Linear Time Hierarchical Capacitance Extraction Without Multipole Expansion

Saisanthosh Balakrishnan,Jong Hyuk Park, Hyungsuk Kim, Yu-Min Lee and Charlie C.-P. Chen
Department of Electrical and Computer Engineering, University of Wisconsin-Madison
1415 Engineering Drive, Madison, WI 53706
(e-mail: chen@engr.wisc.edu)

Abstract— **Recently, hierarchical capacitance extraction algorithms have been shown efficient and accurate capacitance extraction algorithms. An improved algorithm is also proposed to remove its runtime dependency on the number of conductors by a combination of hierarchical and multipole expansion algorithm. In this paper, we show that with the introduction of hierarchical merging operation and supernode representation, we can achieve linear runtime and accuracy without involving multipole expansion. Experimental results show over 10X runtime improvement and 20X memory saving over the multipole approaches with comparable accuracy and better numerical stability.**

I. Introduction

As VLSI clocking frequency and integration density continue to increase, it is crucial to accurately model the effects of three-dimensional coupling through both the electric and magnetic fields interaction to verify signal integrity and timing convergence. Furthermore, the recent advances in system-on-a-chip (SOC) technology require the integration of digital and RF circuit into one chip to reduce manufacturing cost. However, the strong radiation and power fluctuation created by RF circuits may cause soft errors and big power dip/ground bounces to the digital circuits. As a result, efficient and accurate 3-D capacitance and inductance extraction algorithms are need to facilitate the signal integrity verification for multi-giga hertz VLSI design and SOC application.

For this reason, numerous parasitics extraction algorithms have been proposed in the literature [6], [7], [2], of which the hierarchical approximation methods, such as the fast multipole algorithm and Appel's algorithm have been shown to be very effective and have been used in several extractors to significantly improve the matrix solving runtime [6], [2]. A recent study indicates that integration of both methods improve performance significantly [7] but with considerable increase in software complexity.

In this paper, we show that, without using multipole algorithm, the performance and memory requirement of Shi's approach can be significantly improved by using an innovative encoded Oct-tree data structure rather than the conductor-oriented data structure. Together with the introduction of new hierarchical merging operation and supernode representation, we can effectively resolve the runtime issue of Shi's algorithm especially when the number of conductors are huge. It is interesting that the new hierarchical merging operation and supernodal representation comes from the proof of the accuracy bound from the Shi's original paper, the accuracy of this operation is thus ensured. As a result, the runtime of our new algorithm is always linear and constantly faster than the above two algorithms. Second, with the introduction of superleaf representation, we can effectively take care of the runtime and memory impacts caused by many small panels introduced by the VIAs or irregular interconnect structures. Combined with the hierarchical merging operation, this new algorithm will no longer be slowed down by the irregular geometries.

Third, by effectively utilizing the windowing method, we find that the runtime can be improved by orders of magnitudes. Finally, our hierarchical refinement procedures of the capacitance and inductance extraction are highly coordinated and hence the models generated by our algorithm are accurate for a large range of frequencies.

Experimental results shows over 10X runtime improvement and 20X memory saving over both the hierarchical and multipole approaches with equal or better accuracy and numerical stability.

The rests of this paper are organized as follows. In Section 2, the brief overview the hierarchical capacitance extraction method is presented. In Section 3, we present the experimental results.

II. Hierarchical Capacitance Extraction and Improvements

In this section, we present the basic hierarchical capacitance algorithm and various enhancements.

A. Capacitance Extraction

The capacitance of a m-conductor geometry can be summarized by a $m \times m$ capacitance matrix C. The diagonal entries C_{ii} of C are positive, representing the self-capacitance of conductor i, and the non-diagonal entries C_{ij} are negative representing the coupling capacitance between conductors i and j. To determine the j^{th} column of the capacitance matrix, we need to solve for the surface charges by raising conductor j to unit potential while ground all other conductors. Then C_{ij} is numerically equal to the charge on conductor i. This procedure is repeated m times to compute all columns of C.

