Enhancement of Parallelism for Tearing-based Circuit Simulation

Koutaro Hachiya*, Toshiyuki Saito*, Toshiyuki Nakata† and Norio Tanabe‡
*ULSI Systems Development Lab., NEC Corporation
†C&C Systems Research Lab., NEC Corporation
‡ULSI Device Development Lab., NEC Corporation
e-mail:{hachiya,saito,tanabe}@lsi.tmg.nec.co.jp, nakata@csl.cl.nec.co.jp

Abstract— **A new circuit simulation system is presented with techniques "Subcircuit Balancing with Estimated Update operation count"(SBEU) and "Asynchronous Distributed Row-based interconnection parallelization"(A-DR). SBEU estimates Gaussian elimination cost of each subcircuit by counting number of update operations to achieve balanced circuit partitioning. A-DR makes it possible to overlap numerical operations and interprocessor communications in parallel Gaussian elimination of interconnection equations. On a 16-PE distributed memory parallel machine, an experimental simulation shows 9.9 times speedup over 1PE and distribution of the time consumed for each subcircuit is within ±26% deviation from the median.**

I. Introduction

A parallel direct circuit simulator speeds up simulation by using a parallel computer, while maintaining SPICE[1] accuracy. The tearing-based parallelization approach first partitions a given circuit into subcircuits, which corresponds to partitioning the overall equation system into many subsystems and interconnection equations, then extracts subcircuit level parallelism. This method involves circuit partitioning, matrix load-and-solve for each subcircuit, and matrix load-and-solve for the interconnection. It is feasible for parallel machines with relatively slow interprocessor communication speed, because neither shared memory accesses nor interprocessor communications are needed for the matrix load-and-solve phase of each subcircuit. However it needs circuit partitioning which produces balanced subcircuits in terms of computational cost. Furthermore, solution time for the interconnection is large enough to degrade speedup of parallel execution.

Existing tearing-based parallel simulators have resolved these drawbacks as follows. They incorporate min-cut based circuit partitioning which balances subcircuits by predicting their computational costs with number of electrical elements and/or number of nodes(or nets) they have[2][3][4]. Further, they employ Distributed Row-based(DR) interconnection or Multi-Level(ML) interconnection [3] to parallelize solving the interconnection equations, and DR shows good performance. But the existing simulators still have following three problems: 1) imbalance of computational costs for each subcircuit occurs for a matrix-solve intensive circuit because the number of nodes and elements is not enough to express matrix computational cost accurately, 2) communication overheads in DR become more and more serious because improvement of communication speed of modern parallel computers is relatively slow than that of processor speed, and 3) balanced partitioning of the ML interconnection is difficult and has not yet achieved.

In this paper, a new tearing-based parallel circuit simulator with two new techniques to enhance parallelism is presented. These techniques reduce two parallel execution overheads 1) and 2) of existing simulators listed above. To resolve the problem 1), "Subcircuit Balancing with Estimated Update operation count"(SBEU) is used at the circuit partitioning phase. We will show that SBEU enhances speed up of the parallel circuit simulation using a DRAM circuit which contains large LRC lumped networks. To resolve the problem 2), an asynchronous version of DR(A-DR) which overlaps numerical operations and interprocessor communications in parallel Gaussian elimination is devised. Our experimental results using distributed memory MIMD parallel machine Cenju-3[5] show that A-DR is faster than one with old synchronous DR(S-DR), and is also faster than ML parallelization described in [3].

In the next section, general foundations of tearing-based parallelization method are presented. In section III and IV, we show details of SBEU and A-DR, respectively. In section V, experimental results using the parallel machine Cenju-3 is presented. The final section contains concluding remarks.

II. Tearing-based Parallel Circuit Simulation

The concept of tearing method (or diakoptics)[6] is that to solve a large system, we tear the system into n parts and solve each part separately then connect together to get the solution of the overall system. In this section, we show our solution method by tearing. Circuit partitioning method to tear a system is described in section III.

