Parallel Algorithms for Power Estimation

Victor Kim and Prithviraj Banerjee *
Center for Parallel and Distributed Computing
Dept. of Electrical and Computer Engineering
Northwestern University, Evanston, Illinois
{vicjkim,banerjee}@ece.nwu.edu

Abstract

Several techniques currently exist for estimating the power dissipation of combinational and sequential circuits using exhaustive simulation, Monte Carlo sampling, and probabilistic estimation. Exhaustive simulation and Monte Carlo sampling techniques can be highly reliable but often require long runtimes. This paper presents a comprehensive study of pattern-partitioning and circuit-partitioning parallelization schemes for those two methodologies in the context of distributed-memory multiprocessing systems. Issues in pipelined event-driven simulation and dynamic load balancing are addressed. Experimental results are presented for an IBM SP-2 system and a network of HP-9000 workstations. For instance, runtimes have been reduced from over 3 hours to under 20 minutes in one case.

1 Introduction

Power consumption in VLSI circuits has become a major issue of concern in modern IC design. Power minimization is fast becoming a major optimization objective in VLSI design, along with area and speed. But these multiple objectives often conflict with one another, so engineers need fast and reliable power estimation tools to make the right compromises during different stages of the design cycle. The importance of power estimation and their integration into the design cycle have already been recognized and implemented to a certain extent in some commercial CAD products [11].

The focus of this study is on presenting a comprehensive demonstration of the design issues and performance attainable with parallelized execution of well-performing serial-run power estimation algorithms for combinational and sequential circuits. In particular, we address the parallel design issues for estimating the *average* power dissipation (as opposed to the *worst-case* power) using the following two methodologies:

1. *Exhaustive simulation* [13]: A circuit simulator is driven by user-given input and is used to collect statistics about the circuit activity during a very lengthy period of operation. Although the technique makes no guarantees regarding the input-independence of the resulting power value, it is still often used to provide the baseline for an “accurate and reliable” power estimate in many studies.

2. *Monte Carlo sampling*: A sampling procedure is used to refine a power estimate until certain statistical bounds are satisfied. Two recently published techniques, one for combinational [14] and another for sequential circuits [10], were selected for parallel implementation in this study because of their reliability, as reported by their respective publications. The reader is referred to [14] and [10] for a comprehensive treatment of each methodology. Essentially, Monte Carlo sampling employs a 3-phase framework:

1. **Input generation phase**, in which input vectors are randomly generated.

2. **Circuit simulation phase**.

3. **Statistical analysis phase**, in which the *power sample* (power dissipation during the previous simulation phase) is computed and combined with previous power samples to determine whether the entire process needs to be repeated in order to satisfy certain bounds. (The “termination” criteria are computed from user-given constants on statistical “error” and “confidence.”)

One major difference to note between the combinational and sequential methods chosen for this study is that, in the sequential Monte Carlo method, multiple \((2N = 100)\) instances of the circuit are used to keep track of the circuit’s state transitions according to multiple random input streams [10].

Numerous probabilistic estimation approaches have also been proposed in the literature, but they have also been reported to pose some reliability inconsistencies, especially for sequential circuits. Therefore, probabilistic methods were not considered for this study. The reader is referred to [8] for a general survey of the above methodologies.

In their respective studies, both exhaustive simulation and Monte Carlo sampling methods employ a simulation core to solve for power-related variables. The simulation core may be in the form of a circuit-level simulator such as SPICE [7] or PowerMill [3], but for CMOS circuits, the...
Figure 1: Overview of parallel power estimation approaches described in the paper.

Figure 2: Illustration of pattern-partitioning and circuit-partitioning schemes.

static power component (which is highly technology dependent) is generally negligible in comparison to the dynamic power component (which is due to the capacitive switching activity) [13]. Hence, event-driven logic simulators have been used more extensively in related studies.