Each of the m potential problems can be solved using an equivalent free-space formulation where the conductor dielectric interfaces are replaced by a charge layer of density σ. Assuming a homogeneous dielectric, the charge layer in the free-space problem will be the induced charge in the original problem if σ satisfies the integral equation

$$\varphi(x) = \int_{surfaces} \sigma(x') \frac{1}{4\pi\epsilon_0 \parallel x - x' \parallel} da', x \,\epsilon\, surfaces \tag{1}$$

,where $\varphi(x)$ is the known conductor potential, da' is the incremental conductor surface area, x, $x \epsilon \Re^3$, and $\parallel x \parallel$ is the Euclidean length of x given by $\sqrt{(x_1^2 + x_2^2 + x_3^2)}$.

In this approach the conductor surfaces are divided into n small sub panels and it is assumed that on each panel A_i; a charge q_i is uniformly distributed. Then for each panel A_i, an equation is written which relates the known potential on A_i, denoted by v_i, to the sum of contribution of potential from charges on all n panels $A_1, \dots A_n$. The result is a dense linear system

$$Pq = v \tag{2}$$

where, $q \,\epsilon\, \Re^n$ is the vector of panel charges, $v \,\epsilon\, \Re^n$ is the vector of known panel potentials, and $P \,\epsilon\, \Re^{n \times n}$ is the potential coefficient matrix where each entry.

$$P_{ij} = \frac{1}{area(A_i)} \int_{x_i \epsilon A_i} \frac{1}{area(A_j)} \int_{x_j \epsilon A_j} \frac{1}{\parallel x_i - x_j \parallel} da_j da_i. \tag{3}$$

Matrix P is symmetric and positive definite. This linear system has to be solved for panel charges, and the capacitances are derived by summing the panel charges.

B. Hierarchical Refinement

The basis for the Shi's hierarchical refinement algorithm and accuracy bounds is based on the following observation.

Theorem 1: **Hierarchical Refinement Theorem** *Consider a panel A divided into two sub-panels A_1 and A_2 of similar shape and size. Let the radius of the smallest sphere that contains both A_1 and A_2 be R. Consider a point x' distance r from the center of the sphere, for some $r > R$. The error of the potential estimation at point x' by using a uniform charge distribution at A to approximate the aggregated potential of individual charge distributions of A_1 and A_2 is bounded by $O(R/r)$.*

Using the above theorem, an hierarchical panel refinment scheme is defined. As shown in Figure 1, given any two panels, if the estimated potential interaction is less than the user provided error bound, then the panels are allowed to interact at this level of detail. However, if the potential factor estimate is too large then Procedure `Subdivide` subdivides a panel into two new panels and stores it in a binary tree. Thus, a binary tree is needed to represent one conductor surface and each leaf node represents one final panel that is no longer divided.

P_{ab} = PotentialEstimate (A, B)
$R_a \leftarrow$ longest side of A and $R_b \leftarrow$ longest side of B
if $P_{ab} \times R_a < Peps$ and $P_{ab} \times R_b < Peps$ **then**
  RecordInteraction (A, B)
else
  if $R_a > R_b$ **then**
    Subdivide (i)
    Refine ($A \rightarrow left, B$)
    Refine ($A \rightarrow right, B$)
  else
    Subdivide (j)
    Refine ($A, B \rightarrow left$)
    Refine ($A, B \rightarrow right$)
  end if
end if

Fig. 1. Refine (Panel A, Panel B)

To solve the linear system $Pq = v$, an iterative algorithm requires the multiplication of the coefficient matrix P with a vector, which normally takes $O(n^2)$. To fully utilize the hierarchical nature of this algorithm, [2] developed a recursive algorithms to compute the charge, collect and distribute the total potential interactions in $O(n)$ time. The algorithms are summarized in Figure 2, Figure 3, and Figure 4.

