Network Partitioning for Real-time Power System Simulation

P. Zhang, *Student member, IEEE*, J. R. Martí, *Fellow, IEEE,* H. W. Dommel, *Fellow, IEEE*

Abstract-- This paper analyzes the computational complexity of power system elements and the interlink communication costs for the OVNI real-time simulator. Data dependencies in the system simulation are modelled as a weighted graph. A multilevel graph partitioning method is then used to divide the computation among the PC cluster processors. For a given power system network, the partitioning program can generate an optimal partitioning strategy in the context of the OVNI simulator. Partitioning results for the IEEE 118-bus test system are discussed in this paper. The partitioning strategy with an optimal number of partitions guarantees minimum overall computational load and achieves the fastest simulation speed.

Keywords: graph partitioning, processor load balancing, OVNI, real-time simulation

I. Introduction

UBC's Object Virtual Network Integrator (OVNI) real time simulator [1] is based on the MATE (Multi-Area Thevenin Equivalent) concept [2]. As a general circuit simulation algorithm, MATE extends the concepts of Multinode Thevenin Equivalents [3], Diakoptics and MNA. Applying the MATE concept to the EMTP formulas, the algorithm for the real time power system simulator can be briefly formulated as follows.

Assume an electrical network consists of subsystems $[\boldsymbol{A}_1]$, $[\boldsymbol{A}_2]$,… $[\boldsymbol{A}_n]$, which are connected by a vector of link branches $\boldsymbol{\alpha}$. Then the MATE equation set for the system is

$$\begin{bmatrix} \boldsymbol{A}_1 & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{p}_1 \\ \mathbf{0} & \boldsymbol{A}_2 & \cdots & \mathbf{0} & \boldsymbol{p}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{A}_n & \boldsymbol{p}_n \\ \boldsymbol{p}_1^t & \boldsymbol{p}_2^t & \cdots & \boldsymbol{p}_n^t & -\boldsymbol{z} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_{A_1} \\ \boldsymbol{v}_{A_2} \\ \vdots \\ \boldsymbol{v}_{A_n} \\ \boldsymbol{i}_\alpha \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_{A_1} \\ \boldsymbol{h}_{A_2} \\ \vdots \\ \boldsymbol{h}_{A_n} \\ \mathbf{0} \end{bmatrix} \quad (1)$$

where

$\boldsymbol{A}_i$ = admittance matrix of subsystem $[\boldsymbol{A}_i]$ (i=1,2,…,n)

$\boldsymbol{p}_i$ = link current injection matrix of subsystem $[\boldsymbol{A}_i]$

$\boldsymbol{h}_{Ai}$ = history terms of subsystem $[\boldsymbol{A}_i]$

$\boldsymbol{V}_{Ai}$ = nodal voltages of subsystem $[\boldsymbol{A}_i]$

$\boldsymbol{i}_\alpha$ = current vector in link branches $\boldsymbol{\alpha}$

$\boldsymbol{z}$ = impedance matrix of branches $\boldsymbol{\alpha}$

Equation set (1) is then rewritten as

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{q}_1 \\ \mathbf{0} & \mathbf{1} & \cdots & \mathbf{0} & \boldsymbol{q}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{1} & \boldsymbol{q}_n \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{z}_\alpha \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_{A_1} \\ \boldsymbol{v}_{A_2} \\ \vdots \\ \boldsymbol{v}_{A_n} \\ \boldsymbol{i}_\alpha \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_{A_1} \\ \boldsymbol{e}_{A_2} \\ \vdots \\ \boldsymbol{e}_{A_n} \\ \boldsymbol{e}_\alpha \end{bmatrix} \quad (2)$$

where

$$\boldsymbol{q}_i = \boldsymbol{A}_i^{-1}\boldsymbol{p}_i$$

$$\boldsymbol{e}_{A_i} = \boldsymbol{A}_i^{-1}\boldsymbol{h}_{A_i}$$

$$\boldsymbol{z}_\alpha = \sum_i \boldsymbol{p}_i^t \boldsymbol{q}_i + \boldsymbol{z}$$

