Matrix-Free Nodal Domain Decomposition With Relaxation For Massively Parallel Finite-Element Computation of EM Apparatus

Peng Liu and Venkata Dinavahi

Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada

In this paper, the nodal domain decomposition with relaxation (NDDR) scheme is proposed to solve the nonlinear finite-element (FE) problem in electromagnetic apparatus without assembling the global system of equations. Each sub-domain contains only one node with unknown magnetic vector potential, and the calculation of each sub-domain can be massively parallelized to utilize the prevalent parallel computing architectures. The sub-domain solver has excellent modularity for single instruction multiple data programming with a specific data structure, and the required memory shows a linear increase with the problem size. The NDDR scheme is implemented on both multi-core CPUs and many-core GPUs, and the accuracy and efficiency are discussed for different problem sizes. Result comparison with a commercial FE package shows a speedup of more than 30 times for a magnetostatic case and an average speedup of more than 53 times for a time-domain nonlinear FE case with different time steps while maintaining an error of less than 0.85%.

***Index Terms*— Distributed algorithms, domain decomposition (DD), electromagnetic (EM) apparatus, finite-element method (FEM), graphics processing units (GPUs), massively parallel, nonlinear system of equations, relaxation.**

I. Introduction

THE finite-element method (FEM) has been dominantly utilized for the modeling and design of electromagnetic apparatus, such as rotating machines and transformers, due to its accuracy and flexibility for complex geometries. The notorious computational burden, stemming from repeatedly factorizing the updated Jacobian matrix, demands an exploration of efficient nonlinear FE solvers for conveniently designing and testing electromagnetic apparatus.

The prevalent trend in high-performance computing resources is to increase the number of cores massively either as clusters of multi-core central processing units (CPUs) or many-core graphics processing units (GPUs); for example, the recently released NVIDIA Tesla V100 accelerator card is equipped with 5120 Cuda cores and 16 GB of HBM2 memory [1], and it has motivated engineers to explore suitable parallel algorithms to make full use of its compute power. In the past, a GPU-accelerated conjugate gradient solver [2]–[5] and the domain decomposition (DD) method on CPUs [6], [7] have been implemented to improve the efficiency of the electromagnetic field computation. However, a massive parallelism could be hardly achieved, since the *assemble and then solve* procedure in a traditional FE technique is essentially a centralized way of thinking. On the other hand, with the purpose of fully exploring the computational power of GPUs, a decentralized thinking pattern resulting in massive parallelism would be greatly promising. For example, the elementwise FE technique was proposed for matrix-vector multiplication with independent computation at the triangular element level [8], [9] and was then developed to solve a linear FE problem on a GPU architecture [10], [11]. In addition, the Jacobi-based GPU solver has also been explored for the computation of Poisson's equation [12]. Although deficient to handle nonlinear FE problems, the massive parallelism and a decent speedup are achieved on GPUs due to the divide-and-conquer strategy. Massively parallel simulations have also already been investigated for large-scale power system dynamic and transient simulation applications [13]–[18].

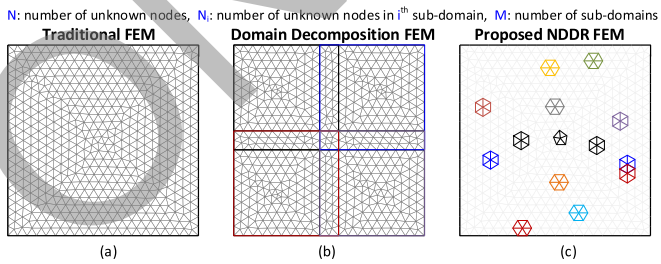

Fig. 1. Illustration of the traditional FEM, the DD-based FEM, and the proposed NDDR-based FEM. (a) Centralized ($M = 1$). Each node must be solved within the $N \times N$ system (no parallelism). (b) Partially Decentralized ($M = 4$). Each node must be solved within its $N_i \times N_i$ sub-system (partially parallelized). (c) Fully decentralized ($M = N$). Each node must be solved within its 1×1 sub-system (massively parallelized).

In this paper, the nodal DD with relaxation (NDDR) is proposed to explore the massive parallelism in FE computation of both linear and nonlinear problems. This novel idea is inspired by the following question regarding the extreme partition of a domain: *what if a sub-domain contains only one unknown node?*

As shown in Fig. 1, the proposed NDDR has the following features compared with the traditional nonlinear FE solver based on the Newton–Raphson (NR) algorithm.

Manuscript received March 23, 2018; revised May 16, 2018 and June 12, 2018; accepted June 13, 2018. Date of publication July 11, 2018; date of current version August 17, 2018. Corresponding author: P. Liu (e-mail: pliu3@ualberta.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2018.2848622

1) There exists only one unknown in each sub-domain, and thus, no matrices are necessary, i.e., the NDDR is matrix-free and memory-friendly.
2) Instead of applying the NR algorithm to a global system, which is sequential in nature, the problem is solved, utilizing a neat relaxation scheme by iteratively updating each node (sub-domain) in a massively parallel manner.
3) Each sub-domain is solved independently following the same pattern, and the computation is concise since there is only one unknown. Therefore, the NDDR shows perfect modularity for Kernel programming on GPU architectures.
4) A specific data structure is required for the single instruction multiple data (SIMD) programming on GPU.