Return if A is a leaf
AddCharge ($A \rightarrow left$)
AddCharge ($A \rightarrow right$)
$A \rightarrow charge = A \rightarrow left.charge + A \rightarrow right.charge$

Fig. 2. AddCharge (Panel A)

C. Hierarchical Merging, Supernode, and Superleaves

In this section, we introduce our major improvements over Shi's algorithm. Theorem 1 can be rephrased as

Return if A is empty
for all Panels B, if Panels A and B have interaction **do**
  $A.Potential = A.Potential + B.charge \times P_{ab}$
end for
CollectPotential ($A \rightarrow left$)
CollectPotential ($A \rightarrow right$)

Fig. 3. CollectPotential (Panel A)

Return if A is a leaf
$A \rightarrow left.potential = A \rightarrow left.potential + A.potential$
$A \rightarrow right.potential = A \rightarrow right.potential + A.potential$
DistributePotential ($A \rightarrow left$)
DistributePotential ($A \rightarrow right$)

Fig. 4. DistributePotential (Panel A)

follows

Theorem 2: **Hierarchical Merging Theorem** *Given two panels A_1 and A_2, let the radius of the smallest sphere that contains both panels be R. Consider a point x' distance r from the center of the sphere, for some $r > R$. The error of the potential estimation at point x' by using a superpanel A to approximate the aggregated potential from individual charge distributions of A_1 and A_2 is bounded by $O(R/r)$.*

By the above theorem, we can recursively merge two top-level panels to form a supernode. The supernodes can also been recursively merged into higher level of supernode. A supernode acts exactly like a normal panel except that its radius is defined by the minimum radius of sphere which covers its sub-panels and its area is equal to the summation of the two sub panels. We define the Merge_Nodes and Merge_SuperNodes operations as shown in Figure 5 and Figure 6.

Theorem 3: **Lazy Refinement Theorem** *Given two panels A_1 and A_2, let the radius of the smallest sphere that contains both panels be R. If there does not exist any panel which requires to interact with A_1 or A_2 individually, then the error caused by using a superleaf A to approximate the aggregated potential interaction with other panels is bounded by P_{eps}.*

The Lazy Refinement theorem serves as the foundation of using superleaves to avoid unnecessary refinement. A superleaf is a merged panel which is not required to be refined by any other panels in the system. The self capacitance of a superleaf can be computed by only considering the interactions between its sub-panels independently. In this way, it is not necessary to create two variables for the individual panels for the global capacitance computation. The impacts of small fractional panels to the runtime is lessened. The Hierarchical Merging operation, supernode, and superleaf concepts are illustrated in Figure 7.

Partition the panels into subpanels based on the window boundaries
for Each window W **do**
  for every two adjacent panels (A, B) inside the window W **do**
    Merge two adjacent subpanels into a supernode SN.
    $A_a \leftarrow$ Area (A)
    $A_b \leftarrow$ Area (B)
    Area of Supernode $SN(A_{sn}) \leftarrow A_a + B_a$
    Centroid of $SN \leftarrow \frac{A_a \times (x_a, y_a, z_a) + A_b \times (x_b, y_b, z_b)}{A_a + A_b}$ where (x_a, y_a, z_a) is the centroid of panel A and (x_b, y_b, z_b) is the centroid of panel B.
    Let R of the supernode $\leftarrow$ Radius of the smallest sphere that contains both the panels A and B.
    The $left$ and $right$ links of the supernode points to panels A and B i.e. $SN \rightarrow left = A$ and $SN \rightarrow right = B$.
  end for
end for

Fig. 5. Merging two panels into a supernode.