$$\boldsymbol{e}_\alpha = \sum_i \boldsymbol{p}_i^t \boldsymbol{e}_{A_i}$$

As shown in (2), the MATE algorithm perfectly matches the concept of a cluster computing system, which consists of a collection of interconnected stand-alone PCs working together as an integrated real time computing resource. For instance, the master unit in a PC cluster can be in charge of receiving the updated history terms from the slave units, solving the link equations and distributing the links solution to the slave nodes. Each subsystem solver (slave units) calculates its part of the network solution, updates its history terms and sends them back to the mater unit.

Obviously, in order to efficiently explore real-time power system simulation capability of a PC cluster, it is an essential step to appropriately partition the power system so as to map the computational load onto the processors. In the context of the MATE solution algorithm of UBC's OVNI real-time simulator, a good partition scheme should distribute to each processor a roughly equal number of computations while, at the same time, minimizing the number of link computations and the amount of inter-processor communication time between slave and master nodes.

Graph models can be used to relate data dependencies in the power system simulation. Graph partitioning can then be used to divide the computation among the PC cluster processors.

As a technique for balancing the load and minimizing

P. Zhang is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, B.C. V6T 1Z4 CA (e-mail: pzhang@ece.ubc.ca).

J. R. Martí is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, B.C. V6T 1Z4 CA (e-mail: jrms@ece.ubc.ca).

H. W. Dommel is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, B.C. V6T 1Z4 CA (e-mail: hermannd@ece.ubc.ca).

Presented at the International Conference on Power Systems Transients (IPST'05) in Montreal, Canada on June 19-23, 2005
Paper No. IPST05 - 177

communication among processors when executing a set of tasks in parallel, graph partitioning is a NP-complete problem [4]. Due to this fact, almost all practical approaches to this problem are heuristics-based. The existing methods can be classified into categories, such as geometric [4], combinatorial and mathematical programming [5, 6], spectral techniques [7, 8], and multilevel methods [9-11]. Among these, the multilevel methods, which can perform high quality partitioning, are the fastest methods so far. In this paper a multilevel recursive bisection approach is chosen as our solution method. Therefore, the emphasis in this paper lies not on the graph partitioning method itself but on the computational complexity and communication cost analysis of our real time simulator.

II. Element Computations

This section analyzes the computational complexity of power system elements and MATE for the real time simulation. We assume that the MATE algorithm is based on the trapezoidal rule.

In this paper, the computational complexity for updating the history terms in the OVNI simulator is quantified by the concept of a *flop*. As defined in [12], a flop constitutes the effort of doing a floating point add, a floating point multiply and a little subscripting. The following describes the way to calculate the number of flops for an element to update its history terms.

A. M Phase Nominal Π-Circuit

(1) Series connection of coupled R and L

Suppose R and L are symmetrical. Fig. 1 illustrates a 3-phase nominal Π-circuit, where

$$\boldsymbol{R}=\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}, \boldsymbol{L}=\begin{bmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{bmatrix}.$$

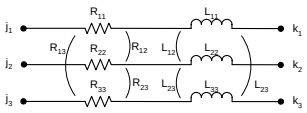

Fig. 1 Three-phase Nominal Π-Circuit

By using the Trapezoidal rule, the discrete time equation of the m-phase R-L circuit is [3]

$$\boldsymbol{i}_{jk}(t) = \boldsymbol{G}_{RL}[\boldsymbol{u}_j(t) - \boldsymbol{u}_k(t)] - \boldsymbol{h}_{RL}(t) \quad (3)$$

where the history term is

$$\boldsymbol{h}_{RL}(t) = -(\boldsymbol{I} - \boldsymbol{G}_{RL}\boldsymbol{R})\,\boldsymbol{h}_{RL}(t-\Delta t) - 2(\boldsymbol{G}_{RL} - \boldsymbol{G}_{RL}\boldsymbol{R}\boldsymbol{G}_{RL})[\,\boldsymbol{u}_j(t-\Delta t) - \boldsymbol{u}_k(t-\Delta t)] \quad (4)$$

and the equivalent resistance is

$$\boldsymbol{G}_{RL} = \frac{\Delta t}{2}\left[\boldsymbol{I} + \frac{\Delta t}{2}\boldsymbol{L}^{-1}\boldsymbol{R}\right]^{-1}\boldsymbol{L}^{-1} \quad (5)$$

