A Scheduling Approach to Parallel Harmonic Balance Simulation

David L. Rhodes[1] and Apostolos Gerasoulis[2] *

[1] US Army CECOM/RDEC, Fort Monmouth, NJ 07703, USA
`d.l.rhodes@ieee.org`
[2] Rutgers University, Piscataway, NJ 08854, USA
`gerasoul@cs.rutgers.edu`

Abstract. Rather than approach the parallelization of the harmonic balance simulation method numerically, a novel scheduling-oriented approach is described. The technique leverages circuit substructure to expose potential parallelism in the form of a directed, acyclic graph (dag) of computations. This dag is then allocated and scheduled using various linear clustering techniques. The result is a highly scalable and efficient approach to harmonic balance simulation. Two large examples, one from the integrated circuit regime and another from the communication regime, executed on three different parallel computers are used to demonstrate the efficacy of the approach.

1 Introduction

The harmonic balance simulation method is widely used in the arena of large-signal, steady-state analysis of nonlinear circuits. It is often times applied to high frequency electronics since it directly accommodates frequency domain models [9, 10, 4]. Direct support for frequency domain models allow harmonic balance to be preferable in situations where distributed elements or frequency-dependent effects are important, for example, where accurate dispersive transmission line analysis (*e.g.* due to skin-effects, dielectric properties) is necessary. Such conditions are typical in radio-frequency (rf) and microwave circuits and increasingly arise in high-speed, deep sub-micron integrated circuits (IC) as well.

Nonlinear circuit simulations, of any type, are computationally intensive and therefore there have been several efforts to speed them via the use of parallel computation. Since linear matrix operations lie at the foundation of many nonlinear circuit simulation techniques, including harmonic balance, it is important to consider a variety of previous circuit simulation parallelization efforts. For example, several non-harmonic balance analog circuit simulators have been speed-up by parallelization of their inherent linear matrix operations [12, 3, 8]. A parallelization of the (linear part of the) harmonic balance method has been demonstrated

* This work supported in part by the DoD HPCMO Office and ARPA under Order Number F425 (via ONR contract N6600197C8534)

[15]. While this effort provided 97% processor efficiency this approach—as well as the others cited—all display limited *scalability*.

Scalability is a measure of the ability of the parallelization technique to *efficiently* make use of an increasing number of available processing elements (PEs). In the ideal, speed-up scales linearly with an increasing number of PEs since *potential* computational power is increasing linearly. Thus, a method which maintains close to linear speed-up over a wide range of available PEs is called scalable. Since we are interested in an efficient technique for today's parallel processors which contain tens to hundreds of PEs, scalability is an important concern.

The method in [15] parallelized the entire analysis of the linear portion of the harmonic balance method on a per-frequency basis, and thus the scalability of this approach is completely limited by the number of frequencies required in the analysis—adding PEs beyond the number of frequencies for the particular input file causes no additional speed up. Alternatively, methods which parallelize the underlying mathematics as it arises in circuit simulation have not, as of yet, achieved reasonable scalability. For example, the results in [12, 3] show efficiencies that are rapidly falling off even for ten PE computing systems. A peak efficiency of about 38% for 8 processors (about $3\times$) has been shown, but speed up then *decreases* beyond 8 processors [8].

A different parallel approach which leverages *circuit substructure*, as we do here, was demonstrated for nonlinear transient domain analysis using the waveform relaxation technique [18]. This approach yielded somewhat better results (efficiencies) than the mathematically oriented approaches but also shows signs of scalability limitations. For example, efficiencies of about 30–60% are shown for ten processors (about $6\times$ speed up), while the method here achieves about 40% efficiency on 64 processors (*i.e.* $25\times$ speed-up) and about 25% efficiency for 128 processors ($32\times$ speed-up). This is for the harmonic balance technique, of course, and not waveform relaxation as in [18].

2 Harmonic Balance Technique

While an overview of the harmonic balance method is given with the aid of Figure 1, please see the references for a full treatment of the technique [9, 10, 4]. The harmonic balance technique divides the circuit or system description into two portions: *linear* and *nonlinear*, interconnected at the *interface* (a set of circuit nodes). The 'balance' then entails iterative adjustment (using an optimization procedure) of the harmonic voltages at the interface until the harmonic currents as given by the linear and nonlinear sides 'agree'. This is done for a fixed set of frequency points, or harmonics, as dictated by the input parameters (the term frequency will be used henceforth for consistency). Sometimes only a few frequency points are used, say for nonlinear amplifier studies, while in other cases hundreds to thousands of frequency points are used, *e.g.* when complete time domain information is important. The 'analysis' on each side of the interface is quite different, but in each case we are interested in computing the frequency-domain currents at the interface given the frequency-domain voltages.