Transient behavior of a circuit is described by non-linear algebraic ordinary differential equation system. Through the application of implicit time integral formula, the differential equations are transformed into a non-linear algebraic equation system. Given an electrical circuit which consists of n subcircuits, the non-linear algebraic equation system has the following form:

$$F_k(x_k, z, t_j) = 0 \quad (k = 1, \ldots, n) \tag{1}$$

$$F_{n+1}(x, z, t_j) = 0 \tag{2}$$

where x_k is a vector of internal variables, $x = (x_1, \ldots, x_n)^t$, and z is a vector of external variables of

ASP-DAC '97
0-89791-851-7$5.00

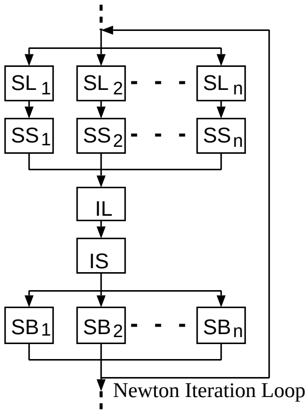

Fig. 1. Flow diagram of solving nonlinear equations for a partitioned circuit.

all subcircuits or node voltages and currents of the interconnection. The equation (2) is called *interconnection equation.*

The non-linear equation system is solved by Newton-Raphson iteration method. The linearized equation system through the method is as follows:

$$R_k \delta x_k + T_k \delta z = -F_k \quad (k = 1, \ldots, n) \tag{3}$$

$$\sum_{k=1}^{n} S_k \delta x_k + y \delta z = -F_{n+1} \tag{4}$$

where

$$R_k = \frac{\partial F_k}{\partial x_k}, T_k = \frac{\partial F_k}{\partial z}, S_k = \frac{\partial F_{n+1}}{\partial x_k}, y = \frac{\partial F_{n+1}}{\partial z}$$

and δx_k and δz are Newton corrections for x and z, respectively.

To solve the equations (3)-(4), we first calculate the Norton equivalent of each subcircuit

$$(G_k, J_k) = (S_k R_k^{-1} T_k, S_k R_k^{-1} F_k) \tag{5}$$

by block elimination technique, which can be done in parallel. Then the interconnection equation solution δz is calculated by

$$\delta z = Y^{-1}(-F_{n+1} + \sum_{k=1}^{n} J_k), \quad Y := y - \sum_{k=1}^{n} G_k. \tag{6}$$

Lastly,

$$\delta x_k = -R_k^{-1} F_k - R_k^{-1} T_k \delta z, (k = 1 \ldots n) \tag{7}$$

are also calculated in parallel for each subcircuit.

The whole process to solve the non-linear equations is depicted in figure 1. To calculate the Norton equivalent (5) for each subcircuit, SL and SS steps are done in parallel. Then, to derive interconnection solution (6), IL and IS steps are done. Lastly, SB step of Eq.(7) is done. SL time, SS time and IS time occupy most of the total time consumed by a circuit simulation. In general, SS time and IS time are dominant for large circuits, while SL time is dominant for small circuits. It is important for efficient parallelization to balance combined SL and SS time for each subcircuit.

III. Subcircuit Balancing with Estimated Update operation count(SBEU)

Before we mention circuit partitioning, we give brief definitions about hypergraph. Hypergraph $G = \langle V, E \rangle$ is a pair of vertices $V = \{v_1, v_2, \ldots\}$ and hyperedges $E = \{e_1, e_2, \ldots\}$, where $e_i \subseteq V$. For a cluster $C(\subseteq V)$, the set of hyperedges cut by C is given by $E(C) = \{e \in E \mid 0 < |e \cap C| < |e|\}$. An electrical circuit can be represented by a hypergraph by representing each electrical element as a vertex and representing each node(or net) as a hyperedge.