The matrix-free feature stems from the fact that the more number of sub-domains, the fewer the number of unknowns required to be solved for each sub-domain. It is also worth mentioning that for a linear problem, both the NDDR scheme and the N-scheme in [10] using the weighted summation of elemental contributions are equivalent to the Jacobi iterative scheme. However, the proposed NDDR scheme can be easily extended for nonlinear problems; in this sense, the NDDR scheme better reveals the DD advantage regardless of the linearity or nonlinearity of the problem. In addition, the massively parallel computing resource (either large-scale CPU or GPU clusters) that matches the FE problem size is a prerequisite to explore the full potential of the NDDR scheme, especially for large-scale FE problems.

This paper is organized as follows. Section II briefly introduces the FE equations of general EM apparatus with the nonlinear B–H curve. Then, Section III presents the methodology of the NDDR and the supporting data structure. In Section IV, case studies employing the NDDR scheme are implemented on both CPUs and GPUs, and the results are compared with the commercial FE package Comsol Multiphysics with regard to the accuracy and speedup. At last, Section V gives the conclusion.

II. Finite-Element Equations of EM Apparatus

Governed by Ampere's law, a 2-D magnetostatic problem can be described by the following equation:

$$\nabla \cdot (\upsilon \nabla A) = -J \tag{1}$$

where A is the z-component of the magnetic vector potential to be solved, υ is the material reluctivity, and J is the z-component of the impressed current density.

The problem can be solved by the well-known Galerkin FEM [19], and the general steps are discretizing the domain, forming elemental equations, assembling, applying boundary conditions, and solving. The product of the weight function and the residual is integrated over each triangular element (see Fig. 2). and the following elemental equations can be obtained by forcing the integral to be zero:

$$\frac{\upsilon^e}{4\Delta^e}\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}\begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \frac{J^e \Delta^e}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{2}$$

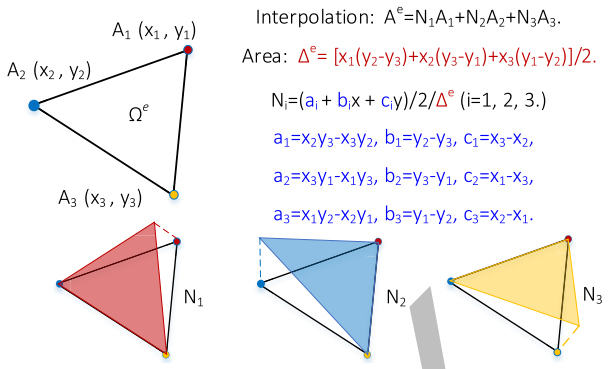

Fig. 2. Triangular element and the interpolation (weight) function for the Galerkin FEM.

where

$$k_{11} = b_1b_1 + c_1c_1, \quad k_{12} = k_{21} = b_1b_2 + c_1c_2$$
$$k_{22} = b_2b_2 + c_2c_2, \quad k_{23} = k_{32} = b_2b_3 + c_2c_3$$
$$k_{33} = b_3b_3 + c_3c_3, \quad k_{31} = k_{13} = b_1b_3 + c_1c_3.$$

For ferromagnetic triangular elements, the reluctivity υ^e depends on the strength of the unknown magnetic vector potential, implying that the problem is nonlinear. For the nonlinear solution, the calculation of the Jacobian matrix for (2) is required using the NR scheme. To obtain $(\partial \upsilon^e/\partial A_i)$, the following chain rule in partial differentiation is usually utilized [20]:

$$\frac{\partial \upsilon^e}{\partial A_i} = \frac{\partial \upsilon^e}{\partial B^2}\frac{\partial B^2}{\partial A_i} \quad (i = 1, 2, 3). \tag{3}$$

The $\partial \upsilon^e/\partial B^2$ can be obtained from the nonlinear B–H representation, while $\partial B^2/\partial A_i$ is derived using the definition of the magnetic flux density: $B = \nabla \times A$.

For the traditional nonlinear FE solver, all the elemental equations are assembled to form a global sparse system and then solved with some iterative schemes, such as the NR technique. Therefore, large matrices and efficient sparse solvers are always inevitable.

In the proposed NDDR scheme, the nonlinear system is solved in a totally decentralized manner.

III. Nodal Domain Decomposition With Relaxation

The traditional Schwartz DD [21] divides the whole domain into several sub-domains, and each sub-domain can be solved independently. The information exchange between sub-domains is implemented by subtly manipulating the boundary conditions (see Fig. 3), and an iterative scheme is required to achieve the steady state for final solutions since the values of the inner boundaries are unknown and are usually assigned guessed values.

