Position Statement

Joel Emer
Intel Corporation

In the course of processor development, performance models form the most abstract of the representations of a processor design. The other forms in order of increasing detail are: the behavioral or register transfer level (RTL) model, the circuit schematics and finally the layout. Despite the number of representations of a processor, the performance model still must serve a wide variety of purposes varying with phase of the development and specific design objectives.

In the earliest stages of research or advanced development the performance model must serve to explore the potential design space. This encompasses both performance estimates of realistic design alternatives running the most sophisticated benchmarks as are feasible and analysis of the benefits of various individual features of the design.

Later in the design process it is desirable that this same performance model serve as an accurate and high-speed (at least much higher speed than the RTL) model of the design. In this stage the performance model can be used further for feature tradeoff analysis (as design constraints limit choices), detailed performance characterization of the machine, and continued benchmark performance characterizations.

In addition to serving to generate performance estimates, more recently we have called upon performance models to generate power estimates and help with RTL verification. In specific, most traditional verification of the RTL is done to determine correct architectural semantics, but not expected performance. Using a performance model to check an RTL model can uncover performance divots that traditional verification misses.

In order to serve these myriad needs, I see performance models as needing to have three major attributes:

- Speed
- Validity
- Modifiability

Most obviously a performance model needs to run quickly to be of any use. A performance model whose speed is as slow as the RTL model will serve little use.

Second, I am using model validity to represent a broad range of attributes associated with the accuracy and representativeness of the model. Clearly a model that is mis-coded is undesirable, but so is one that cannot model characteristics of importance to the performance of the system (or the individual feature) being studied. These missing characteristics may be design attributes, such as unmodeled components or lack of realistic timing; lack of ability to model certain phenomenon, such as bad-path execution in an out-of-order processor; or inadequate workloads, such as using specmarks for
studying a multiprocessor transaction processing machine. Furthermore, I would define validity to include the generation of models that actually represent what can be built, and not include impossible-to-build characteristics, such as instantaneous global communication.

Finally, a model being used for design space exploration should admit of easy modifiability to study a wide variety of alternatives, while at the same time guaranteeing as great a degree of validity as possible.

The previous two generations of models I developed each focused on speed. The first was a direct execution model that compiled a benchmark (either source or executable) into a performance model. Optimizations were performed in this process that allowed the simulator to avoid simulating activity associated with those activities whose timing behavior could be determined statically. Much of the remainder of the model was written in an event-driven style to take advantage of the relative infrequency of events that needed to be modeled. Despite very a very fast simulation rate, limitations in the range of designs that could be modeled and the general inapproachability of that model (the designers were uncomfortable with the model compilation process and in general with event driven models), forced us to abandon this model and apply a different approach for our next model.

Our next generation models were based on a purely cycle-based approach with carefully speed-engineered code for certain core structures. Unfortunately, again as the model evolved it became increasing unapproachable. This time the culprits were the lack of clear interfaces between components inside the model and the use of #ifdefs to specify design alternatives. The result was at least three complete rewrites of the front-end stages of the pipeline to accommodate design changes and assertions that it would be over a man-month to change individual components such as the branch predictor. Furthermore, the use of often incompatible or broken #define variables controlling conditional compilation with over 400 #ifdefs, which sometimes surround no more than a single “}”, resulted in a model that was fragile and difficult to read or modify.

As a consequence of this experience I felt the need to let the pendulum swing the other way, and I have come to believe that the attributes of validity and modifiability are more important than simulation speed, and we now need to address them as first priorities in our performance modeling endeavors. Furthermore, I believe that a potential avenue for achieving this is to create more modular model frameworks, because through the creation of such a module infrastructure, I believe that we can achieve more model validity and allow easier exploration of the design space.

In our work at Compaq we created a new performance-modeling framework, which we call Asim. The central concept of Asim is that a model is represented as a collection of modules each with a well-defined interface. Starting with the first module of the system each module requires a set of sub-modules that it needs to provide its functionality. The result is that a model is a hierarchy of modules that represents the design being modeled.
As a result of this framework, one can build design alternatives as different implementations of a module and thus configure different models with different modules.

We have already had positive experience with modules where in some cases we were able to create a variety of different implementations of a module. Each implementation is a cleanly coded alternative (without #ifdefs) and can be used as alternatives in a single model or in different models. We also have been able to run 2x2 factorial experiments with two variants of two distinct modules in a hierarchy. We have also seen considerable code reuse, for example though the use a single module sub-tree of a quite realistic memory subsystem in different processor models.

The reuse of modules illustrates an interesting aspect of modularity for improving the validity of the model. Because there is no way to actually check the performance correctness of a machine feature, simply reusing a module many times and using it in many different contexts will give increased confidence that the module is actually coded correctly. For example, though checking that the miss rate of a branch predictor is correct is difficult, at least through using the same predictor for a long time and in different models, one could hope that the bugs will be shaken out of it.

The replaceable module paradigm also leads to flexibility in the level of detail being modeled. Thus alternative implementations of a module might easily provide different levels of detail, depending on the aims of a particular study. But since the alternatives are provided in the same modeling context, the assumptions about the simplifications of the model can be more easily checked. Conversely, the module paradigm can be used to provide a testbed for much for sophisticated (and likely slow) attributes, such as supporting bad-path execution in a multiprocessor model. In all, using modules might let us customize models for the experiment being run, and not force a one-size-fits-all model.

Another aspect of the module hierarchy is that it makes the model more isomorphic to the hardware design. We believe that this is desirable, first for the plug-and-play functionality this engenders, but second because it makes the model more likely to be implementable. Asim goes a step further, by providing and encouraging the use of an abstraction for communication (with delay) between modules. This leads to the more realistic modeling of wire delay (an increasingly important factor in microprocessor design) and avoids the temptation to assume the existence of instantaneous global communication.

In summary, I believe that for these reasons and more in the value of providing a modular framework for performance modeling. Such models can be more realistic, by being more isomorphic to the hardware. They also can be more flexible, allowing a variety of levels of detail of modeling. And finally, they can be more valid, not just because they are more detailed, but because the widespread reuse of modules will result in increased confidence in their accuracy. In the end, this can lead the community to better performance modeling and higher quality and more applicable research results.