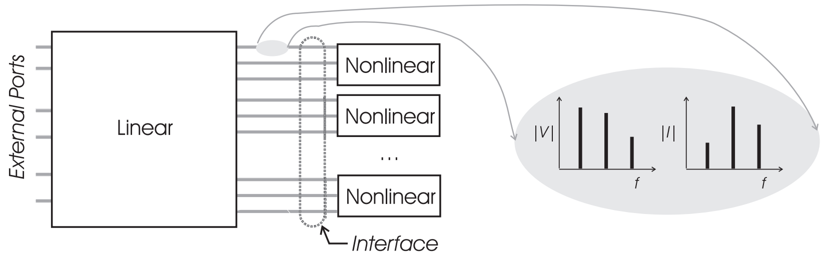


Fig. 1. Visualization of the harmonic balance method

During the balancing iterations, each of the linear- and nonlinear-sides must compute frequency-domain currents given frequency-domain voltages at the interface and at external ports. For the **linear-side**, this is computed as a matrix multiplication (at each frequency) of the frequency-domain voltages with the equivalent ('reduced') *admittance matrix* of the linear portion of the circuit. Thus, the linear portion of the circuit may be reduced to its equivalent admittance matrix at each frequency point *once* and then subsequent 'analyses' during the balance process are merely matrix multiplications.

In order to compute the equivalent admittance matrix (at each frequency) of the linear side, circuit nodes that are internal to the linear side and not connected to either of the interface or external ports must be reduced. As it turns out from Kirchoff's current/voltage laws (KCL and KVL respectively), Gaussian elimination becomes the basic step of such an equivalent reduction. Assuming the typical matrix notation [9, 10, 4], an internal node, k, may be equivalently reduced in the admittance matrix using the expression:

$$Y_{i,j}^{f} = Y_{i,j}^{f} - \frac{Y_{i,m}^{f} Y_{m,j}^{f}}{Y_{k,k}^{f}} \qquad \forall m \neq k \quad \text{and} \quad \forall i, j, f \tag{1}$$

where $\mathbf{Y}^f$ is the admittance matrix for a single frequency point f. Assuming that the reduction is to a constant number of nodes, this reduction can be seen to be of *cubic* order; *i.e.*, as linear circuit nodes are added, computational time for the reduction to a fixed-sized admittance matrix increases in a cubic fashion.

On the **nonlinear-side**, each nonlinear model appears connected individually at the interface. Figure 1 only shows example models which have three nodes (as typical of transistors), but any number greater than or equal to two is possible. As is the case for the linear side, nonlinear model evaluation requires computation of frequency-domain currents given the frequency-domain voltages; these models typically make use of Fourier domain transform techniques to allow time-domain modeling.

The overall current at each interface node (at each frequency) is the sum of current contributions from both the linear and nonlinear sides. In accordance with KCL, this current should be zero. Thus an appropriate error function widely used in harmonic balance uses the L2 norm:

$$error = \left(\sum_{j \,\in\, \text{interface nodes}} \; \sum_{k \,\in\, \text{frequency}} I_{j,k}^2 \right)^{1/2} \tag{2}$$

where $I_{j,k}$ is the sum of all currents entering the interface node indexed j at frequency k. Finally, a conjugate-gradient optimization routine adapted from MINPACK [1] performs the 'balance' by iteratively adjusting the frequency-domain voltages until an acceptably small current $error$ is reached. Of course, at each frequency-domain voltage revision, the linear and nonlinear evaluations described must be used.

3 A Scalable Parallelization Technique

The harmonic balance algorithm, irrespective of parallelization, is then:

Form linear matrix and reduce to nodes of external ports and nonlinear models
Guess harmonic voltages at interface
forever
    Compute harmonic currents from linear side (matrix multiplication)
    Compute harmonic currents from nonlinear side (model evaluation)
    Evaluate $error$ (Eq. 2)
    if $error$ is acceptably small **then done**
    Update harmonic voltage guess (via conjugate-gradient optimization)

Within this analysis framework, several parallelization techniques are possible. In general, since only a few components are interconnected in typical circuits, even as the circuit itself becomes large, the admittance matrices—or other expressions of circuit equations—tend to be *sparse* [16]. Even with this sparseness characteristic, parallelization of the matrix solutions in other circuit simulators has met with limited scalability [12, 3]. On the linear side only, parallelization of the entire process of model evaluations, matrix fill and reduction was shown effective, but that approach has scalability limited to the number of frequencies to be analyzed [15].

At first, it might appear that the harmonic balance iterations will dominate computation time as these are looped. However, as mentioned earlier the linear portion of the computation, represented entirely on the first line of the algorithm, actually dominates many harmonic balance calculations. As the circuit grows, linear fill and reduction tend towards cubic order, while the balance remains close to linear. This isn't always true of course, but many rf/microwave circuits do not exhibit strong interaction among nonlinear elements (unlike digital switching circuits)—although there are certainly exceptions. Thus, in many cases the linear part of the analysis dominates.

Obviously, a critical element for any parallelization technique is that sufficient parallelism be *exposed*, with a granularity that is not too small. **Granularity** may be loosely defined as the ratio of useful computation to inter-PE communication time. It is therefore a function of the particular multi-processor computing resource, even for a fixed problem—thus a technique is needed which both exposes a good deal of parallelism with a granularity useful for the computing resource at hand. The method here is just such a technique, leveraging circuit substructure to expose medium grain parallelism.

