very small blocks.
4) As ".option hpp_hiersim" provides automatic memory arrays detection, it may help to speed up simulation time.
5) Using ".option hpp_block_isomorphism" and ".option hpp_block_latency" may help to speed up the model evaluation.

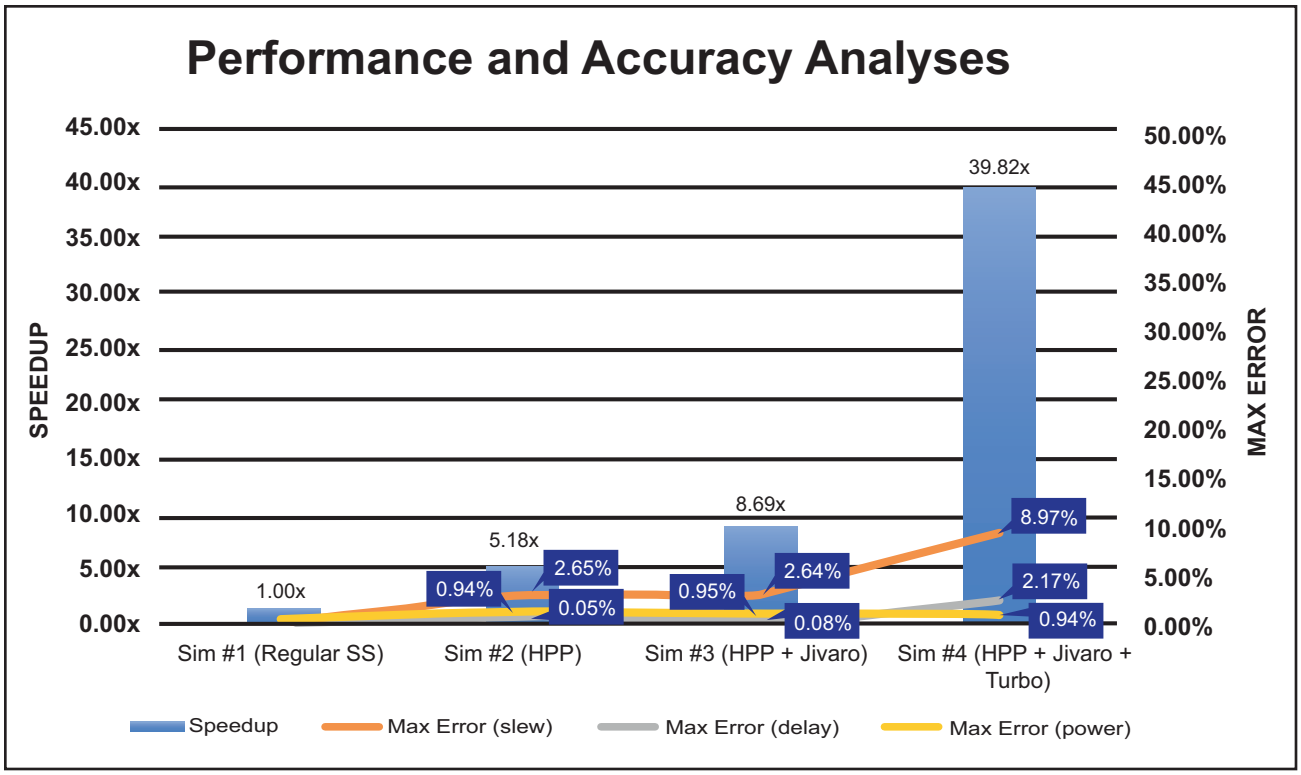

Figure 4. Performance and Accuracy analysis, comparing regular SmartSpice and SmartSpice HPP.

Other General Recommendations:

SmartSpice HPP should operate in multithreading mode (-P N, N>1) to speed up pre- and post-layout circuit transient simulations. HPP mode implements very scalable and optimized model and solver multi-threading algorithms on the latest multicore hardware platforms.

It is recommended to use the SPS solver with non-pivoting reordering algorithm for large circuits. This algorithm can be enabled by setting the following options:

.option pivot=1 pivtol=1e-13

As we presented in Section 8, SmartSpice HPP is also compatible with both the high performance "-turbo" mode and Jivaro for RCL-reduction. These two features are highly recommended in the case of large post-layout netlist simulations.

10. Summary

This application note introduced the main concepts related to SmartSpice HPP and described how to use them. Different simulation scenarios have been presented, each bringing different accuracy/performance trade-offs. The presented simulation results show that SmartSpice HPP can be up to 8x more scalable than regular SmartSpice. They also show that, when integrated with other SmartSpice features, HPP can be up to 40x faster, while keeping the user in control of the simulation accuracy. SmartSpice HPP can be widely used for fast and accurate transient simulations on a variety of designs, from medium- to large-size circuits, either pre- or post-layout.