Logic simulators often run an order of magnitude faster than circuit-level simulators due to their simplistic modeling of power. Still, both exhaustive simulation and Monte Carlo sampling procedures using logic simulation cores can often require lengthy runtimes in order to ensure their reliability. For instance, transistor-level logic simulation of a relatively small sequential circuit (i.e., s35992) with a relatively small number of input vectors (i.e., 20,000) on a high-end workstation (HP-9000) can require more than 3 hours. Their runtimes are proportional to the runtime of the simulation core and the execution of the simulation framework.

The problem, then, of parallel power estimation in the context of this study is that of achieving speed-up and scalability for the logic simulation core while maintaining the overall execution framework in which the simulation core resides. This problem of parallel simulation is similar to that of parallel logic and fault simulation, for which numerous approaches have been reported [1, 6]. The approach to parallel power estimation in this study is based on schemes similar to those used in parallel logic/fault simulation. However, the following factors distinguish parallel logic simulation in the context of this study from that in the context of other parallel logic/fault simulation:

- The sought-after result is an average power value. Since we seek average power, exact tracking of circuit activity throughout each and every simulation step for a single stream of input vectors is not necessary for estimating the average power value, thereby loosening some of the restrictions commonly encountered in, for instance, parallel simulation of sequential circuits.
- The content and number of input vectors are either specified up-front or generated independently from one vector to another, thereby allowing for potential of speed-up without necessitating concurrent simulation over particular input vector(s).

To the best of the authors’ knowledge, this is the first published report describing a thorough investigation of parallel logic simulation in the context of parallel power estimation. Two types of parallelization schemes were considered for this study (Figure 1). In the pattern-partitioned approach, the given circuit is replicated in each processor to perform concurrent simulation and power computation (Figure 2). For very large circuits, the circuit-partitioned approach distributes the circuit to each processor and performs pipelined computation. Each partitioning scheme is subdivided into handling exhaustive simulation and Monte Carlo sampling frameworks, each of which is further subdivided into combinational and sequential cases. A comprehensive treatment of this work appears in [4].

2 Experimental setup

Each of the following sections include experimental results obtained on benchmark circuits, which are comprised of the five largest circuits from each of the ISCAS'85 suite (c2670, c3540, c5315, c6288, c7552) and the ISCAS’89 suite (s13207.1, s15850.1, s35932, s38417, s38584.1). Average, high, and low speed-ups are displayed as bar charts, along with a corner box listing the serial runtimes of each benchmark in each methodology. The parallel implementations
were developed using the Message Passing Interface (MPI) [2] on a 16-processor IBM SP-2 distributed-memory system. Portability of the implementation is demonstrated on a network of 8 HP-9000 workstations.

As for the simulation core, two different implementations are presented in this paper. For the pattern-partitioning sections (Section 3), the simulation core was adapted from IR-SIM [12], a popular switch-level, variable-delay, event-driven logic simulator. For the circuit-partitioning sections (Section 4), a gate-level, variable-delay, event-driven logic simulation code from MED [14] was used instead, because our particular choice of circuit-partitioning scheme (described in Section 4) required that all signal flow be unidirectional.

For exhaustive simulation power estimation, a fixed sequence of 20,000 randomly-generated input vectors was used. For Monte Carlo sampling, baseline runtimes were established by modifying each framework to collect a fixed number of samples, which roughly averaged out to be equivalent to setting 99%-confidence/1%-error constraints.

3 Parallel power estimation via pattern-partitioning

3.1 Pattern-partitioning for exhaustive simulation

3.1.1 Combinational circuits

The entire circuit is replicated for each of \( P \) processors, and the total of \( N \) input vectors is divided equally into \( N/P \) portions. Each processor then independently drives its local circuit copy with its share of \( N/P \) input vectors. Since communication is minimal in this setup, the speed-ups (Figure 3) were near-linear in most cases, as expected. Quality (in terms of absolute percentage error from serial-run power values) was also maintained highly accurately in all cases (less than 0.1% error).