As shown in Fig. 3, in each sub-domain, the inner nodes are solved based on the global boundary nodes and the neighboring boundary nodes from other sub-domain, and they also serve as the boundary nodes for other sub-domain. Although this neighboring information is not always correct,

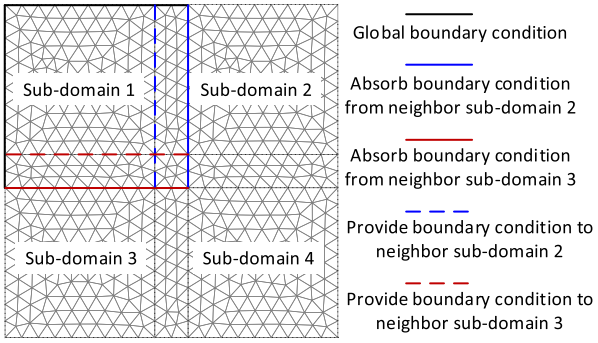

Fig. 3. Information exchange in a sub-domain for overlapping Schwartz DD.

if all sub-domains work together repeatedly at the same time, the information exchange becomes effective to converge to the final solution. In this sense, the DD method is a *relaxation* scheme.

Note that the term *relaxation* describes the iterative nature of the solution process, namely, the independent sub-domain solver and the repeated data communication between sub-domains.

Despite the necessity of iteration, the reduced problem size and the parallelism have made the DD method beneficial for parallel computing architectures, such as multi-core CPUs. The number of nodes in each sub-domain, which determines the matrix size of the sub-problem, depends on the domain partition. It is very common that each sub-domain can still contain more than hundreds of nodes for a medium size FE problem [6], [7], and only partial parallelism can be achieved. However, from the perspective of a Cuda core on a GPU, which is suitable for massive parallelism, an ideal sub-domain should contain as few nodes as possible.

With the relaxation scheme, an extreme situation where each sub-domain contains only one inner node is considered. As shown in Figs. 1(c) and 4, each non-boundary node and its direct neighbors make up a sub-domain. Applying the neighbors' values from the previous iteration as boundary conditions, the value of the inner node can be updated, which is a miniature FE problem. Due to the overlapping, each node is updated based on its direct neighbors and, meanwhile, serves as boundary conditions when its neighbors are updated. Thus, each inner node is updated independently, and the steady state can be reached in an iterative manner, which is perfect for massively parallel architectures.

At first glance, solving the miniature FE problem in Fig. 4 is to solve a 7×7 nonlinear system, but a closer inspection reveals that it is actually a 1×1 nonlinear system because the other equations are overwritten by the imposed boundary conditions.

As shown in Fig. 4, all the neighboring elements Ω_{Ki} $(i = 1, 2, 3 \ldots)$ contribute to the solution of the inner node A_K, whereas not all the elemental equations are useful depending on the element-node numbering scheme. For example, for element Ω_{K1}, the node A_K to be solved is numbered 1, and A_2 and A_3 are given as boundary conditions, and thus, only the first elemental equation (highlighted in blue) is valid for the solution of A_K, since the other two equations will be overwritten. Similar things occur in the other neighboring elements; therefore, the following equations considering all the neighboring elements need to be solved for A_K:

$$\sum_{i=1}^{N} F_{Ki}(A_K) = 0 \tag{4}$$

where K is the index of the node to be solved, N is the total number of neighboring elements of node K, Ki is the element index of its ith neighboring element, and F_{Ki} is one of the elemental equations of element Ki determined by the element-node numbering scheme.

To solve the 1×1 nonlinear equation (4), the Newton iteration is applied, and the increment ΔA_K can be calculated by

$$\Delta A_K = \frac{-\sum_{i=1}^{N} F_{Ki}(A_K)}{\sum_{i=1}^{N} \frac{\partial F_{Ki}(A_K)}{\partial A_K}}. \tag{5}$$

Note that the updating scheme in (5) works for all cases where the neighboring elements are all linear elements, all nonlinear elements, or a mixture of linear and nonlinear elements.

Since all the calculations are executed at the nodal level and no matrices are involved, a matched data structure is required for efficient data access. Based on the solution process presented in Fig. 4, the *structs* defined in C language are presented in Fig. 5. Each node or element is an entity with some attributes. Thus, the memory required for the NDDR scheme increases linearly with the problem size, and all computations can be completed only with two arrays of the defined structs. Note that the maximum number of neighbors was set to 8 based on the inspection that for general 2-D triangular mesh, each node has no more than 8 neighboring triangular elements. For some extreme 2-D mesh or 3-D mesh, the limit can be adjusted accordingly.

IV. Case Studies

A. Finite-Element Model of E-Core Transformer

The 2-D E-core transformer model in Fig. 6 is studied. The transformer size is 5.2 m $\times$ 3.6 m, the width of the yoke and the limb is 0.5 m, the coil size is 0.25 m $\times$ 2 m, and the coil turns are 390 for the primary side and 810 for the secondary side. The conductivity is 10^6 S/m, and the time-varying winding currents are $I_p = 5000 \sin(120\pi k \Delta t)$ A and $I_s = 2000 \sin(120\pi k \Delta t)$ A $(k = 0, 1, 2 \ldots)$. The transformer core material is Electrical Steel 35ZH135, and the nonlinear B–H curve can be found in [22]. Both the primary winding and the secondary winding are fed with sinusoidal current sources, and the produced magnetic vector potential can be calculated with the NDDR scheme. The flowchart of the NDDR scheme is presented in Fig. 7(a).