An introductory example will help clarify the general technique; first consider the linear part of harmonic balance by the example circuit, composed using sub-circuits, as shown in Figure 2. In Figure 2.A, the circuit A has an instance of circuit $B1$ and $B2$ in it. In turn, $B1$ has instances of $C1$, $C2$ and $C3$ in it while $B2$ contains $C4$ and $C5$. The dark circles are circuit nodes, shown with their node labels.

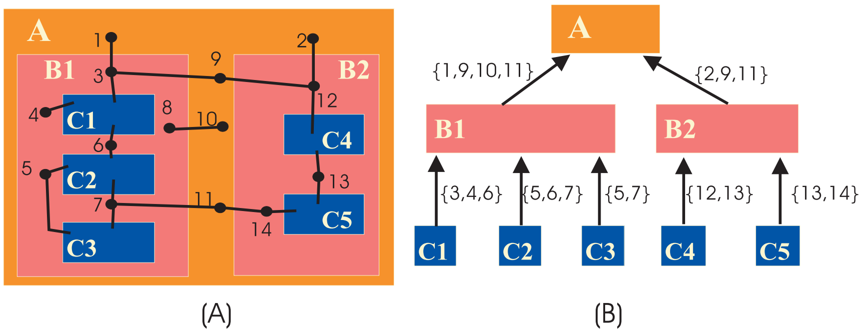


Fig. 2. (A) Linear sub-circuit structure; (B) computational structure resulting from the method, where arrows represent data-dependencies

A typical analysis method would fully elaborate the circuit, forming a very large, sparse matrix [16] for the entire circuit description including all circuit nodes in all sub-circuits. Alternatively, each sub-circuit's internal nodes can be reduced prior to fill into the next level; with this technique, a full elaboration never exists. In this sense, such a method takes advantage of the sparsity present in circuit descriptions but without use of explicit sparse matrix techniques. Figure 2.B shows the computational flow of this approach for the example in Figure 2.A. The computation represented by each box is: (i) **matrix fill** in which requires admittance matrix evaluation of local circuit element models and use of sub-circuit admittance matrices in the fill, and (ii) **matrix reduction** which then reduces internal circuit nodes to allow presentation of the reduced admittance matrix to the next level. Note that the boxes in Figure 2.B represent the *computation* required for this sub-circuit and the arc represents the passing of admittance information to the next hierarchical level. Arcs are labeled using the node names for the equivalent, reduced sub-circuit as it must appear at the higher level. Since sub-circuit matrix information is required prior to computation, this approach gives rise to a directed, acyclic graph (dag), actually a special form called an **in-tree**, of computations.

Note that the linear matrix reduction computation shown in Figure 2.B is for a single frequency, each frequency point in the analysis requires an independent computation, so the whole linear computation becomes a **forest** of in-trees. Inclusion of nonlinear models within the hierarchy is handled by 'carrying' the nodes attached to any nonlinear model 'up' the hierarchy. Such a procedure results in exactly the form Figure 1 while leaving internal linear-only node reduction in hierarchical form as in Figure 2.

For nontrivial circuits, this method exposes a good deal of parallelism, which exists across siblings for the in-tree at each frequency and across the trees themselves. The method in [15] can be viewed as a subset of this approach in that it parallelizes across entire frequency trees only. Since all communicated results are only needed by one consumer (the graph is an in-tree), a message-passing approach is appropriate. The current implementation is message-passing based and uses the Message Passing Interface (MPI) standard [7, 17].

Even though linear computation is the dominant and more interesting part of the overall technique, next consider opportunities for parallelization in the balance portion of the method. First, the ('forever') loops themselves are serially dependent—*i.e.* for some iteration i, the frequency-domain voltage update is done in loop $i-1$, while the update itself cannot be done until the i^{th} *error* is computed. Thus, only parallelization within a single step is possible (assuming that the basic algorithm itself is not changed). Within this loop the nonlinear current computations dominate—the linear current computations are small matrix multiplications (at each frequency), and the error evaluation and frequency-domain voltage updates are also small calculations. The optimization update, a non-sparse, second-order, conjugate-gradient matrix computation is also not readily parallelizable and also is of small size. This matrix size is exactly

$$2 \times \textit{number-frequencies} \times \textit{number-nodes-in-interface} \tag{3}$$

where the factor of 2 arises because the frequency-domain voltages are complex-valued numbers. More importantly, this calculation does not usually tend to be a significant contributor to the iteration computation time.

Fortunately, the nonlinear current computation is readily parallelizable, by parallelizing the evaluation of each nonlinear device, as can be visualized in Figure 1. Thus, the non-linear analysis becomes a straightforward parallelization of the nonlinear model evaluations; the next section describes the background and specific technique for parallelizing the forest of in-trees that arises from the linear computation.

4 Allocation and Scheduling

For the linear part of the method, the forest of in-trees must be allocated and scheduled on the number of PEs to be used. Finding an optimal schedule for this resource-constrained, homogeneous processor problem is, in general, NP-hard [5]. A further difficulty is that inter-node communication times as well as the computation times of the nodes in the in-tree are not known in advance. In practice, however, suboptimal heuristics have been shown to work very well in many circumstances.