for Each Window W **do**
  for every supernode (S_1, S_2) **do**
    $A_{S_1} \leftarrow$ Area (S_1)
    $A_{S_2} \leftarrow$ Area (S_2)
    Area of the supernode (SN), $A_{SN} \leftarrow A_{S_1} + A_{S_2}$
    Centroid of $SN \leftarrow \frac{A_{S_1} \times (x_a, y_a, z_a) + A_{S_2} \times (x_b, y_b, z_b)}{A_a + B_a}$ where (x_a, y_a, z_a) is the centroid of supernode S_1 and (x_b, y_b, z_b) is the centroid of supernode S_2.
    R of the supernode $\leftarrow R_{S_1}/2 + R_{S_2}/2+$ Distance between S_1 and S_2
    The $left$ and $right$ links of the supernode points to supernodes S_1 and S_2 i.e. $SN \rightarrow left = S_1$ and $SN \rightarrow right = S_2$
  end for
end for

Fig. 6. Merging two supernodes

D. Adaptive Refinement

Adaptive refinement is another effective way to reduce the number of refined panels to achieve the target accuracy. Figure 8 shows that by proper refinement, we can use small number of panels to obtain equal or better

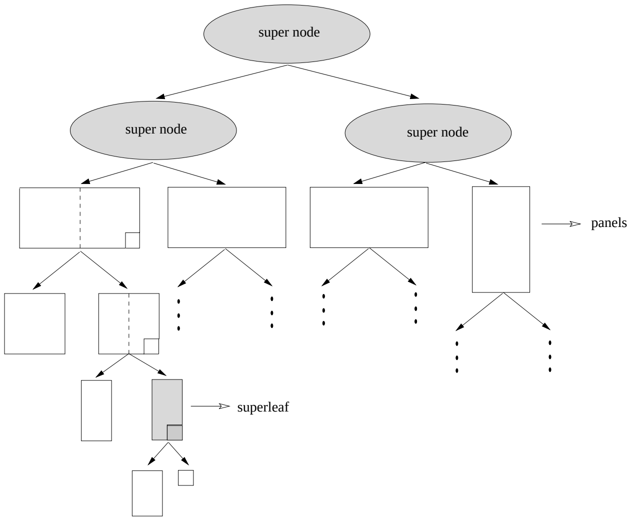

Fig. 7. Merging panels into supernodes

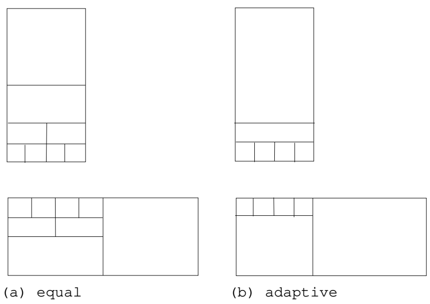

Fig. 8. Discretization of panels equally and adaptively

accuracy.

We utilize an enhanced adaptive mesh refinement algorithm, given two conductors, we utilize the following divide factor to chose a cutting position to avoid further refinement.

$$DivideFactor = \frac{Peps}{P_{ij}} \times \frac{1}{R}. \quad (4)$$

That is, we divide panel into two unequal size sub panels which have lengths of $DivideFactor \times R$ and $(1 - DivideFactor) \times R$, respectively. The DivideFactor is carefully decided to use the minimum number of dividing panels to satisfy accuracy requirement.

E. Encoded Oct-Tree Data structure

Shi's algorithm can be significantly speed up using the Oct-tree data structure instead of conductor based data structure especially for the case when there are many small conductors presented since Shi's algorithm searches

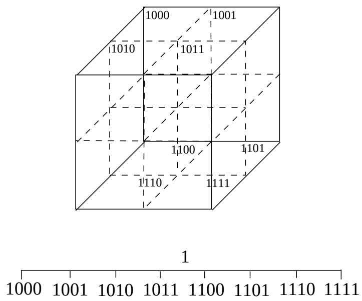

Fig. 9. Three dimensional design divided into cubes as in octtree and its tree representation

the conductors sequentially and builds the hierarchical based on conductors. In our algorithm, we sort the panels according to its geometry location based on Oct-tree data structure. An Oct-tree data structure arranges objects according to their locations such that geometry closer objects will be put closer in the data structure. As a result, the Oct-tree structure give us an efficient way to search for nearby objects. To further enhance the runtime of Oct-Tree search, we devise an encoded hash structure such that the query for the adjacent boundary only use $O(n)$ time.