The circuit partitioning problem for tearing-based parallel circuit simulation can be stated as the following size-constrained min-cut multiway partitioning:

> For a given circuit described in hypergraph $\langle V, E \rangle$, find n-way partitioning $P^n = \{C_1, C_2, \ldots, C_n\}$ which minimize $F(P^n) = |\bigcup_{h=1}^{n} E(C_h)|$ under size-constraint
>
> $$|max_k\{w(C_k)\} - min_k\{w(C_k)\}| < \epsilon, \tag{8}$$
>
> where $C_i \cap C_j = \emptyset$ for all $i, j (i \neq j)$ and $C_k \subseteq V$ for all $k, (1 \leq i, j, k \leq n)$, and $w(C_k)$ is a weight of cluster C_k. The weight $w(C_k)$ is defined as
>
> $$w(C_k) = k_t \cdot N_t(C_k) + k_u \cdot N_u(C_k), \tag{9}$$
>
> where $N_t(C_k)$ and $N_u(C_k)$ are number of transistors in C_k and number of update operations needed to derive the Norton equivalent of C_k, respectively, and k_t, k_u are constant coefficients.

In general, n should be set to total number of PEs. Minimizing $F(P^n)$ is corresponding to minimizing the size of interconnection matrix Y in Eq.(6) which reduces the time for step IS in the solution method described in section II. The constraint (8) maintains balanced partitioning which avoids PE idling in combined SL and SS step in the solution method. The weight of a cluster $w(C_k)$ models the sum of SL time and SS time. SL time is modeled by number of electrical elements, especially transistors which are the most time consuming elements for model evaluation in SL step. SS time is modeled by number of update operations needed for block elimination in SS step. All existing methods have used number of nodes in C_k to model SS time. However, it cannot express computational cost for SS accurately, since R_k $(k = 1 \sim n)$ in Eq.(3) have different sparsity. So, existing methods sometimes produce imbalanced partitioning results for SS-intensive circuits. The values of coefficients k_t, k_u are determined such that $k_t : k_u = \langle$time taken for a transistor in step SL$\rangle$: $\langle$time taken for an update operation in step SS$\rangle$, which is almost independent of the given circuit.

To realize the above multiway partitioning, we use two-step partitioning: first contract a given circuit by clustering(*initial clustering*) then apply Fiduccia and Mattheyses(FM)[8] min-cut bipartitioning repeatedly until n subcircuits are obtained. The initial clustering enables FM to handle large circuits and to give stable solutions.

Although we can count $N_t(C_k)$ easily, counting $N_u(C_k)$ is difficult and time consuming. A straight-forward way to derive $N_u(C_k)$ is doing the same process in SL and SS steps of the solution method, setting up a matrix with modified nodal analysis(MNA) for C_k and counting the number of update operations while eliminating the matrix. Since numerical values of the matrix elements are not needed for the counting, model evaluation for C_k is not needed and the underlying graph which represents the non-zero pattern of the matrix can be used instead of the matrix itself. For a square matrix $A = \{a_{ij}\}$ of size N, the underlying graph of the matrix A is defined as a directed graph $G_e = \langle V_e, E_e \rangle$, where $V_e = \{1, 2, \ldots, N\}$ and $E_e = \{\langle i,j \rangle | a_{ij} \neq 0\}$. When $a_{ii} \neq 0$, the number of update operations n_{op} needed to eliminate i-th variable is the product of the number of incoming edges and the number of outgoing edges of the vertex i:

$$n_{op} = |E_I(i)| \cdot |E_O(i)| \tag{10}$$

where $E_I(i) = \{\langle j,i \rangle \in E_e \mid j \in V_e, j \neq i\}$ and $E_O(i) = \{\langle i,k \rangle \in E_e \mid k \in V_e, k \neq i\}$. After the elimination, the underlying graph is changed to

$$\begin{aligned} G'_e = \langle V_e - \{i\}, \quad & (E_e - E_I(i) - E_O(i) - \langle i,i \rangle) \\ & \cup (V_I(i) \times V_O(i)) \rangle, \end{aligned} \tag{11}$$

where $V_I(i) = \{j \in V_e \mid \text{for all } j \neq i, \langle j,i \rangle \in E_e\}$, $V_O(i) = \{k \in V_e \mid \text{for all } k \neq i, \langle i,k \rangle \in E_e\}$, and $V_I(i) \times V_O(i) = \{\langle j,k \rangle \mid \text{for all } j \in V_I(i) \text{ and } k \in V_O(i)\}$.