The same FE problem was solved with the commercial software package Comsol Multiphysics. The DD solver settings are presented as follows: the number of available cores for parallel processing is set to 40; the additive Schwarz scheme is applied with 40 sub-domains and the direct linear

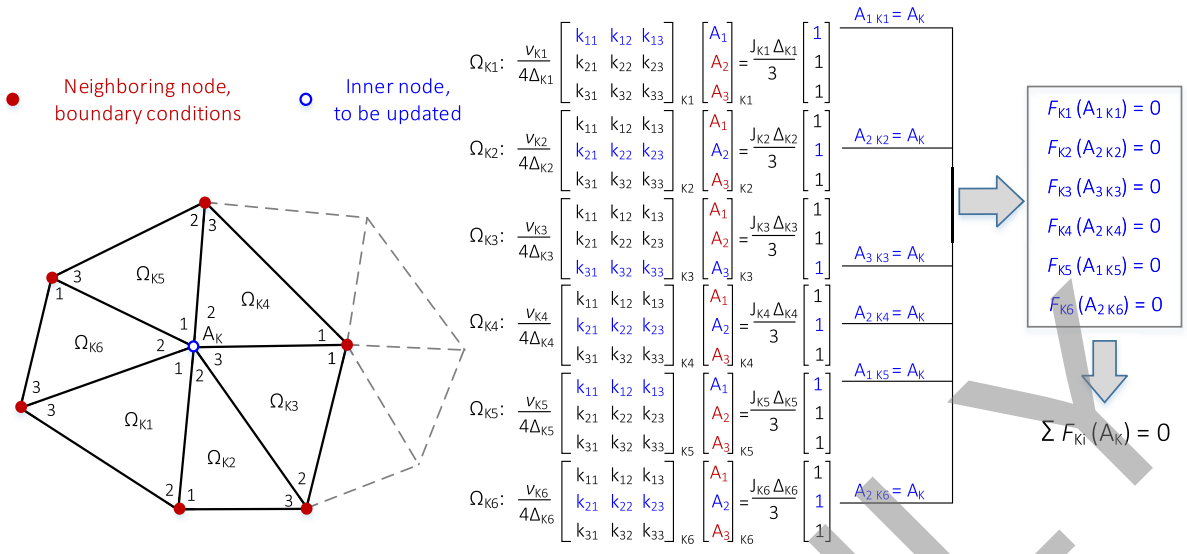

Fig. 4. Sub-domain in NDDR and its solution.

```
typedef struct
{
	int Id;
	double Cordinate_X;
	double Cordinate_Y;
	int Number_of_Neighbour_Element;
	int Neighbour_Element_Id[8];
	int Neighbour_Element_Number[8];//1,2,3
	int Node_Type;//Boundary node indicator
	double Anew;//Current iteration
	double Aold;//Previous iteration
}NDDR_FEMNode;
typedef struct
{
	int Id;
	int I,J,K;//Element-node numbering
	double Element_Area;
	int Element_Type;//Air or Ferromagnetic
	double Ve;//Reluctivity
	double Js;//Impressed current density
	double k11,k12,k13,k22,k23,k33;//Matrix
}NDDR_FEMElem;
```

Fig. 5. Data structure related to the FE nodes and elements for the NDDR scheme.

solver is used; and the nonlinear method is automatic Newton and the termination technique utilizes a tolerance factor of 10 3 or a maximum iteration number of 25. Other settings, such as damping factor and coarse preconditioning, remain the defaults. The results are regarded as reliable, and the efficiency of its optimized nonlinear DD solver is assumed to be state of the art. Thus, the results obtained from Comsol serve as the benchmark to evaluate the accuracy and efficiency of the proposed NDDR scheme.

B. Implementation, Accuracy, and Efficiency of Magnetostatic Case

To evaluate the accuracy and efficiency of a single FE computation, a magnetostatic case is studied, where the

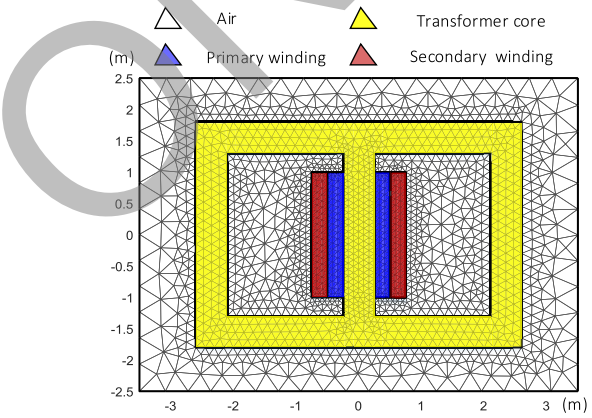

Fig. 6. FE model of an E-core transformer for the case studies.

winding currents are set to the peak values ($I_p = 5000$ A and $I_s = 2000$ A).

As an iterative relaxation scheme, the accuracy of the NDDR is determined by the convergence criteria, i.e., the relative tolerance ϵ_0 between two successive iterations. The change of the problem size is also considered. The problems are solved with both the NDDR scheme and Comsol, respectively, and Table I provides the prescribed ϵ_0, the iteration number N required, and the relative error $Error$ of the two methods for five different problem sizes.

It can be concluded from Table I that the NDDR scheme converges exactly to the same solution of Comsol if the iteration number keeps increasing. In engineering problems, since an error of less than 1% is usually acceptable, it is safe to set the relative tolerance ϵ_0 of the NDDR to 10^{-5} in all the cases in Table I.