Our encoded Oct-tree is described as follows. We first divide the three dimensional design into 8 overlapping cubes or windows, similar to 8 nodes in an oct-tree. Figure 9 shows an oct-tree, the keys assigned to each node, and tree representation of the oct-tree. Since fast retrieval of conductors in a window and its adjacent windows is critical we do not use the hierarchical tree data structure. We then use a new data structure of a key and hash table for accessing the window. Each window is named in such a way that the higher level internal nodes can be distinguished from the lower level nodes. The MSB or most significant bit is a place-holder bit, which is always 1. Thus, the root node is represented by 1. A simple two dimensional representation of a quad tree is shown in the figure. In general each key corresponds to some composite data describing the window. To further simplify the data structure, the keys are generated in such a way that, the memory location holding the data of the window can be intrapolated from it, thus saving the space needed for a hash table. The key space generated by the above method is very convenient for tree traversals. In order to find child nodes, the parent key is left shifted by d bits, and the result is `OR`ed to daughter number from 0 to $2^d - 1$. Also the key retrieval mechanism is much more flexible in terms of the kinds of accesses which are allowed. If we wish to find a particular node of a tree in which pointers are used to traverse the tree, we must start at the root of tree and traverse until we find the desired node (which takes of order log N operations). On the other hand, a key provides immediate $O(1)$ access to any object in tree.

F. Extension with Inductance Models

We integrated our capacitance extraction with a formulae-based partial inductance extraction method to build the PEEC (Partial Equivalent Element Circuits). The formulae for both self inductance and mutual inductance are present as follows:

Partial Self Inductance Calculation:

$$L_{self} = \frac{\mu_0}{2\pi}\left[l\,ln\left(\frac{2l}{w+t}\right) + \frac{l}{2} + 0.2235\frac{(w+t)}{l}\right] \quad (5)$$

Equation (5) shows the formula for self inductance computation when $l \gg (w+t)$ where l is the length of the writes, w and t are the width and thickness of the rectangular cross section, respectively. Equation 6 is for mutual inductance of two parallel wires of distance d and equal length l when $l > d$.

Partial Mutual Inductance Calculation:

The formula for mutual inductance of parallel filaments with equal length is:

$$M = \frac{\mu_0 l}{2\pi}\left[ln\left(\frac{2l}{d}\right) - 1 + \frac{d}{l}\right] \quad (6)$$

The general formula for the mutual inductance between two parallel filaments as shown in Figure 10 is as follows:

$$M = 0.001\left[\begin{array}{l} \alpha\, sinh^{-1}\frac{\alpha}{d} - \beta\, sinh^{-1}\frac{\beta}{d} - \gamma\, sinh^{-1}\frac{\gamma}{d} + \delta\, sinh^{-1}\frac{\delta}{d} \\ -\sqrt{\alpha^2+d^2} + \sqrt{\beta^2+d^2} + \sqrt{\gamma^2+d^2} - \sqrt{\delta^2+d^2} \end{array}\right] \quad (7)$$

where $\alpha = l + m - \delta$, $\beta = l - \delta$, $\gamma = m + \delta$.

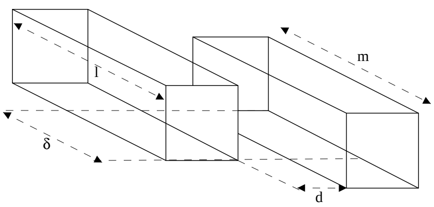

Fig. 10. Two parallel filaments.