A classical approach is dynamic allocation and scheduling of the complete computational graph. Examples include self-scheduling or guided self-scheduling [14]. However since the graph structure itself is fixed, it is reasonable to make use of static allocation techniques. Such techniques have been shown to be quite successful for coarse-grain graph structures, especially when computation times are known.

In this paper, we will consider both completely dynamic as well as static allocation techniques. The well-known master/slave technique—where one PE acts as the manager for the remaining PEs which are slaves—will be used as an example of a fully dynamic approach. For the static case, we will consider a multistep approach as used in PYRROS [20]. In the first step, linear clustering is determined. Next, these clusters are allocated based upon a 'wrap allocation' which assigns clusters to processors. Finally, a local execution ordering, which can be static or dynamic, of tasks on each processor is determined.

The first step of the method is to identify a **linear clustering** [6] of the in-tree structure. Linear clusters are just single dependency chains of the in-tree. To illustrate linear clustering, the in-tree of the previous figure is shown in Figure 3

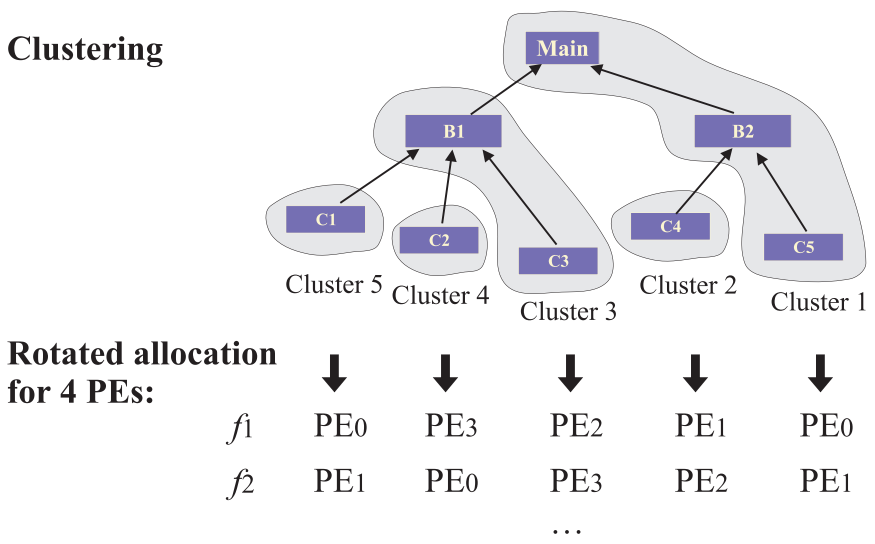


Fig. 3. A sample in-tree and its linear clustering. Rotated static allocation of the clusters for a 4 PE example is shown in the bottom portion.

with its clustering. In PYRROS the DSC clustering heuristic is used, while the method here used a simpler technique. For each as of yet unclustered terminal node, recurse up the in-tree to form a cluster until a node which is already clustered (or the root) is encountered. The 8 node example graph clustered into 5 groups that will result from this method are shown in Figure. The next step is the mapping of clusters to PEs (allocation), here we first used *wrap allocation of linear clusters* wrapping for each in-tree. The final component is local scheduling, which makes use of a dynamic technique where each PE executes the first 'ready' computation in a preferential order. This order is such that computations in lower frequency in-trees—recall that there are multiple in-trees, one for each frequency requested in the analysis, and that all have the same structure—are selected first and then as a secondary consideration those of lower level (farther from the root).

The first results obtained using this method revealed that load-balancing remained problematic. To remedy this situation, **rotated allocation** across 'frequency' in-trees is used. That is, a single in-tree is clustered and wrap allocated, but then this static assignment is 'rotated' across frequencies (modulo the number of PEs). For example, suppose that the in-tree and clustering of Figure 3 is to be allocated on a 4 PE system, where the PEs are labeled: pe_0, pe_1, pe_2, and pe_3. Without the rotated allocation, Clusters 1 and 5 are allocated to pe_0 *for every frequency point evaluation*. Cluster 2, 3, and 4 would be allocated to pe_1, pe_2, pe_3, respectively. Obviously there is no assurance that a balanced allocation exists in such a case, and in fact without node runtimes or at least estimates, no allocation method could make such a guarantee. However, the rotated allocation aids in providing balance by incrementing the allocation for each frequncy (modulo the number of PEs). Here, for example, Cluster 1, is allocated on pe_0 for the first frequency, pe_1 for the next, etc. Although only in-tree was allocated, this provides a method for varying the allocation across the frequency in-tree forest which remains static.

In Figure 4 we compare the speedups of the dynamic master/slave approach (labeled: 'RUN TIME: DYNAMIC'), versus the static clustering and mapping approach (labeled: 'RUN TIME: STATIC') along with the predicted performance by PYRROS (labeled: 'PYRROS PREDICTION: STATIC'). The results show a substantial improvement in speedup when using the static clustering and mapping approach versus the dynamic approach. On the other hand PYRROS prediction show that further improvements are possible for this application. Since PYRROS had access to node computation run-times and communication time estimates to predict the performance, the tool was able to utilize more sophisticated local scheduling algorithms. However, as we mentioned above determining the weights is not free since we need to execute at least a frequency in advance. If we add this cost to the total PYRROS time then the speedup differences between the PYRROS and our method could become less dramatic.