According to (4), the number of flops for updating the history terms of the *R-L* circuit is $2(m^2+m)$. If $m = 3$, then the number of flops is 24.

(2) Coupled Capacitances

By using the Trapezoidal rule, the discrete time equation of an *m*-phase *C* branch is

$$\boldsymbol{i}_j(t) = \boldsymbol{G}_C\boldsymbol{u}_j(t) - \boldsymbol{h}_C(t) \quad (6)$$

where the history term is

$$\boldsymbol{h}_C(t) = \boldsymbol{h}_C(t-\Delta t) + 2\boldsymbol{G}_C\,\boldsymbol{u}_j(t-\Delta t) \quad (7)$$

and the equivalent resistance is

$$\boldsymbol{G}_C = \frac{2}{\Delta t}\boldsymbol{C} \quad (8)$$

Then the number of flops for updating history terms of the *C* circuit is (m^2+m). If $m = 3$, the number of flops is 12.

For an *m*-phase Π-circuit consists of *R*, *L* and *C* matrices, the total number of flops is $3(m^2+m)$.

B. Transformers

If we neglect the core saturation effects, transformers can be represented in the form of the branch resistance and inductance matrix in Fig.1 [3].

Therefore, for a three phase three winding transformer, the number of flops for updating the history terms is $2(9m^2+3m) = 180$ with $m=3$. There are the same number of flops for a three phase autotransformer.

If the element is a three-phase two-winding transformer, then the number of flops is $2(4m^2+2m) = 84$.

Here the formulas of the discrete time circuit equivalent are omitted due to space limits.

C. Synchronous Machine

The equivalent equations of the electrical part of the phase-coordinates three-phase synchronous machine model used in OVNI [13] are given below

$$\boldsymbol{v}_s(t) = -\boldsymbol{R}_{eq}(t)\,\boldsymbol{i}_s(t) + \boldsymbol{e}_s(t) \quad (9)$$

where

$$\boldsymbol{R}_{eq}(t) = \frac{2}{\Delta t}\{\,\boldsymbol{R}_1(t) - \boldsymbol{R}_2(t)\,[\boldsymbol{R}_4(t)]^{-1}\,\boldsymbol{R}_3(t)\,\} \quad (10)$$

$$\boldsymbol{e}_s(t) = \boldsymbol{R}_2(t)[\boldsymbol{R}_4(t)]^{-1}[\boldsymbol{v}_r(t) - \boldsymbol{e}_{rh}(t)] + \boldsymbol{e}_{sh}(t) \quad (11)$$

The $\boldsymbol{R}_1(t)$~ $\boldsymbol{R}_4(t)$ are 3×3 matrices. The physical meaning of these matrices can be found in [13]. These matrices can be calculated from the linear interpolation of a series of pre-calculated lookup tables, which requires $4m^2$ flops of calculation.

An efficient method to calculate a matrix inverse is the Gaussian elimination in Chapter 4 of [12]. The number of flops needed of this method is $[(m-1)\,m^2 - (m-1)^2\,m + m^2 + (m-1)^3/3]$, where $m=3$ is the dimension. Taking into account the effort to calculate the $\boldsymbol{e}_{rh}(t)$, $\boldsymbol{e}_{sh}(t)$ and the matrix/vector products/adds, the number of flops for updating the history terms is $[(m-1)\,m^2 - (m-1)^2\,m + (m-1)^3/3 + m^2] + 4m^2 + m^3 + 2(m+1)\,m + m + m^2 + m$.