3.1.2 Sequential circuits

With sequential circuits, one cannot simply partition the input sequences among processors and perform parallel simulation since the state dependencies over time will be violated, and may potentially lead to error in the power estimate. One recent study [5] addressed a similar problem in parallel fault simulation and proposed partitioning the input vectors along with some overlap between the partitions. A similar approach was adopted for parallel power estimation (Figure 4). Processors 1 through \( P-1 \) replicates \( M \) overlap vectors from the \( N/P \) portion owned by the "preceding" processor. Figure 5 shows the effect of the overlap length on quality deviation (percentage error from serial power) and increase in runtime (percentage increase compared to no-overlap) for the largest sequential benchmark in our suite. It has been noted that, in general, the larger the circuit the more input vectors it requires to achieve a properly "warmed-up" state. Therefore, it is expected that, with larger circuits and more processors, the need for proper "warm-up" in the form of some extra input vectors will become evident. In the case shown in Figure 5, adding overlap does appear to decrease quality deviation without severely impacting the runtime. Since the benchmarks were too small to provide a clear indication of an optimal overlap size, 10% overlap was used to demonstrate our concept for the other benchmarks (Figure 6). Quality was maintained to within 1% in all but one benchmark, as noted above.

3.2 Pattern-partitioning for Monte Carlo sampling

3.2.1 Combinational circuits

In parallel Monte Carlo sampling, each processor performs sampling (the input-generation and simulation phases) independently with its own unique stream of random inputs. In order to minimize communication and synchronization overhead, processor 0 is designated to gather the power samples from the other processors and to determine whether...
the stopping criteria have been satisfied. The submission of power samples proceed in an asynchronous “push” manner. The resulting speed-ups (Figure 7) were increasingly sublinear as more processors were added due to the increased communication load on processor 0. However, quality was maintained to within 1% in all cases.

3.2.2 Sequential circuits

Parallel Monte Carlo sampling for sequential circuits is similar to that for combinational circuits, except for the provision that each instance of the sequential circuit (with its own state history) needs to be maintained in the same processor throughout the entire procedure. Hence, each processor is now responsible for $2N/P^1$ instances of the circuit (Figure 8). Each processor maintains the power samples for its share of the circuit instances and sends all the updates to processor 0, which evaluates the stopping criteria. Quality was maintained to within 1% in all except one case.

4 Parallel power estimation via circuit-partitioning

In the circuit-partitioning approach, a circuit is subdivided and distributed across the processors so as to accommodate for large circuits which may not fit in the memory space of a single processor. The circuit is partitioned in a pipelined fashion (Figure 9), wherein each processor acts as a “pipeline stage,” which simulates its portion of the circuit in one phase, and then propagates the resulting circuit activity to the next stage during another phase. The partitioning is performed on a leveled graph of the circuit, so as to maintain the signal delay hierarchy and to streamline communication, which is restricted to one-way send-receive between neighboring stages only. A similar partitioning scheme has been reported in the context of parallel fault simulation with a zero-delay model in [9], but in this study, pipeline execution of variable-delay event-driven simulation is performed in an asynchronous store-and-forward model in order to avoid synchronization overhead and to minimize communication start-up overhead. Any message that needs to propagate beyond the following stage is bundled up with the rest of the downstream message packet, and each stage is responsible for sorting out the destination of the packet’s content. We can then identify two major factors that would influence the actual speed-up:

- Load imbalance may be caused by one or more “bottleneck” stages. In an event-driven simulation environment, the load is a dynamic function of the fanout of each node, their switching activity, and the communication overhead associated with the partition-boundary nodes.
- Communication overhead adds extra latency according to the size and frequency of the message packets. The message frequency can be minimized by using
a store-and-forward approach, as described above. But the message size is determined by the number of edges that were “cut” during partitioning and the transition activity of those edges.

With event-driven simulation, these issues are compounded by the fact that the latency of any one stage can vary unpredictably depending on the inputs to that stage.

4.1 Circuit partitioning strategies
4.1.1 Static partitioning
A fast and simple approach, which has often been found in other parallel simulation work, is to distribute an equal number of nodes to each processor. Another approach may be to use a min-cut type of heuristic to minimize the partition “cut size.” However, static partitioning of either kind does not take into account the actual switching activity which directly relates to the pipeline load. Since the computational cost of static partitioning may not be amortized when used inside a design optimization cycle (during which a complete re-partitioning may be necessary at every cycle), only a simple static partitioning strategy based on balanced node count was implemented for this study.