We could of course consider using a static approach similar to that used in PYRROS. However, an approach to obtaining task runtimes would be needed. On the one hand, these could be estimated but it is unclear whether using

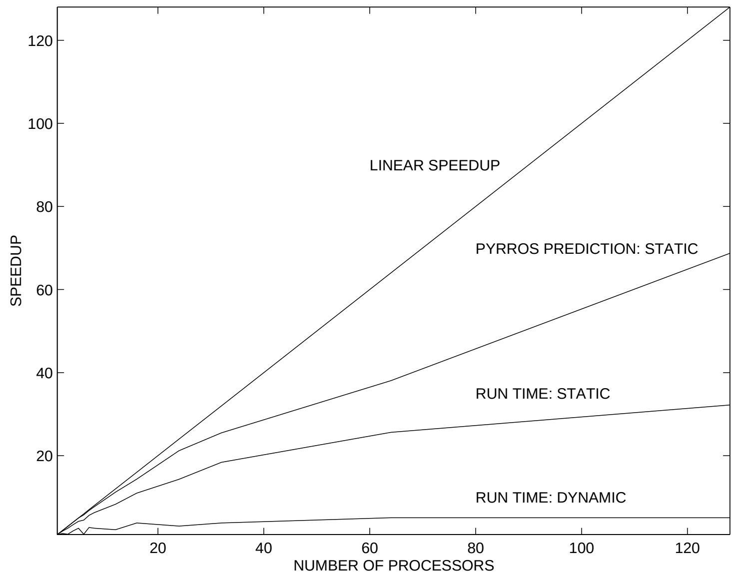


Fig. 4. PYRROS predicted, and actual speedups for static and dynamic approaches on the CRAY-T3E

estimated runtimes would result in a benefit. Alternatively, the in-tree of the first frequency could be analyzed (using a different scheduler) in order to generate accurate runtimes for subsequent analysis. That is first execute one frequency to estimate the computation and communication weights. Then, since runtimes are not typically very dependent on frequencies, re-schedule using PYRROS for the remaining in-trees. Of course, the additional cost for re-scheduling time must be included in the net parallel efficiency achieved—meaning that only 'low-order' methods are potential candidates. We did not implement this approach here but it will be of interest to see how much more improvement is possible for this important class of problems.

In another view of this data, Figure 5 shows the efficiency (rather than speedup) of the three methods, here on a semilog basis to highlight the lower part of the x-axis. The interesting observation is how well the PYRROS tool captures the behavior of the actual run time execution for this problem. Both the run time method and PYRROS prediction show a dip in the curve around 6 processors.

Detailed trace executions were also used to study the techniques and to extract communication times for PYRROS. This was done by instrumenting the code using the VAMPIR tool [13] from Pallas GmbH. From execution times with and without the instrumentation, it was observed that VAMPIR overhead was less than about 5%; thus the traces represent an accurate portrayal of execution. Figure 6 shows a zoomed portion of AGILE's operation for an 8 PE Cray T3E

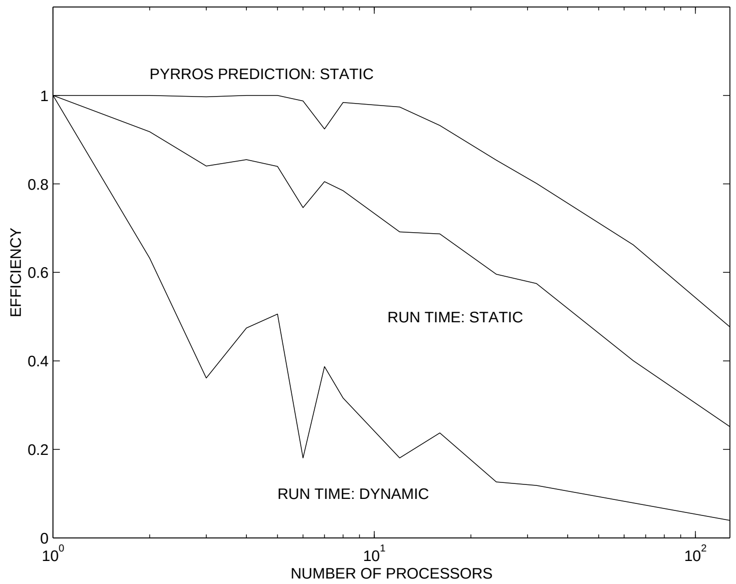


Fig. 5. PYRROS predicted and actual efficiencies for static and dynamic approaches, on the CRAY-T3E