Different from the non-rotating element, the $\boldsymbol{R}_{eq}(t)$ changes at each time step. The number of flops for updating the equivalent resistances is $[(m-1)\,m^2 - (m-1)^2\,m + (m-1)^3/3 + m^2] + 2m^3 + m^2$. Thus the total number of flops for electrical part is the sum of these two numbers.

The equivalent equations for the mechanical part are

$$\boldsymbol{\omega}_m(t) = \boldsymbol{A}^{-1}(t)\boldsymbol{C}(t) \quad (12)$$

with

$$\boldsymbol{A} = \boldsymbol{J}_m + \frac{2}{\Delta t}\boldsymbol{D}_m(t) + \left(\frac{2}{\Delta t}\right)^2 \boldsymbol{K}_m(t) \tag{13}$$

$$\boldsymbol{B}(t) = \frac{\Delta t}{2}\boldsymbol{T}(t) - [\boldsymbol{J}_m - \frac{2}{\Delta t}\boldsymbol{D}_m(t) - \left(\frac{2}{\Delta t}\right)^2 \boldsymbol{K}_m(t)] \tag{14}$$

$$\boldsymbol{C}(t) = \boldsymbol{B}(t)\omega_m(t-\Delta t) + \Delta t \boldsymbol{K}_m(t)\theta_m(t-\Delta t) + \frac{\Delta t}{2}\boldsymbol{T}(t-\Delta t) \tag{15}$$

The total number of flops for the mechanical part is $[(m-1)\, m^2 - (m-1)^2\, m + (m-1)^3/3 + m^2] + 6m$.

For any other power system elements, for instance the induction machine, the number of flops for updating the history terms can be estimated in the same way above by examining the discrete time equivalent circuit of the element [14].

III. Network Partitioning Method

As is discussed before, the goals of network partitioning during real time power system simulation with the OVNI simulator are as follows: 1) The computational load among processors should be balanced and, if necessary, the memory requirement of each PC should also be balanced; 2) The communication cost of different processors should also be roughly equal; 3) The number of links should be minimized.

It has been found in [1] that the inherent needed communication time of a solver node is determined by the number of link branches between this node and its neighboring solver nodes. The communication time will be minimum if the number of links is minimized. Hence items 2) and 3) above become one. From a mathematical programming point of view, the third item can be considered as the *objective* and the first item can be considered as the *constraint*.

It can be clearly seen that our network-partitioning problem for the OVNI simulator can be exactly modeled as a graph-partitioning problem as below.

1) We can consider a power system element (such as a transmission line, generator, transformer, load, etc.) as a weighted vertex v in a graph. Each vertex (element) v has a weight vector $\boldsymbol{w}^v$ of size 2. Conceptually, the $w_1{}^v$ can be used to represent the number of computations for updating the history terms at each time step; $w_2{}^v$ can be set to 1 for each vertex. Let us explain the second weight in detail.

As it is known, the main computational load of a subsystem solver (slave units) can be split into two parts. One part is the computational load for updating the history terms, which corresponds to $w_1{}^v$. The other part is solving the subsystem nodal equations. We will not calculate the number of flops of this part of the computational load explicitly. Instead, we will do it implicitly by setting $w_2{}^v = 1$ for each vertex. If the second weight is perfectly balanced in the network partitioning, then it means that the dimension of each subsystem nodal equations is well balanced. Therefore, the computational load for solving the subsystem nodal equations will also be balanced among the slave processors.

2) The physical links between elements can be modeled as an edge. Conceptually, the weight of the each edge can be set to p, where p is the number of phases of the link branch. Thus, the objective of the graph partitioning becomes to minimize the total number of links of the real time simulator.

Figure 2 illustrates the transformation of the IEEE 30-bus system into a weighted graph.

The graph partitioning problem is then defined [11] as follows.