Since counting $N_u(C_k)$ is needed for each merging in initial clustering and for each move in FM, the counting by the straight-forward way is too time consuming although it gives an accurate count. To reduce the counting cost, we estimate the update operation count by applying block elimination technique. Consider we calculate the weight of the resulting cluster C after merging m clusters $C_k (k = 1 \sim m)$ by eliminating the matrix for C. First, we assume that external variables of C_ks are eliminated after all internal variables of C_ks are eliminated. Under this assumption $N_u(C)$ can be calculated by

$$N_u(C) = \sum_{k=1}^{m} N_u(C_k) + n_u(E_{in}) \tag{12}$$

where $E_{in} = \bigcup_{k=1}^{m} E(C_k) - E(C)$ is the set of new internal nodes generated by the merging, and $n_u(E_{in})$ is the number of update operations needed to eliminate all voltage variables of nodes in E_{in} from the matrix generated by eliminating all internal variables of C_ks from the matrix for C. To make the above elimination order assumption hold, as far as possible, merging order must be controlled so that the order of the eliminations which are generated by mergings follow the matrix reordering criteria used in the solution method. Because we use Markowitz reordering criteria[7], we contract the node first which has minimum degree in an underlying graph in initial clustering. Also,in the case of $|E_{in}| > 1$ in (12), we follow the same criteria for eliminations to count $n_u(E_{in})$.

Since there is no unmerging in initial clustering, we can count update operations by first constructing the underlying graph for a whole given circuit and then eliminating all new internal variables generated by each merging in initial clustering. But in FM min-cut, an unmerging occurs whenever a cluster is moved from one block to another. Reverse elimination of variables which are once eliminated from an underlying graph needs complicated mechanisms and will take very high computational cost, so we employ an additional assumption. In FM min-cut phase, the underlying graph(or uneliminated part of the matrix) produced by initial clustering is dense enough with many fill-ins to enable us to look it as a rather dense matrix having uniform density K_d. The uniform density assumption makes it possible to calculate $n_u(E_{in})$ statistically as

$$n_u(E_{in}) = K_d^2 \sum_{k=|E(C)|}^{|E_{in}|+|E(C)|-1} k^2 \tag{13}$$

without any graph manipulation, where E_{in} is a set of new internal nodes generated by a cluster merging, and $E(C)$ is a set of external nodes of the cluster C. Although the value of K_d depends on the given circuit, it can be easily calculated from the underlying graph obtained after initial clustering.

Whole steps of our circuit partitioning with SBEU is briefly depicted in figure 2.

IV. Asynchronous Distributed Row-based (A-DR) Interconnection Parallelization

Even if we employ the min-cut circuit partitioning mentioned in section III, the size of a matrix Y in Eq.(6) tends to be large and IS time in the solution method also tends to be long when a given circuit is large and it is partitioned into many subcircuits. In this section, we describe two existing parallelization methods to solve Eq.(6) and then propose a new technique to enhance the parallelism of one of those methods.