The field distributions of the magnetic vector potential and the magnetic flux density obtained from the NDDR scheme in Case 2 with $\epsilon_0 = 10^{-5}$ are plotted in Fig. 8 with a relative error of 0.15% compared with Comsol.

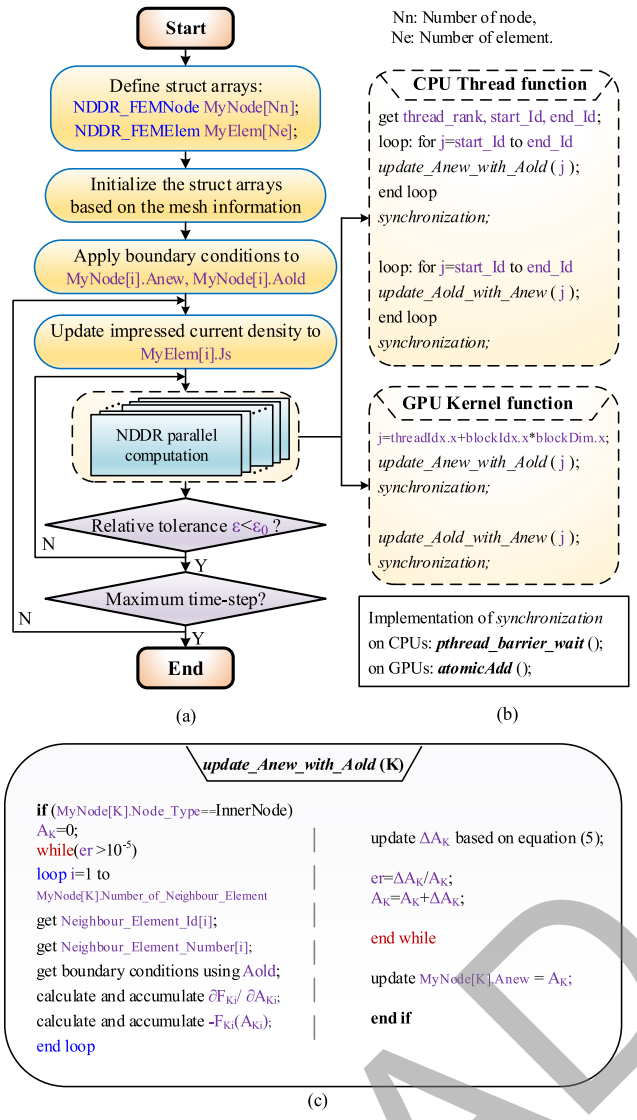

Fig. 7. Detailed implementation of the NDDR scheme on CPUs and GPUs. (a) Flowchart of NDDR. (b) Parallelization on CPUs and GPUs. (c) Sub-domain solver.

As mentioned before, the NDDR scheme is perfectly suited for massively parallel architectures, since each sub-domain can be solved independently within each iteration. The NDDR scheme is implemented on a parallel workstation with multi-core CPUs and many-core GPUs. Specifically, the workstation has dual Intel Xeon E5-2698 v4 CPUs, 20 cores each, 2.2 GHz clock frequency, and 128 GB RAM. The GPU is the NVIDIA Tesla V100-PCIE-16 GB with 5120 Cuda cores, and details can be found in [1]. Fig. 7(b) provides the parallel implementation with regard to POSIX Threads on CPUs and Kernel on GPUs, and Fig. 7(c) shows the details of the sub-domain solver. For the implementation on the 40 CPU cores, each core still needs to handle hundreds of sub-domains due to the limited number of cores, whereas, for the implementation on the GPU, each Cuda core can handle much fewer sub-domains. In fact, in Cases 1–4, each Cuda core only needs to handle one single sub-domain, since the number of nodes is less than the number of Cuda cores. The massive parallelism of the NDDR scheme also results in decent computational efficiency.

TABLE I

RELATIVE TOLERANCE ϵ_0, ITERATION NUMBER N, AND *Error* OF THE NDDR SCHEME FOR DIFFERENT PROBLEM SIZES

Cases	Number of nodes	Number of elements	Relative tolerance ε_0, iteration number N, and *Error*					
Case 1	528	1000	ε_0	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
			N	89	161	230	327	443
			Error	6.1%	0.55%	0.085%	0.011%	0.001%
Case 2	1273	2482	ε_0	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
			N	196	378	537	1104	1445
			Error	9.3%	0.82%	0.15%	0.038%	0.003%
Case 3	3303	6516	ε_0	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
			N	331	731	1456	2047	2865
			Error	20.5%	3.71%	0.45%	0.083%	0.013%
Case 4	4923	9682	ε_0	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
			N	394	1032	1663	2879	3979
			Error	33.7%	4.57%	0.78%	0.15%	0.065%
Case 5	10104	19956	ε_0	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
			N	565	1683	3067	4252	6214
			Error	51.7%	7.69%	0.85%	0.17%	0.085%