Consider a graph $G(V, E)$ with $|V| = n$, where V is a set of weighted vertices such that each vertex $v \in V$ has a weight vector $\boldsymbol{w}^v = (w_1{}^v, w_2{}^v)$ and E is a set of scalar-weighted edges. Suppose the weight vectors of the vertices are normalized such that $\sum_v w_i^v = 1.0$ for $i = 1, 2$. Given a positive integer k, find k subsets (partitions) $V_1, V_2, \ldots, V_k$ of V satisfying $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \Phi$ for $i \neq j$, such that

1) The edge-cut, i.e. the sum of the edge-weights whose incident vertices belong to different subsets, is minimized subject to the constraint

2) $\forall i,\ l_i \le c_i$

where l_i is the load imbalance factor with respect to the i^{th} weight

$$l_i = k \max_j \left(\sum_{\forall v \in V_j} w_i^v \right) \tag{16}$$

c_i (≥ 1.0) can be set close to 1.0 to guarantee a perfect load balancing.

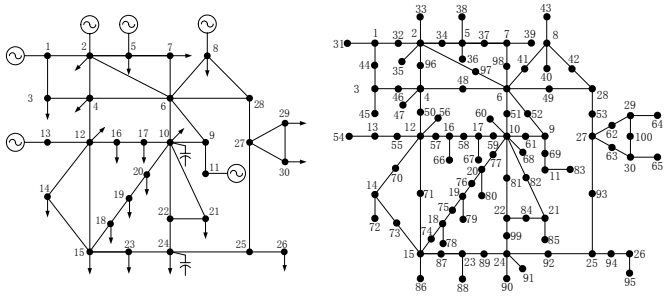

Fig. 2 IEEE 30-bus System and Its Graph Description

This multi-constraint graph partitioning problem can be solved by a multilevel recursive bisection method [11]. After recursively calling the bipartitioning functions for about $log_2 k$ times, the original graph G_0 is divided into k parts. The recursive bisection tree for a 16-way partitioning is shown in Fig.3, where the nodes except for the leaves represent the bipartitioning of a graph (subgraph). Each node processing includes three stages: graph coarsening, initial partitioning and refinement.

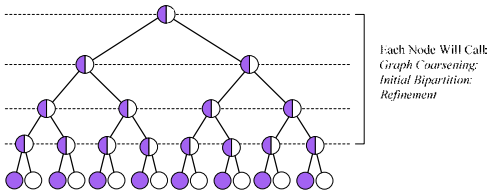

Fig. 3 Recursive Bisection Tree

A. Graph Coarsening

First normalize the vertex weights for the graph to partition.

Graph coarsening of an original graph G_0 means transforming G_0 into a coarser graph G_1, by collapsing together selected vertices of G_0. Taking the coarser graph as the original graph and iteratively call the coarsening routine, a series of graphs G_2, G_3,…,$G_{c\text{-}1}$, are constructed until a sufficiently small graph G_c is obtained. An efficient method suitable for multi-constraint partitioning is the combination of the heavy edge matching (HEM) and balancing edge heuristic [10].

B. Initial Bipartitioning

Let $G_c = (V, E)$ is the coarsest graph to be partitioned into two balanced subgraphs $G_A= (V_A, E_A)$ and $G_B= (V_B, E_B)$. The bipartition algorithm proceeds as follows.

1) Select a vertex $v \in V$ randomly, set $V_A= \{v\}$, $V_B= V \backslash V_A$. Create two FIFO priority queues;

2) All vertices $u \in V_B$ are inserted into the two queues according to the following rule: u belongs to the j^{th} queue if $w_j^u = \max_i w_i^u$ $(i, j \in \{1, 2\})$;

3) If $w_j^{V_B} = \max_i w_i^{V_B}$, then the j^{th} queue is selected. If the j^{th} queue is empty, choose the nonempty queue corresponding to the second heaviest weight (in this 2-constraint case, simply choose another queue). Move the top vertex in the j^{th} queue to G_A;

4) Repeat 3) until one of the weights of G_A, say the i^{th} weights, satisfies $\sum_{u\in V_A} w_i^u \ge \frac{1}{2}\sum_{u\in V} w_i^u$.