A. Existing Parallelization Methods

Parallelization of solving Eq.(6) can be achieved by parallel Gaussian elimination or hierarchical application of tearing. Since the matrix Y is dense with many fill-ins, parallel Gaussian elimination called Distributed Row-based interconnection(DR) can be applicable, which extract row level parallelism by assigning same number of rows to each PE. The other approach, multi-

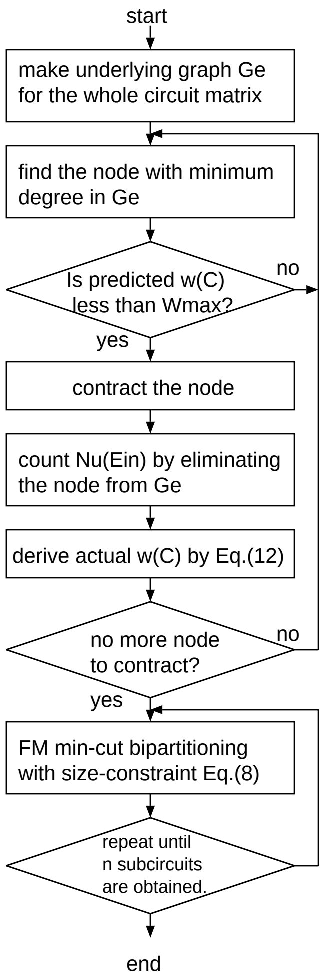

Fig. 2. Flow diagram of circuit partitioning with SBEU

level interconnection(ML), partitions a given circuit hierarchically, namely a like binary tree, and produces sub-interconnections, where the overall matrix has a nested BBD structure.

Although DR shows better performance than ML for the large circuit in [3], which of the two is better depends on the execution platform. Since DR is rather fine grain parallelization, its performance is sensitive to the relative speed of interprocessor communication compared to the operation speed of PEs. An experimental simulation with the same circuit on a different platform with different relative communication speed is shown in figure 6. The details are described in section V.

Abstract program code of DR parallelization of LU factorization is shown in fig.3. At the beginning of an elimination for each variable, all PEs are synchronized with each other by "`barrier`" statement, then "`broadcast`" a pivot-row. So, we call it Synchronous DR (S-DR) hereafter. The `barrier` is needed to avoid over-writing a single pivot-row buffer on each PE before the elimination of the previous variable has finished. Since it is assumed that it is implemented on a distributed shared memory parallel machine which has no hardware support for synchronization and broadcasting, `barrier` and `broadcast` are implemented by writing to memory on remote PEs. The `barrier` is implemented by hand-shaking between one PE and the all other PEs, and the `broadcast` is implemented by hierarchical sending in binary method where half of all PEs are concerned in the sending.

```
for k:=1 to n do
   barrier();
   if (is_PE_contains_Pivot_Row) then
      broadcast A[k,j] (j=k to n);
   endif
   for all i in k+1 to n which concerns this PE do
      A[i,k] := A[i,k]/A[k,k];
      for j:=k+1 to n do
         A[i,j] := A[i,j] - A[i,k]*A[k,j];
      done
   done
done
```

Fig. 3. Abstract program of synchronous DR(S-DR) method

To simplify further discussions, consider the elimination of only one of variables and assume the following simple model:

- There is no congestion in remote write operation.
- It takes time T_C to write data of any size to remote memory and the writing PE is blocked until the write completes.
- The latency of a remote write is also T_C and is independent of data size
- Time to read data of any size from local memory is negligible.
- Time T_U for update operations in eliminating a variable is the same on every PE, and is equal to T_S/n where T_S is elimination time when all eliminations are done by only one PE,and n is number of PEs used in DR.

A time chart of S-DR for 4-PE case is shown in fig.4 based on the above simple model; The time T_{SDR} needed to eliminate a variable by S-DR is

$$T_{SDR} = \frac{T_S}{n} + T_C \cdot n + T_C \log_2 n. \tag{14}$$

The first term is time for the update operations, the second term is time for a `barrier`, and the third term is time for a `broadcast`. The barrier time is proportional to number of PEs and the broadcasting time is proportional to the level of sending hierarchy $\log_2 n$.