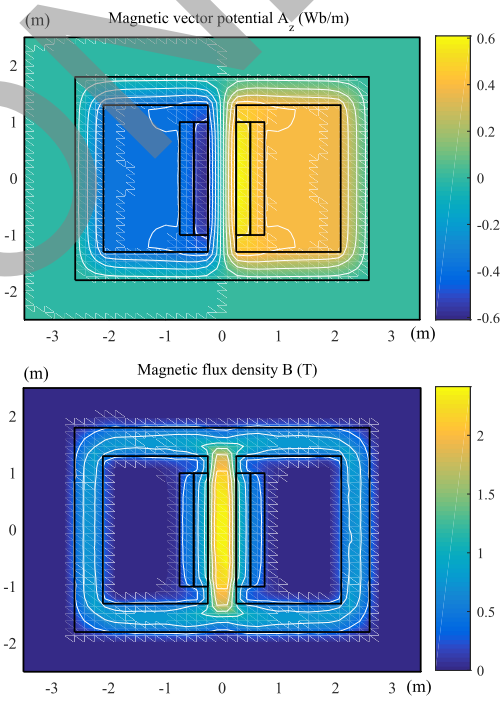

Fig. 8. Field distribution of the NDDR scheme in Case 2 with $\epsilon_0 = 10^{-5}$.

With prescribed $\epsilon_0 = 10^{-5}$, the execution time and speedups of the parallel NDDR scheme implemented on CPUs and GPU are provided in Table II. It can be inferred that the execution time of the optimized Comsol solver increases almost linearly with the problem size (node number), and it is revealed from Table I that the iteration number required for NDDR also increases linearly with the problem size (node number).

In Cases 1–4, the maximum parallelism is achieved, i.e., enough hardware resources are available so that each sub-domain is projected onto one single hardware core. Therefore, the execution time is only determined by the iteration number and thus also increases linearly with the node number. Compared with Comsol, a steady speedup of more than 30 times is achieved on the GPU implementation.

TABLE II
NDDR EXECUTION TIME AND SPEEDUP FOR CPU AND GPU PARALLELIZATION WITH $\epsilon_0 = 10^{-5}$

Cases	Comsol™ Execution Time (s)	NDDR CPU Parallelization						NDDR GPU Parallelization	
		Execution Time (s)-Thread_N					Speedup	Execution Time (s)	Speedup
		1	4	8	20	40			
Case 1	2.1	0.35	0.16	0.10	0.088	0.25	23.9	0.045	47
Case 2	3.4	1.92	0.87	0.54	0.37	0.57	9.2	0.107	32
Case 3	9.0	13.90	5.24	3.23	1.64	1.26	7.1	0.29	31
Case 4	10.3	21.61	8.89	4.88	2.22	1.67	6.2	0.32	32
Case 5	18.0	85.24	31.34	18.55	7.71	5.31	3.4	0.76	24

However, in Case 5, the speedup drops because each Cuda core has to handle two sub-domains instead of only one. Similarly, for CPU implementation, the drop of the speedup when node number increases can also be explained. With multiple GPUs in the future with more than 10 000 cores, the speedup of Case 5 with maximum parallelism can also reach more than 30 times without a doubt.

C. NDDR for Magnetodynamic Case

For magnetodynamic case, the governing equation will include the eddy current term

$$\nabla \cdot (v \nabla A) = \sigma \frac{\partial A}{\partial t} - J. \tag{6}$$

Similarly, applying the Galerkin FEM will result in the following elemental equations:

$$\frac{v^e}{4\Delta^e}\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}\begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} + \frac{\sigma^e \Delta^e}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}\begin{bmatrix} \frac{\partial A_1}{\partial t} \\ \frac{\partial A_2}{\partial t} \\ \frac{\partial A_3}{\partial t} \end{bmatrix} = \frac{J^e \Delta^e}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{7}$$

The following algebraic equations can be obtained after time discretization with the Backward Euler method:

$$\begin{aligned} &\frac{v^e}{4\Delta^e}\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}\begin{bmatrix} A_1(t+\Delta t) \\ A_2(t+\Delta t) \\ A_3(t+\Delta t) \end{bmatrix} \\ &\quad + \frac{\sigma^e \Delta^e}{12\Delta t}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}\begin{bmatrix} A_1(t+\Delta t) \\ A_2(t+\Delta t) \\ A_3(t+\Delta t) \end{bmatrix} \\ &= \frac{J^e(t+\Delta t)\Delta^e}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \frac{\sigma^e \Delta^e}{12\Delta t}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}\begin{bmatrix} A_1(t) \\ A_2(t) \\ A_3(t) \end{bmatrix}. \end{aligned} \tag{8}$$

Thus, the magnetic vector potentials at time point $t+\Delta t$ are unknowns, and the solution procedure at each time step is very similar to the magnetostatic case. The sub-domain equation (4) needs to be adjusted according to (8).

Note that in the magnetostatic case, the initial guess of the magnetic vector potential for each node is set to 0, since there is no information for reference before the problem is solved. In fact, the number of the required NDDR iterations is less if the initial guess is closer to the correct solution. This feature is very useful for the magnetodynamic case, since the final solution of each time step can serve as the initial guess of the next time step and contribute to the converging process. In the transient simulation where a small time step is applied, the field usually changes slowly, implying that the NDDR iteration number between two successive time steps can be less than the magnetostatic case presented before.