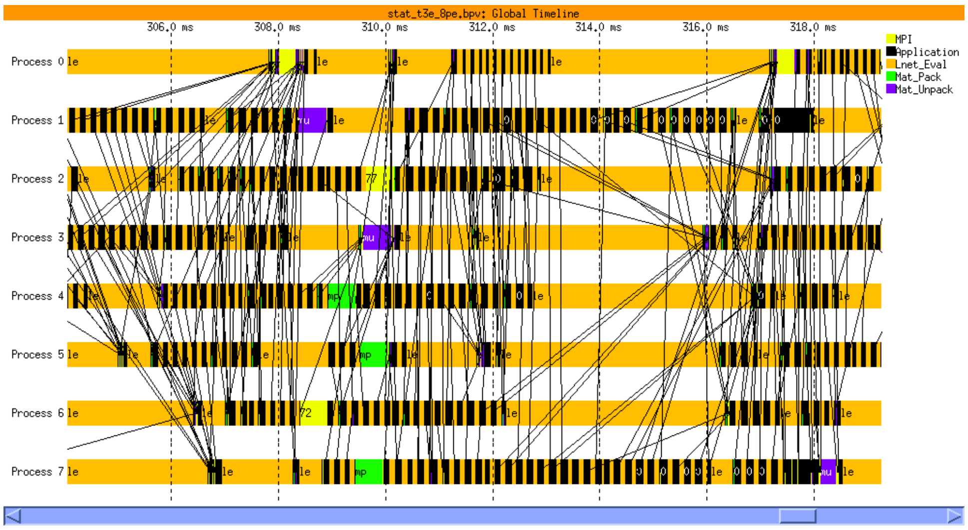


Fig. 6. Zoomed trace execution on 8 PEs for the Cray T3E

system. As can be seen, computation times vary a good deal and most time is spent on useful work (labeled: 'Lnet_Eval'). Lines represent data transmittals. The black areas, labeled as 'User_Code' where this text string fits, is the dynamic local scheduling loop. Zooming in further (at a different time range), Figure 7

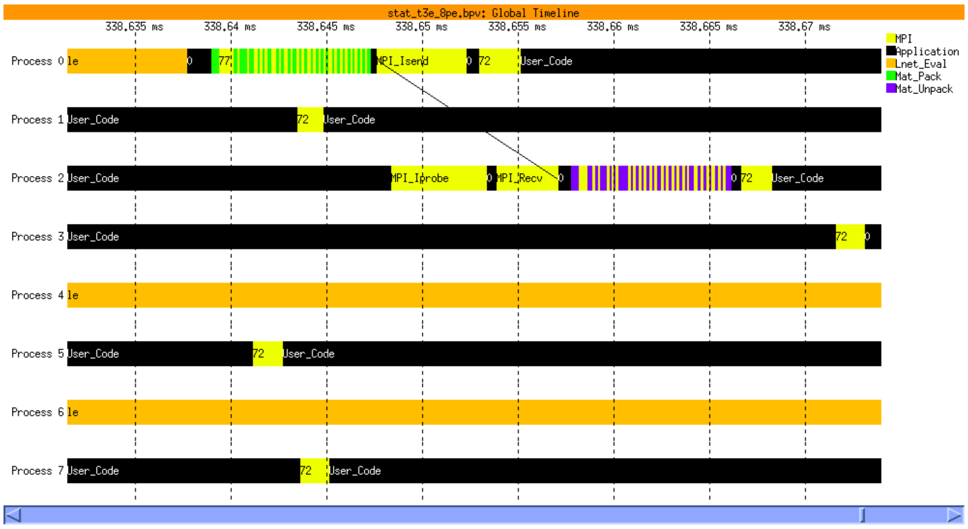


Fig. 7. Detailed trace execution for the Cray T3E

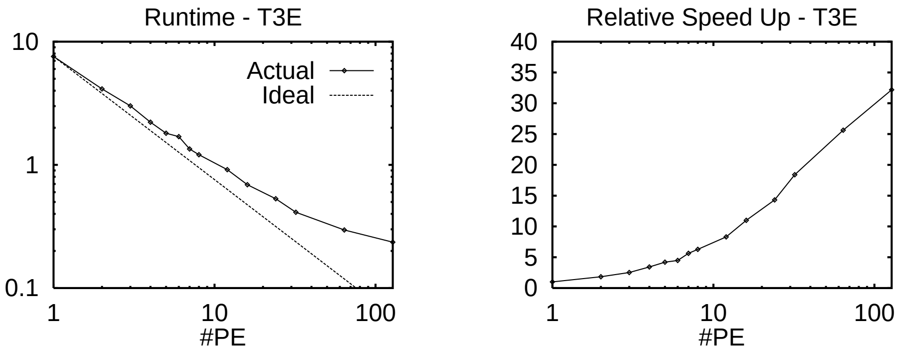
shows the detailed action of packing the matrix data for transmittal via MPI, the communication time-of-flight, and the unpacking. As can be seen, the entire process consumes approximately 30μs (from time point 338.636 ms to 338.666 ms); a value of 30μs was used as communication time for the PYRROS analysis. Note that the observed communication overheads are consistent with other investigations [19, 2, 11].