B. Proposed Method

Now we propose an asynchronous version of DR,A-DR, which enables DR to overlap numerical operations and interprocessor communications and makes it less sensitive to communication speed. We remove the `barrier` by giving local memory space for all matrix rows to all PEs, which avoid over-writing the previous pivot-row. Further, we implement the broadcasting of a pivot-row by sending the row data sequentially with one PE to which the row

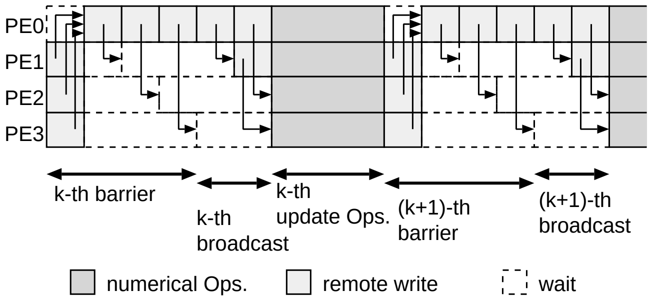

Fig. 4. Time chart of Synchronous DR(S-DR) method

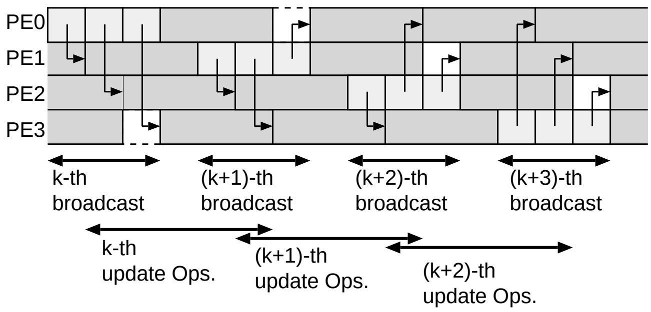

Fig. 5. Time chart of Asynchronous DR(A-DR) method

is assigned. The sequential sending allows the other PEs to start elimination of the next variable immediately. To minimize bubbles (non-numerical operation time), the sequential sending must first send a pivot-row to the PE to which the next pivot-row is assigned. The time chart of A-DR for 4-PE case is shown in fig.5, which is also based on the above simple model. The amount of bubbles in A-DR is much fewer than in S-DR. The time T_{ADR} needed to eliminate a variable by A-DR is

$$T_{ADR} = \frac{T_S}{n} + T_C \qquad (15)$$

in throughput, which shows that communication time is independent of number of PEs used.

Experimental results are shown in the next section. Parallel Gaussian elimination with A-DR can be applied to any dense matrix. It is also applied to Monte Carlo device simulation[9] and reveals its effectiveness.

V. Experimental Results

In this section, we show experimental results of conventional methods using Cenju-1 and Cenju-3 first, then show results of proposed methods using parallel machine Cenju-3.

Both Cenju-1 and Cenju-3 are MIMD parallel machines with distributed shared memory, the relative communication speed of Cenju-3 is about 200 times slower than that of Cenju-1 while PE speed of Cenju-3 is about 30 times faster than that of Cenju-1 (table I). In reality the result above is somewhat exaggerated since the latency is measured under the condition where no conflicts are incurred

		Cenju-1	Cenju-3
MFLOPS		1.6	50
remote write	latency (μsec)	5.2	37
	throughput(Mbyte/sec)	5	34.7
relative speed	latency(write/flops)	0.12	0.00054
	throughput (byte/flop)	3.1	0.69

TABLE I
Performance comparison of Cenju-1 and Cenju-3

name	# of Trs.	# of LRCs	# of nodes
CKT1	15826	11252	14051
CKT2	6873	1180	3831

TABLE II
Size of the benchmark circuits for experiments

by other remote memory accesses which makes it more ideal for bus based Cenju-1 rather than for Cenju-3 which has a multistage interconnection network. Because of the diffrence between their relative communication speeds, speedup performances of the conventional simulators on Cenju-3 are worse than that on Cenju-1(fig.6). Benchmark circuits used here are DRAM circuits and their sizes are shown in table II. CKT1 includes many LRCs to model power lines.