C. Uncoarsening and Refinement

The concept of the gain of a vertex $v \in V$ is first introduced. The *gain* g^v is the reduction on the edge-cut if v is moved from one partition V_i to another

$$g^v = \sum_{(u,v)\in E \wedge u\notin V_i} w^{(u,v)} - \sum_{(u,v)\in E \wedge u\in V_i} w^{(u,v)} \qquad (17)$$

Also define the *boundary* of a partition V_i as

$$B^{V_i} = \{v \mid v \in V_i \wedge (u,v) \in E \wedge u \notin V_i\} \qquad (18)$$

At this stage, the coarser graph G_c is projected back to the original graph G_0 by going through the intermediate graphs $G_{c\text{-}1}$, $G_{c\text{-}2}$,…, G_1. During each projecting step, two refinement algorithms are called. Here assume the intermediate graph to be refined is $G_f (V_f, E_f)$.

a. Weight Balancing Algorithm

1) Create 2 FIFO queues for each of the 2 partitions. All the vertices in a partition are inserted into the 2 queues for this partition according to the same rule as step 2) in stage B.

2) If $w_k^{V_l} = \max_i w_i^{V_j}$, $i \in \{1,2\}$, $j \in \{A, B\}$, then the k^{th} queue for partition V_l is selected. Pick a vertex from this queue and move it to another partition, such that the two partitions are best balanced. If the selected queue is empty, then select the nonempty queue corresponding to the second largest weight in the same partition. The moved vertex is then locked.

3) Iterate step 2) for $|V_f|$ times.

b. Edge-cut Reduction Algorithm

1) Create 2 FIFO queues for each partition. Only the vertices in B^{V_j} ($j \in \{A, B\}$) are inserted into the queues for this partition, according to the same rule as step 2) in stage *B*.

2) If $w_k^{V_l} = \max_i w_i^{V_j}$, $i \in \{1,2\}$, $j \in \{A, B\}$, then the k^{th} queue for partition V_l is selected. Pick a vertex with the largest gain from this queue and move it to another partition. If the selected queue is empty, then select the nonempty queue corresponding to the second largest weight in the same partition. The moved vertex is then locked and the gains of other boundary vertices are updated.

3) Repeat step 2) for N times, say, N =10.

IV. Network Partitioning of the Test System

Based on the computational load analysis of OVNI, the IEEE 118-bus test systems was investigated by using the multilevel network partitioning program.

The load imbalance factor l_i with respect to the i^{th} weights and the edge-cuts are employed to evaluate the partitioning results,

$$l_i = \frac{l_{\max,i}}{l_{av,i}} = \frac{\max_j \left(\sum_{\forall v\in V_j} w_i^v \right)}{\sum_{v\in V} w_i^v / k} \qquad (19)$$

A graph G can have at most $|V|$ partitions. However, if the number of partitions increases, the computational load to solve the link equations in MATE may also increase. The disadvantage due to the increase of link branches may overwhelm the advantage due to the reduction of computation loads on slave processors. The partitioning program is designed to find the optimal number of partitions under these constraints. Assume the computational load on slave processor i is $l_{\text{slave},\,i}$ and the load for solving the link equations is l_{link}. Then the total computational load l_{total} can be estimated as

$$l_{total} = l_{link} + \max_i l_{slave,i} \qquad (20)$$

Perform repeated runs of partitioning and scan the number of partitions, then the optimal number of partitions will be found. The partitioning strategy with an optimal number of

partitions guarantees minimal overall computational load and achieves the fastest simulation speed.

The partitioning results for the IEEE 118-bus system are listed below. Fig.4 illustrates the computational loads with different number of partitions. It is seen that the optimal number of partitions is 5. All the other loads are normalized by this optimal load. Some detailed partitioning data are listed in Table 1. Fig. 5 shows the 5-way partition scheme for the IEEE 118-bus system.