The culmination of these studies resulted in development of the final method to be used. This is the static allocation method using the dynamic local scheduling technique. This is the allocation/scheduling technique for which results are subsequently shown.

5 Examples and Results

In order to validate and demonstrate the approach, three different large-scale, parallel processors were used, with up to 128 PEs, an order of magnitude higher than presented in previous circuit simulators [3, 18]. These are: Cray's T3E, IBM's SP2, SGI's Origin-2000. Vendor supplied MPI routines were used in each case. Every effort has been made to ensure that the runtimes presented are accurate, however, there are a few factors that cannot be controlled. Since these machines are multi-user, various types of interference are possible including processor multi-tasking on some machines and communication contention. Nevertheless, multiple executions were done on 'quiet' machines and the data presented is felt to be correct. Two examples are executed on each of the processors with varying numbers of PEs.

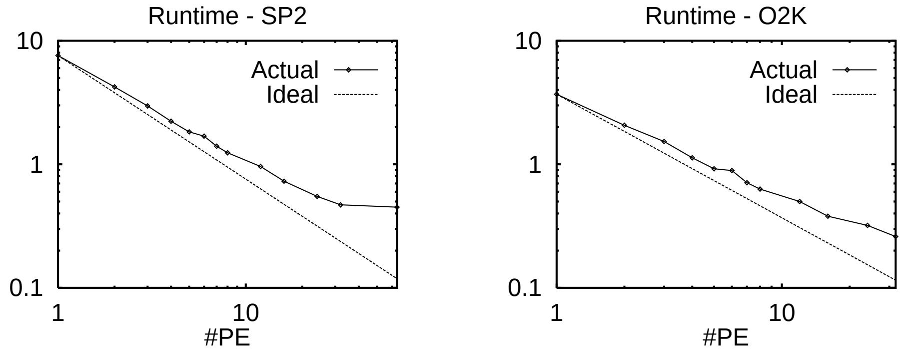
The first example comes from the monolithic, microwave integrated circuit (MMIC) domain. It is a linear circuit hierarchically composed with 7 parametric circuit descriptions which, when elaborated, contains 169 sub-circuit instances. It is analyzed at 20 frequency points, this gives 3,380 computational nodes total in the in-tree forest when elaborated across both circuit hierarchy and frequency. Figure 8 shows the results when using wrap clustering, (frequency index) rotating allocation, and dynamic local scheduling for the Cray T3E for up to 128 PEs. As can be seen, runtime is consistently improving all the way up to 128 PEs and, importantly, shows **reaching 32$\times$ speed up** (Figure 8.B). This appears to be the **highest reported speed up** for any type of circuit simulation. Figure 9.A and Figure 9.B show similar results for the IBM SP2 for up to 64 PEs and the SGI Origin-2000 for up to 32 PEs. All graphs are plotted in log-log form since the ideal runtime curve becomes linear with a slope of -1, *i.e.* ideal runtime is $x/number\text{-}of\text{-}PEs$ where x is a single PE runtime.



(A) (B)

Fig. 8. Ideal and actual runtimes (A) and resulting relative speed-up (B) for the mmic circuit on the Cray T3E

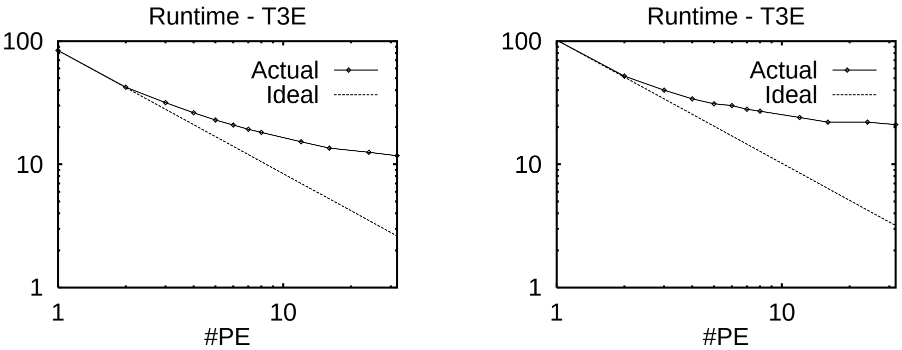
The second example file represents a set of 8 radios arranged in a circular fashion; each radio has a nonlinear transmitter and receiver, as well as filtering circuitry and an antenna model. A simple dispersive (loss dependent on frequency) model for the atmosphere is included. When elaborated, the circuit contains 16 nonlinear elements—a set of 6 frequency (harmonically related) points are analyzed. Figure 10.A shows the results for the linear part of the method only for this input. The parallelization here is not as good as for mmic on the T3E as shown in Figure 8 but consistent improvement up to 32 PEs is shown.



(A) (B)

Fig. 9. Runtime data for mmic circuit for the IBM SP2 (A) and SGI Origin 2000 (B)

Figure 10.B shows the results for the whole analysis. By examining the runtime difference for a single PE in Figures 10.A and 10.B, it can be seen that the nonlinear balance time is about 15 seconds of the 100 seconds run (this clearly illustrates the earlier statement that linear analysis time tends to dominate a harmonic balance simulation). As can be seen in Figure 10.B, *overall* parallel efficiency is not as good, but this is partly because the circuit only has 16 nonlinear elements, hence for any number of PEs over this amount the nonlinear portion of the method cannot be sped up.