With SBEU and A-DR, simulation speed for CKT1 is about 2 times faster at 8 PE and 16PE than that without SBEU and with ML(fig.7). The speedup over 1PE of the new simulator almost reaches to 10 times at 16PE. Distribution of total computational time for each subcircuit(SL+SS time) in the simulations of CKT1 with and without SBEU is shown in fig.8. Coefficients in Eq.(9) and (13) are set to $k_t : k_u = 100 : 1, K_d^2 = 0.075$, which are optimal values derived by some experiments on Cenju-3, and the circuit partitioning time with SBEU is only 44.5sec using a workstation NEC EWS4800/360EX(R4400SC 200MHz). Deviations from the median of the time for each subcircuit are within $\pm 26\%$ in the simulation with SBEU, although the deviations span to $\pm 62\%$ in the simulation without SBEU. With SBEU,the maximum SL+SS time is reduced to about half compared to that without SBEU. When we vary n,number of PEs for A-DR, total time for numerical operations is inversely proportional to n, and total time for communication is almost constant(fig.9). This result follows the relation predicted by Eq.(15).

On the other hand, the simulation of CKT2 with ML has already been efficient enough(fig.7) because the circuit is partitioned into ballanced subcircuits and ballanced sub-interconnections even without SBEU. The simulation of CKT2 with A-DR and SBEU shows almost the same performance as that with ML.