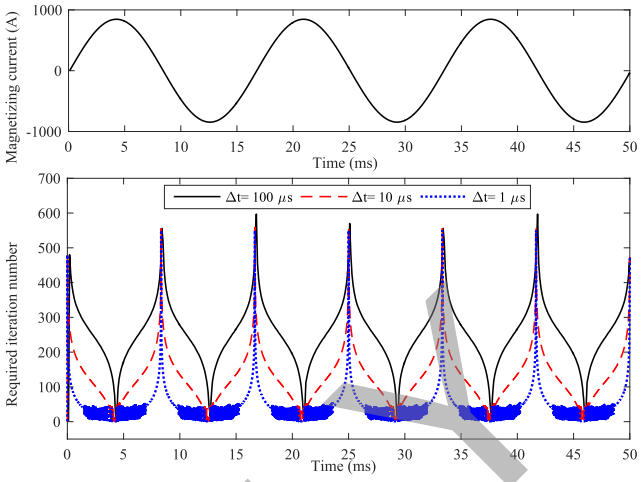

Fig. 9. Number of iterations required for different time steps to maintain a relative tolerance of 10^{-5} for the NDDR in Case 2.

For Case 2 in Table I with 1273 nodes, 2483 elements, and the prescribed relative tolerance $\epsilon_0 = 10^{-5}$, the required iteration number is 537. If the windings are fed with the time-varying sinusoidal currents provided, the initial guess of the magnetic vector potential for each time step could be set to the solution of its previous time step instead of 0. Fig. 9 shows the number of iterations required for each time step when a different Δt value is applied, which reveals that the required iteration number can be much less than 537 for a magnetodynamic problem. For example, the average number of iterations required is 254, 128, and 40 when the applied time step is 100, 10, and 1 μs, respectively. And the average execution time per time step of the Comsol time-domain solver is 2.5 s when $\Delta t = 100$ μs, 1.6 s when $\Delta t = 10$ μs, and 0.52 s when $\Delta t = 1$ μs. Thus, the average speedup of the NDDR scheme for time-domain FE computation is 53, 60, and 70 times for the time steps of 100, 10, and 1 μs, respectively.

D. Scalability and Limitation

As mentioned before, with unlimited parallel hardware resources, the execution time of the NDDR scheme will increase linearly with the problem size and gives more than 30 times speedup compared with the commercial solver, which are very attractive features. However, in practical applications, the potential of the NDDR scheme may be capped by the available parallel hardware resources. For any fixed parallel hardware resource, the execution time of the NDDR scheme will eventually increase quadratically with the number of nodes N, while the time cost of the NR and incomplete Cholesky conjugate gradient solver will roughly increase with $N^{1.5}$, and the execution time of the commercial software only provides a linear increase with N. Therefore, to gain decent speedup for an FE problem, parallel hardware resources that match the problem size would be a prerequisite for the proposed NDDR scheme.

Fortunately, the development of modern high-performance parallel computing architecture provides many possibilities for such massively parallel algorithms. In our future research, the FE problems with tens of thousands of nodes would be solved on the workstation with multiple GPUs; for larger 3-D FE problems, computer clusters would be considered, which consists of millions of computational cores.

It is also worth mentioning that due to its matrix-free property, the NDDR scheme is also promising in those FE problems with dynamic mesh generation such as rotating machines with moving parts. The changing geometries only impact the attributes of the nodes and elements involved, and all the other computations remain the same, because the proposed NDDR scheme is essentially decentralized.

In addition, the Galerkin scheme in this paper utilized node-based FEM, and thus, the sub-domain is defined by a node and its neighboring triangular elements. Also, the NDDR scheme can be potentially applied in the edge-based FEM such as 3-D problems using tangential vector FEs. In that case, the sub-domain consists of an edge element and all the tetrahedrons that share this edge.

V. Conclusion

In this paper, a novel NDDR scheme is proposed, and the massive parallelism is implemented on the prevalent parallel computing architectures. For the first time, the nonlinear FE problem in EM apparatus is solved in a decentralized manner without having to assemble a global matrix. The miniature sub-domain solver shows perfect modularity for SIMD programming with the specifically defined data structure, and the memory required increases linearly with the problem size. The accuracy and efficiency of the NDDR scheme implemented on both multi-core CPUs and many-core GPUs are discussed for different problem sizes, and comparison with the results from Comsol shows a speedup of more than 30 times while maintaining high accuracy (error less than 0.85%). In addition, for time-domain FE computation, the average speedup achieved is more than 53 times, since the solution of each time step could serve as a valuable information to contribute to the convergence of the next time step.

Future research will focus on applying the NDDR scheme in coupled electromagnetic field-transient FE computation where dynamic mesh generation is required, and extending it to a large-scale 3-D FE solution on compute clusters of multiple CPUs and GPUs.

Acknowledgment

This work was supported by the Natural Science and Engineering Research Council of Canada. The work of P. Liu was supported by the China Scholarship Council under Grant 201506120058.

References

[1] *tesla-volta-v100-datasheet-letter-fnl-web.pdf*. Accessed: Jun. 22, 2018. [Online]. Available: https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf

[2] M. M. Dehnavi, D. M. Fernández, and D. Giannacopoulos, "Enhancing the performance of conjugate gradient solvers on graphic processing units," *IEEE Trans. Magn.*, vol. 47, no. 5, pp. 1162–1165, May 2011.