(A) (B)

Fig. 10. Ideal and actual runtimes for linear portion only (A) and full analysis (B) for the cosite file executed on the Cray T3E

6 Concluding Remarks

A novel approach to the scalable parallelization of a circuit simulation problem has been developed. This includes a new approach to exposing parallelism as well as application-domain specific methods for allocation and scheduling. Measured speed-ups for three different parallel computers demonstrate the efficacy of the approach which appears to be a world-record speedup for any type analog circuit simulation. Several domain-specific and general-purpose techniques were used to obtain these results. Although the method does not reach the limit as predicted by PYRROS, it is important to realize that those predictions do not include several aspects that are part of real system execution, including: communication contention, dynamic local scheduling, etc.

The authors would like to thank Professor Shirley Browne of the University of Tennessee Knoxville for assistance in obtaining execution trace results using the VAMPIR tool [13], the DoD High Performance Computing Modernization Office's CEN Computational Technology Area.

References

1. `http://netlib.att.com/netlib/minpack/`.
2. Ghieth A. Adandah and Edward S. Davidson. Modeling the communication performance of the IBM SP2. In *Proc. of 10^{th} IEEE Int. Parallel Processing Symp.*, pages 249–57, 15–19 April 1996. Honolulu, HI.
3. C. Chen and Y. Hu. A practical scheduling algorithm for parallel LU factorization in circuit simulation. In *Proc. of IEEE Int. Symp. on Circuits and Systems*, pages 1788–91, 8–11 May 1989. Portland, OR.
4. Benjamin Epstein, Stewart Perlow, David Rhodes, J. Schepps, M. Ettenberg, and R. Barton. Large-signal MESFET characterization using harmonic balance. In *IEEE Microwave Theory and Techniques-S Digest*, pages 1045–8, paper NN–2, May 1988. New York, NY.
5. Michael R. Garry and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, NY, 1979.
6. A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Trans. on Parallel and Distributed Systems*, 4(6):686–701, June 1993.
7. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, Cambridge, MA, 1994.
8. Volkhard Klinger. DiPaCS: a new concept for parallel circuit simulation. In *Proc. of IEEE 28^{th} Annual Simulation Symposium*, pages 32–41, 9–13 April 1995. Phoenix, AZ.
9. K. Kundert and A. Sangiovanni-Vincentelli. Finding the steady-state response of analog and microwave circuits. In *IEEE 1988 Custom Integrated Circuits Conf. (CICC)*, 16–19 May 1988. paper 6.1, Rochester, NY.
10. K. Kundert, J. White, and A. Sangiovanni-Vincentelli. *Steady-state Methods for Simulating Analog and Microwave Circuits.* Kluwer Academic Publishers, Boston, MA, March 1990.

11. Yong Luo. MPI performance study on the SGI Origin 2000. In *IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing (PACRIM'97)*, pages 269–72, 20–22 August 1997. Victoria, CA.
12. K. Mayaram, P. Yang, J. Chern, R. Burch, L. Arledge, and P. Cox. A parallel block-diagonal preconditioned conjugate-gradient solution algorithm for circuit and device simulations. In *IEEE Int. Conf. on Computer-Aided Design (ICCAD'90)*, pages 446–9, 11–15 Nov 1990. Santa Clara, CA.
13. PALLAS GmbH, Brühl, Germany. *Vampir User's Manual, Release 1.1.*
14. Constantine D. Polychronopoulos and David J. Kuck. Guided self-scheduling: A practical scheduing scheme for parallel supercomputers. *IEEE Trans. on Computers*, 36(12):1425–39, December 1987.
15. D. Rhodes and B. Perlman. Parallel computation for microwave circuit simulation. *IEEE Trans. on Microwave Theory and Techniques*, 45(5):587–92, May 1997.
16. V. Rizzoli, F. Mastri, F. Sgallari, and V. Frontini. The exploitation of sparse-matrix techniques in conjunction with the piecewise harmonic balance method for nonlinear microwave circuit analysis. In *IEEE Microwave Theory and Techniques-S Digest*, volume 3, pages 1295–8, paper OO–5, May 1990. Dallas, TX.
17. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference.* MIT Press, Cambridge, MA, 1996.
18. E. Xia and R. Saleh. Parallel waveform-newton algorithms for circuit simulation. *IEEE Trans. on Computer-Aided Design*, 11(4):432–42, April 1992.
19. Zhiwei Xu and Kai Hwang. Modeling communication overhead: MPI and MPL performance on the IBM SP2. *IEEE Parallel and Distributed Technology: Systems and Applications*, 4(1):9–23, 1996.
20. T. Yang and A. Gerasoulis. PYRROS: static scheduling and code generation for message passing multiprocessors. In *Proc. of Sixth ACM Int. Conf. on Supercomuting*, pages 428–37, July 1992. Washington, DC.