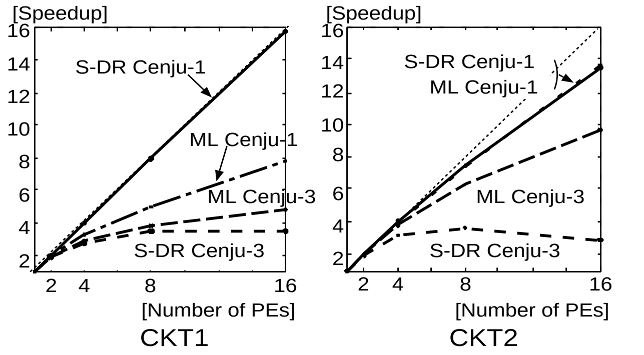

Fig. 6. Speedup of the conventional simulators on Cenju-1 and Cenju-3

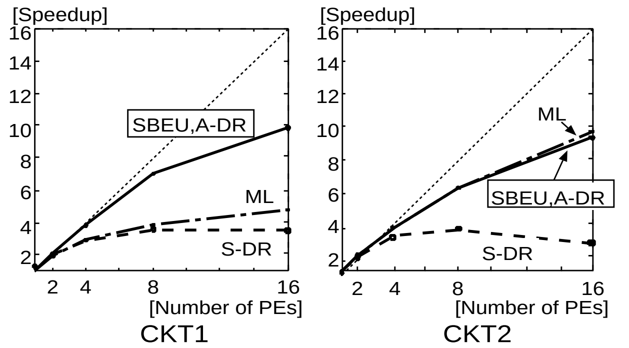

Fig. 7. Speedup of the new simulator on Cenju-3

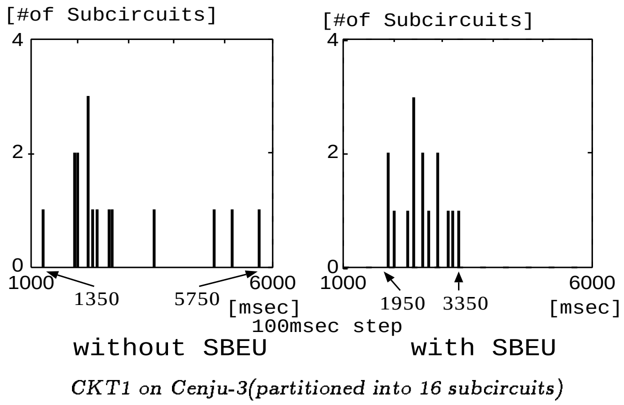

Fig. 8. Histogram of computational times for each subcircuit

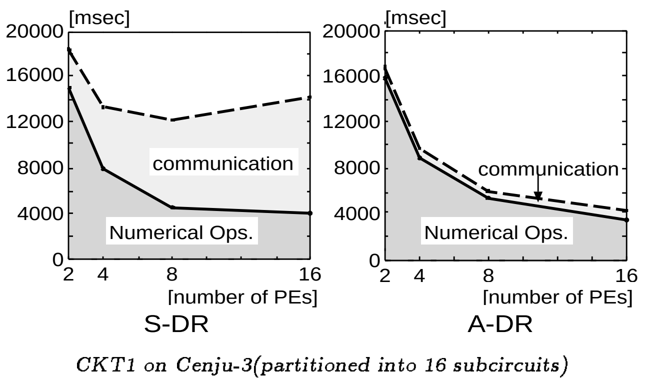

Fig. 9. Comparison of S-DR and A-DR time taken to solve interconnection equations

VI. Conclusion

We have presented a new parallel circuit simulator with tearing, with two new techniques, SBEU and A-DR. SBEU counts the number of update operations with approximations and predicts matrix-solve time for clusters or subcircuits, then enables circuit partitioning to produce balanced subcircuits, which enhances speedup of the simulator. A-DR enables parallel Gaussian elimination to overlap numerical operations and interprocessor communications and makes it less sensitive to communication speed, which reduces interconnection matrix solution time. The new simulator can simulate a matrix-computation intensive circuit about two times faster than conventional simulators. But interconnection solution time is still too long. For future work, applying the levelized solver with incomplete LU factorization [10] to the simulator is a good candidate to reduce the interconnection solution time.

Acknowledgements

The authors would like to thank Mr.Fujitaka and Dr.Takamizawa who support our development of the simulation system. We would also like to thank C.Mizuta, M.Kanoh, S.Tajino, K.Shimano and C.Katoh for developing and maintaining the system. Also, we would like to thank Y.Hoshino for establishing know-how to apply the system to practical memory chip design.

References

[1] L.W.Nagel, "SPICE2:A computer program to simulate semiconductor circuits," Memo No. ERL-M520, Electronics Research Lab. University of California, Berkeley, May 1975.
[2] D.C.Yeh and V.B.Rao,"Partitioning Issues in Circuit Simulation on Multiprocessors," Proceedings of ICCAD-88,1988.
[3] H.Onozuka,M.Kanoh,C.Mizuta,T.Nakata and N.Tanabe, "Development of Parallelism for Circuit Simulation by Tearing," Proceedings of European Design Automation Conference, 1993.
[4] T.Kage, F.Kawafuji and J.Niitsuma, "A Circuit Partitioning Approach for Parallel Circuit Simulation," IEICE Trans. Fundamentals, Vol.E77-A,No.3,Mar.1994.
[5] K.Muramatsu,S.Doi,T.Washio,T.Nakata, "Cenju-3 Parallel Computer and its Application to CFD," Proceedings of the 1994 International Symposium on Parallel Architectures, Algorithms, and Networks, Dec.1994.
[6] G.Kron, "Diakoptics – The Piecewise Solution of Large-Scale Systems," London,England,Macdonald,1963.
[7] W.J.McCalla, "Fundamentals of Computer-Aided Circuit Simulation," Kluwer Academic Publishers, 1988.
[8] C.M.Fiduccia and R.M.Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," Proceedings of 19th DAC, 1982.
[9] K.Shigeta,T.Nakata,T.Iizuka, and H.Katoh, "Parallel Calculation of Monte Carlo Device Simulation," Technical Report of IEICE, ED93-90,DSM93-104,VLD93-45,1993.
[10] K.M.Eickhoff and W.L.Engl, "Levelized Incomplete LU Factorization and its Application to Large-Scale Circuit Simulation," IEEE Trans. on CAD, Vol.14, No.6, June 1995.