[3] T. Okimura, T. Sasayama, N. Takahashi, and S. Ikuno, "Parallelization of finite element analysis of nonlinear magnetic fields using GPU," *IEEE Trans. Magn.*, vol. 49, no. 5, pp. 1557–1560, May 2013.

[4] A. F. P. de Camargos, V. C. Silva, J.-M. Guichon, and G. Munier, "Efficient parallel preconditioned conjugate gradient solver on GPU for FE modeling of electromagnetic fields in highly dissipative media," *IEEE Trans. Magn.*, vol. 50, no. 2, pp. 569–572, Feb. 2014.

[5] Q. Dinh and Y. Marechal, "Toward real-time finite-element simulation on GPU," *IEEE Trans. Magn.*, vol. 52, no. 3, Mar. 2016, Art. no. 7207304.

[6] W.-J. Wang *et al.*, "Massively parallel simulation of large-scale electromagnetic problems using one high-performance computing scheme and domain decomposition method," *IEEE Trans. Electromagn. Compat.*, vol. 59, no. 5, pp. 1523–1531, Oct. 2017.

[7] Y. Takahashi, K. Fujiwara, T. Iwashita, and H. Nakashima, "Parallel finite-element analysis of rotating machines based on domain decomposition considering nonconforming mesh connection," *IEEE Trans. Magn.*, vol. 52, no. 3, pp. 1–4, Mar. 2016.

[8] G. F. Carey, E. Barragy, R. McLay, and M. Sharma, "Element-by-element vector and parallel computations," *Commun. Appl. Numer. Methods*, vol. 4, no. 3, pp. 299–307, 1988.

[9] I. Kiss, S. Gyimothy, Z. Badics, and J. Pavo, "Parallel realization of the element-by-element FEM technique by CUDA," *IEEE Trans. Magn.*, vol. 48, no. 2, pp. 507–510, Feb. 2012.

[10] J. P. A. Bastos and N. Sadowski, "A new method to solve 3-D magnetodynamic problems without assembling an $Ax = b$ System," *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 3365–3368, Aug. 2010.

[11] D. M. Fernández, M. M. Dehnavi, W. J. Gross, and D. Giannacopoulos, "Alternate parallel processing approach for FEM," *IEEE Trans. Magn.*, vol. 48, no. 2, pp. 399–402, Feb. 2012.

[12] P. Liu and V. Dinavahi, "Finite-difference relaxation for parallel computation of ionized field of HVDC lines," *IEEE Trans. Power Del.*, vol. 33, no. 1, pp. 119–129, Feb. 2018.

[13] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 7, pp. 1255–1266, Jul. 2012.

[14] Z. Zhou and V. Dinavahi, "Parallel massive-thread electromagnetic transient simulation on GPU," *IEEE Trans. Power Del.*, vol. 29, no. 3, pp. 1045–1053, Jun. 2014.

[15] H. Karimipour and V. Dinavahi, "Extended Kalman filter-based parallel dynamic state estimation," *IEEE Trans. Smart Grid*, vol. 6, no. 3, pp. 1539–1549, May 2015.

[16] Z. Zhou and V. Dinavahi, "Fine-grained network decomposition for massively parallel electromagnetic transient simulation of large power systems," *IEEE Power Energy Technol. Syst. J.*, vol. 4, no. 3, pp. 51–64, Sep. 2017.

[17] S. Huang and V. Dinavahi, "Fast batched solution for real-time optimal power flow with penetration of renewable energy," *IEEE Access*, vol. 6, pp. 13898–13910, 2018.

[18] P. Liu and V. Dinavahi, "Real-time finite-element simulation of electromagnetic transients of transformer on FPGA," *IEEE Trans. Power Del.*, vol. 33, no. 4, pp. 1991–2001, Aug. 2018.

[19] O. Zienkiewicz, R. Taylor, and J. Z. Zhu, *The Finite Element Method*, 6th ed. London, U.K.: Elsevier, 2005.

[20] S. Salon, *Finite Element Analysis of Electrical Machines*, vol. 101. Boston, MA, USA: Kluwer, 1995.

[21] M. Dryja and O. B. Widlund, "Towards a unified theory of domain decomposition algorithms for elliptic problems," Division Comput. Sci., Courant Inst. Math. Sci., New York Univ., New York, NY, USA, 1989.

[22] *Free BH Curves - Magweb*. Accessed: Jun. 22, 2018. [Online]. Available: http://magweb.us/free-bh-curves

Peng Liu (S'15) was born in Xuchang, Henan, China, in 1992. He received the B.Sc. and M.Eng. degrees in electrical engineering from the Harbin Institute of Technology, Harbin, China, in 2013 and 2015, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada.

His current research interests include computational electromagnetics, real-time simulation, and parallel and distributed processing.

Venkata Dinavahi (S'94–M'00–SM'08) received the Ph.D. degree from the University of Toronto, Toronto, ON, Canada, in 2000.

He is currently a Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His current research interests include real-time simulation of power systems, large-scale system simulation, and parallel and distributed computing.