TABLE I
PART OF THE PARTITIONING RESULTS

No. of Partitions	No. of Edge-Cuts	Load Imbalance Factor for Updating the History Terms	Load Imbalance Factor for Solving Subsystems
2	7	1.000579	1.002198
3	9	1.007528	1.002198
4	18	1.002895	1.002198
5	24	1.007528	1.00
6	31	1.011002	1.002198
7	32	1.017371	1.015385
8	36	1.009844	1.019780
⋮	⋮	⋮	⋮
117	269	1.287203	1.285711
118	297	1.298205	1.296703

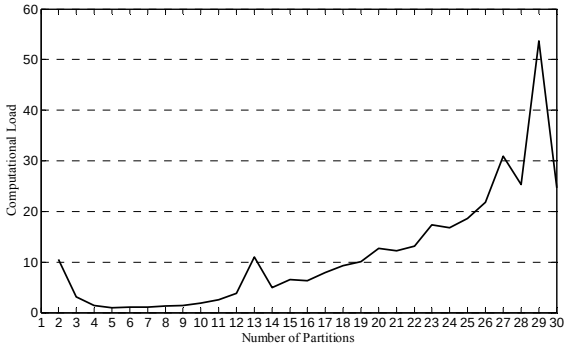

Fig. 4 Normalized Computational Loads with Different Number of Partitions

V. CONCLUSION

Based on the analysis of computational load of power system elements and the interlink communication cost, a multilevel network partitioning method is employed to divide the computation among the PC cluster processors for the OVNI real-time simulator. For a given power system structure, the partitioning program can generate an optimal partitioning strategy in the context of the OVNI simulator. Partitioning results for the IEEE 118-bus test system were discussed in this paper.