III. Experimental Results

The methods discussed above have been implemented and executed on an Alpha 21264 667 MHz dual processor system with 2GB of memory. The capacitance extracted

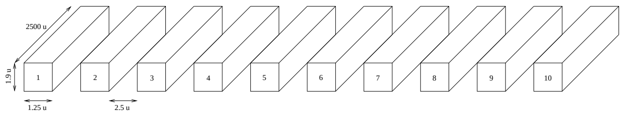

Fig. 11. All conductors are 2500 μ long, 1.25 μ wide and 1.9 μ thick. The space between conductors is 2.5 μ.

by our algorithms were compared with results reported by FastCap. Figure 11 shows the experimental setup for our test case with 10 conductors. The space between adjacent conductors is 2.5 μ. The length, width and height of each conductor are 2000 μ, 1.25 μ and 1.9 μ, respectively. The capacitance extraction results from FastCap and our methods are shown in Table V and Table IV, respectively. Each bus in the test example was scaled by a factor of 10^6 to help FastCap converged. Figure 12 and 13 show the runtime and memory usage comparison between FastCap with hierarchical method with various enhancement in the case when 10 to 50 conductors are presented. It shows that the combination of windowing and supernode methods can effectively help the hierarchical method maintaining linear runtime with respect to the the number of conductors.

Algorithm	**Speedup**	**Memory Usage**
FastCAP	1x – 1x	1 x – 1 x
Hierarchical	2x – 3x	1/6 x – 1/8 x
+ supernodes	3x – 4x	1/8 x – 1/10 x
+ windowing	5x – 9x	1/15 x – 1/20 x
+ windowing and supernodes	10x – 12x	1/20 x – 1/22 x

TABLE I

Runtime and memory improvement by various techniques.

Conductors	Memory Used (in MB)				
	FastCap	Hier	Win	Super	Super+Win
10	1200	103	15	40	7
20	1400	245	35	113	20
30	1550	403	54	262	37
40	1700	562	98	401	65
50	1900	720	120	559	101
60	2100	907	157	695	120
70		1126	174	857	146
80			213	1016	168
90			267	1250	189
100			311	1521	211

TABLE II

Memory Usage Comparison

References

[1] Roger F. Harrington, *Field Computation by Moment Methods*, IEEE Press, 1993.

[2] Weiping Shi, Jianguo Liu, Naveen Kakani, Tiejun Yu, "A fast

Conductors	Time (in Seconds)				
	FastCap	Hier	Wind	Super	Super+Wind
10	80	20	8	10	4
20	247	138	45	73	27
30	630	401	136	248	68
40	1129	901	294	478	159
50	2232	1139	424	756	235
60	4354	1910	569	979	314
70		2439	756	1207	395
80			924	1478	491
90			1202	1921	590
100			1470	2391	697

TABLE III

RUNTIME COMPARISON

67.4									
-47.36	101.9								
-6.90	-44.2	102.3							
-0.61	-7.26	-43.99	102.4						
-4.74	-2.55	-5.12	-43.92	102.4					
-7.27	-0.94	-3.46	-3.08	-43.92	102.4				
-8.04	-11.68	-1.88	-3.43	-3.09	-43.93	102.4			
-3.92	-0.54	-11.63	-1.87	-1.47	-6.13	-43.99	102.3		
-0.91	-2.49	-2.52	-1.64	-0.96	-3.55	-3.26	-44.2	101.9	
-1.75	-2.90	-2.96	-7.12	-7.21	-5.71	-0.26	-7.89	-47.4	67.4

TABLE IV

OUR CAPACITANCE VALUES FOR 10 CONDUCTORS

hierarchical algorithm for 3-D capacitance extraction", *Proc. 35th DAC*, June 1998, pp. 212 – 217

[3] Michael Beattie, Lawrence Pileggi, "Error Bounds for Capacitance Extraction via Window Techniques", *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, Volume 18, Issue 3, March 1999, pp. 311 – 321

[4] Albert Ruehli, Pierce Brennan, "Efficient Capacitance Calculations for Three-Dimensional Multiconductor Systems", *IEEE Transactions on Microwave Theory and Techniques*, Feb. 1973, pp. 76 – 82.