4.1.2 Dynamic partitioning
Since the load cannot be determined statically, we propose a dynamic profiling method of partitioning, which monitors a portion of its runtime (i.e. first $K$ simulation cycles) for the average load (latency) in each processor, and then re-partitions and re-distributes the circuit before proceeding further with the rest of the simulations. Figure 10 illustrates the re-distribution process, in which each node is weighed as processed load on processor and then re-located to a new processor (if necessary) to meet a “target” average latency (as computed from the profile). The re-assignment and re-distribution procedure takes $O(n)$ time (where $n$ is the number of nodes) so it adds relatively little overhead to the overall runtime. In order to determine whether the profiling load measure should include the accompanying communication latency, two heuristic measures were implemented: Dynamic1, which measured only the relevant simulation load; and Dynamic2, which measured both the simulation and communication latencies. Additional experiments on the profiling length did not reveal any conclusive clues about an optimal profiling length. Therefore, an empirical value of 30 pipeline cycles was demonstrated for the following sections.

4.2 Circuit-partitioning for exhaustive simulation
4.2.1 Combinational circuits
The absence of feedback in combinational circuits allows for a streamlined pipeline operation, configured as shown in Figure 9. As illustrated by the high-low lined superimposed in the speed-up chart (Figure 11) there was a varied distribution of actual performance gain. It should be noted that the quality was identical to serial-run quality for all cases.

4.2.2 Sequential circuits
For sequential circuits, state feedback is allowed to wrap-around the pipeline (Figure 12,13). In order to overlap the workload while the state feedback is being propagated, we use the lesson learned from pattern-partitioned sequential exhaustive simulation by using multiple ($P$) instances of the circuit along with some overlap (same $M$ as in section 3.1.2), yielding the same quality as that from pattern-partitioning.

4.3 Circuit-partitioning for Monte Carlo sampling
4.3.1 Combinational circuits
In pipelined combinational Monte Carlo sampling, the power samples are sent down the pipeline and eventually end up in
the last stage, which is responsible for evaluating the stopping criteria and broadcasting the termination signal. Resulting speed-ups are shown in Figure 14.

4.3.2 Sequential circuits

Pipelined sequential Monte Carlo sampling also requires state feedback, as shown in section 4.2.2. Each pipe stage cycles through the multiple instances of the circuit so as to overlap state propagation in one instance with simulation update in another. Resulting speed-ups are shown in Figure 15.

5 Portability of implementation

On an ethernet network of HP-9000 workstations, the same MPI code that ran on the IBM SP-2 achieved similar performance gains with pattern-partitioning, and up to factors of 3 on 4 processors and 3.5 on 8 processors with circuit-partitioning. Scalability of circuit-partitioning appears to level off after 4 processors due to increased communication costs.

6 Concluding remarks

In this paper, we have presented an overview of a comprehensive study on parallel power estimation using exhaustive simulation and Monte Carlo sampling techniques. The reader is referred to [4] for a detailed treatment of each case study. The pattern-partitioning strategies yielded good speed-ups for all cases on both a distributed-memory system and a network of workstations. With pipelined circuit-partitioning, consistent speed-ups of as much as 7 and individual speed-ups of as much as 10 on 16 processors were achieved. The performance gain was particularly susceptible to the nature of the circuit: a “deep/narrow” circuit (more levels and less “surface area” of the cuts) generally performed better with pipelined partitioning than a “shallow/wide” circuit. The impact of dynamic partitioning also varied. In cases where it did improve against static partitioning, it did so sizeably (as much as 65%), while in other cases it actually degraded the performance (by 15% or less).

Communication overhead is clearly one of the major factors that impacted the performance. The dynamic nature of an event-driven simulation core has shown itself to be a challenge to our load balancing efforts. A future study that integrates dynamic re-partitioning with cut minimization may prove to be more effective.

References