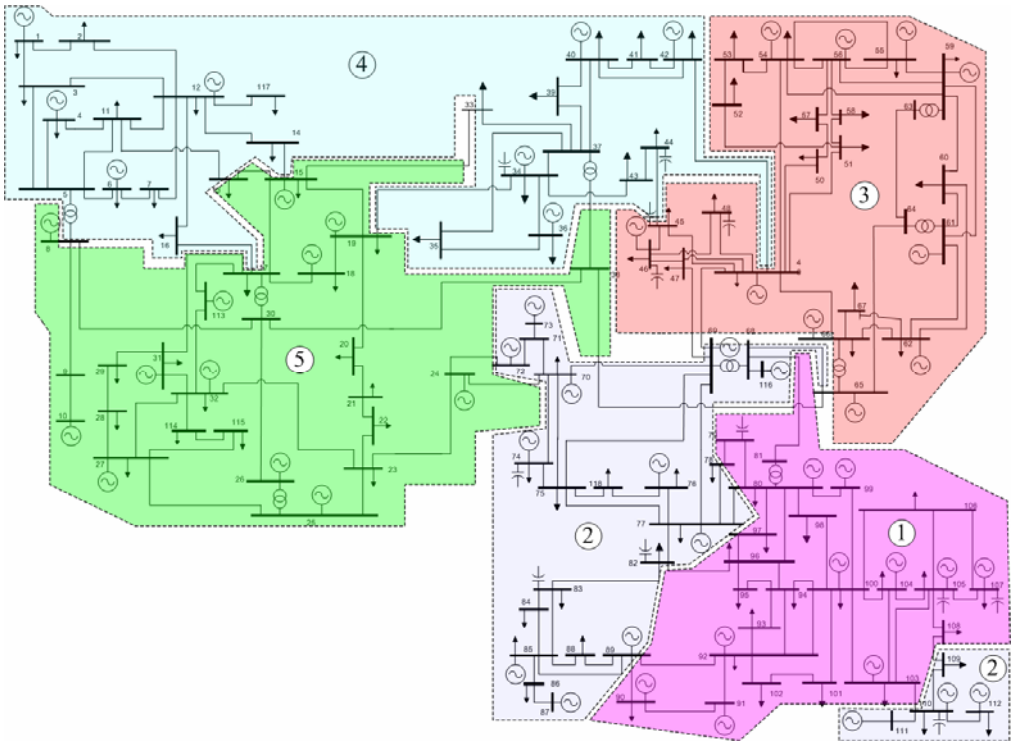

Fig. 5 Network Partitioning Scheme for the IEEE 118-bus System

VI. REFERENCES

[1] J. A. Hollman, J. R. Martí, "Real Time Network Simulation with PC-Clusters," *IEEE Transactions on Power Systems*," vol. 18, no. 2, pp. 563~569, May 2003.

[2] J. R. Martí, L. R. Linares, J. A. Hollman, F. A. Moreira, "OVNI: Integrated software/Hardware Solution for Real-time Simulation of Large Power Systems," *in Proceedings of the PSCC02*, Sevilla, Spain, June, 2002.

[3] H. W. Dommel, *EMTP Theory Book* (*2nd edition*). Vancouver, BC: Micrtran Power System Analysis Corporation, 1996.

[4] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359~392, 1998.

[5] B. W. Kernighan, S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291~307, Feb. 1970.

[6] C. Fiduccia, R. Mattheyses, "A linear-time heuristic for improving network partitions," *Technical Report 82CRD130*, General Electric Co., Corporate Research and Development Center, Schenectady, NY, 1982.

[7] A. Pothen, H. Simon, K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Mat. Anal. Appl.*, vol. 11, pp. 430~452, 1990.

[8] M. Fiedler, "Algebraic Connectivity of Graphs," *Czech. Math. J.*, vol. 23, pp: 298-305, 1973.

[9] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," *Technical Report SAND93-1301*, Sandia National Laboratories, 1993.

[10] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96~129, 1998.

[11] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Proc. Supercompting'98*, Orlando, 1998.

[12] G. H. Golub, C. F. Van Loan, Matrix Computations. Baltimore, Maryland: The Johns Hopkins University Press, 1983.

[13] J. R. Martí, K. W. Louie, "A Phase Domain Synchronous Generator Model Including Saturation Effects," *IEEE Trans. on Power Systems*, vol. 12, no. 1, pp. 222~227, February, 1997.

[14] R. Hung, H. W. Dommel, "Synchronous Machine Models for Simulation of Induction Motor Transients," *IEEE Trans. on Power Systems*, vol. 11, no. 2, pp. 833~838, May 1996.

VII. BIOGRAPHIES

Peng Zhang (S'04) received his B.Sc. and M.Sc. degrees in Electrical Engineering from Shandong University, China in 1996 and 1999, respectively. He is currently pursuing the Ph.D. degree at the University of British Columbia, Vancouver, BC, Canada. His research topics are in power system real-time simulation, stability and control.

José Ramón Martí (S'79–M'80–SM'01–F'02) was born in Lérida, Spain. He received the degree of Electrical Engineer from Central University of Venezuela in 1971, the M.E.E.P.E. degree from Rensselaer Polytechnic Institute in 1974, and the Ph.D. degree from the University of British Columbia in 1981.

He has made contributions to the time-domain modeling of transmission lines, transformers, and electrical machines, and has developed numerical solution techniques for transient simulation programs and real-time applications. He is currently a Professor at the University of British Columbia and a Registered Professional Engineer in British Columbia, Canada.

Hermann W. Dommel (LF'01) was born in Germany in 1933. He received the Dipl.-Ing. And Dr.-Ing. degrees in electrical engineering from the Technical University Munich, Munich, Germany, in 1959 and 1962, respectively. From 1959 to 1966, he was with the Technical University Munich, and from 1966 to 1973, with Bonneville Power Administration, Portland, OR. Since 1973, he has been with the University of British Columbia, Vancouver, BC, Canada.