[5] Hansruedi Heeb, Albert Ruehli, "Three-Dimensional Interconnect Analysis Using Partial Element Equivalent Circuits", *IEEE Transactions on Circuits and Systems*, Volume 39, Issue 11, Nov. 1992, pp. 66 – 72.

[6] K. Nabors, J. White, "FastCap: A Multipole-Accelerated 3-D Capacitance Extraction Program", *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 10, Nov. 1991, pp. 1447 – 1459.

[7] Michael Beattie, Lawrence Pileggi, "Electromagnetic Parasitic Extraction via a Multipole Method with Hierarchical Refinement", *IEEE Transactions on Computer-Aided Design*, Nov. 1999, pp. 437 – 444.

68.16									
-45.86	102.99								
-6.85	-42.50	103.93							
-0.10	-7.34	-42.59	104.23						
-4.74	3.41	-6.55	-42.41	103.56					
-7.32	-0.55	3.78	-6.89	-40.83	103.56				
-8.01	-12.48	-1.59	3.54	-6.89	-42.41	104.25			
-3.75	10.35	-11.94	-1.55	3.70	-6.43	-42.69	104.18		
-2.72	-2.54	10.30	-12.45	-0.60	3.51	-7.50	-42.29	103.24	
-2.5	-2.94	-3.89	7.99	-7.23	-5.04	0.25	-7.54	-46.56	73.09

TABLE V

FASTCAP CAPACITANCE VALUES FOR 10 CONDUCTORS

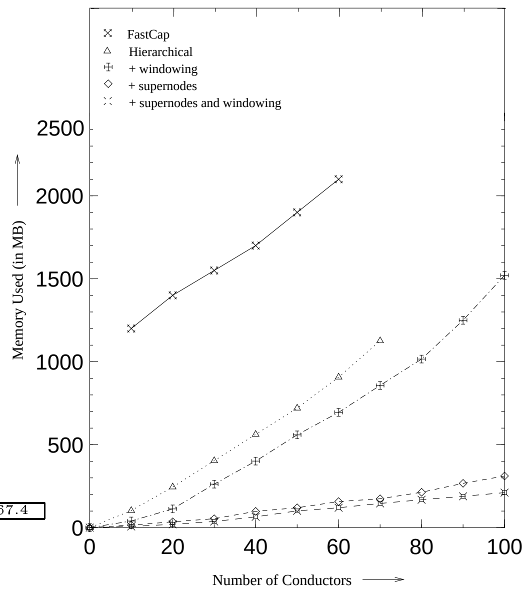

Fig. 12. Runtime comparing between FastCap and hierarchical method with various improvements.

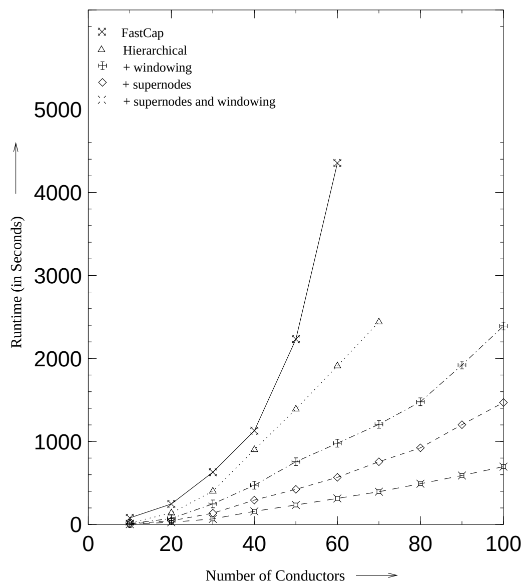

Fig. 13. Memory usage plots comparing between FastCap and hierarchical method with various improvements.