International Journal of Innovative
Computing, Information and Control
Volume 9, Number 6, June 2013

ICIC International ©2013 ISSN 1349-4198
pp. 2303–2326

STAGEWISE OPTIMIZATION OF DISTRIBUTED CLUSTERED FINITE DIFFERENCE TIME DOMAIN (FDTD) USING GENETIC ALGORITHM

Nordin Zakaria, Anindya Jyoti Pal and Syed Nasir Mehmood Shah

High Performance Computing Centre
Universiti Teknologi Petronas
Bandar Seri Iskandar 31750, Tronoh, Perak, Malaysia
{ nordinzakaria; anindyajp; nasirsyed.utp }@gmail.com

Received March 2012; revised July 2012

Abstract. *In this paper, we explore the use of hybrid genetic algorithm for optimized clustering and distribution of Finite Difference Time Domain (FDTD) computation over a large number of desktop computers and servers. Given a large number of computers, we first attempt to compute an optimal set of clusters. The clustering takes into consideration similarity of machine capability plus the interconnection speed. It considers as well the predicted availability pattern for each computer. Then, for each cluster, we optimize the distribution of FDTD workload over its computers. Hence, the overall optimization procedure optimizes clustered distributed FDTD. We show in this paper how pure as well as hybrid genetic algorithm can effectively be used to perform the optimization.*
Keywords: FDTD, Optimized, Cluster, Distribution, Hybrid, Genetic algorithm

1. **Introduction.** The Finite Difference Time Domain (FDTD) method, introduced in [42], is a widely used numerical method for approximating the solution of coupled partial differential equations, in particular, that which describes the propagation of electromagnetic (EM) waves. The method performs forward modeling, i.e., given the geometric and electrical properties of a particular structure, the electric field generated within a region over a time duration is computed. Forward modeling is at the core of EM inverse modeling – reconstruction of the parameters of the structures that are responsible for measured EM properties. The typical inverse modeling process can be described conceptually as a generate-and-test optimization procedure, and in certain domain, for example, geophysical exploration [1,37], requires large number of runs and hence very large computational resources.

Driven by an EM inverse modeling problem in the context of hydrocarbon exploration, we consider in this paper the problem of running as many FDTDs as possible, within a campus grid. A campus grid is a large heterogeneous collection of computing resources distributed throughout a university campus, interconnected at a physical level through the local area network and at the operating system level through a middleware (such as MPI). In the context of this paper, the resources refer primarily to the thousand of desktop computers and workstations available in the university computer labs. While FDTD has typically been run on large, shared-memory machines or on tightly interconnected clusters, we seek to reduce the workload on such available machines in our campus and to explore instead the possibility of fully utilizing the collective power of the desktops and workstations. Despite the huge latent computing power, it makes no sense though to run a single FDTD run over all of the available resources. The inter-processor communication required in distributed FDTD would result in poor scalability. Clusters of computers

need to be formed instead, each one of which is to focus on just a single FDTD run at a time. The formation of the clusters must take into consideration machines characteristics and availability, that is the opportunity for high performance. Hence, the clustering is opportunistic in nature. Once the clusters have been formed, the FDTD workload must be distributed among the computers within each, again taking into consideration heterogeneous machine characteristics and availability.

The overall problem can be described formally as follows: We assume an FDTD run to comprise of a fixed number of temporal iterations, N, over a fixed workload $v = w \times h \times d$. Given a set M of n computers, $M = \{m_0, m_1, \ldots, m_{n-1}\}$, in order to perform the maximal number of FDTD runs, we need to form a partitioning, $P = \{p_0, p_1, \ldots, p_{q-1}\}$ of M, where each partition p_i denotes a computer cluster. The partitioning should be such that the average number, F_{avg} of FDTD runs completed by the clusters within a given time period T is maximal.

For the problem posed in this manner, it is very hard to find optimal result in one go, as it is essentially a two-level problem. At one level, we need to optimize the clustering, while at another level we need to optimize the FDTD parallelism within each cluster. In other words, the fitness of a solution at the higher level can only be determined by searching for the fittest solution at the lower level. While a population-based optimization procedure such as genetic algorithm can, in principle, solve the problem, the computational requirement is practically forbidding.

To make the problem more tractable, we split the optimization into two stages. In the first stage, the clustering stage, we compute clustering solution such that the performance of each cluster is likely to be maximal. Then, in the second stage, the workload distribution stage, the FDTD workload distribution within each cluster is optimized so as to attain that maximal performance.

The overall solution performance then depends on two factors – the optimality of the cluster, and the optimality of the FDTD workload distribution within each cluster. The primary challenge in addressing this problem is the need, within the setting of a campus grid, to deal with the heterogeneous machine and interconnection availability factor. Further, each machine has its own expected availability pattern. While there has been a steady body of work dealing with parallel FDTD [11,26] or similar computation, none to our knowledge dealt with a similar context. In fact, to our knowledge, no previous work considers workload balancing taking into consideration the availability factor.

Due to the nature of the problem, we have chosen to derive a solution based on genetic algorithm (GA) [10]. The advantage in using GA is that, it makes no assumption about the mathematical tractability of the problem. Starting with a large set or population of random solutions, the algorithm applies principles from natural evolution to drive the population towards optimal solutions. The key to GA's strength is in fact in its population-based nature, allowing it to attack and probe the search space from a wide range of vintage points. The main setback to GA is however in its computational demand, prohibiting its use for real-time purpose. In our case, however, this is not an issue (so long as the computational time is reasonable), since our objective is to derive a static resource allocation plan for clustered distributed FDTD executions.

In the general context of grid computation, GA has been widely used for generic form of grid resource allocation and scheduling problem. In [32] for example, the use of GA for resource brokering in grid environment is discussed. In [39], the authors addressed the optimization of energy efficiency, makespan and user perceived Quality of Service (QoS) in grid scheduling. In [40], a comprehensive set of experiments to investigate the optimal form of GA for grid job scheduling was presented. Multi-stages optimization has been deployed elsewhere in the literature in very different context. For example, T. Tometzki

and S. Engell [33] investigated a two-stage stochastic mixed-integer programming problem where an evolutionary algorithm handles the first-stage decisions and mathematical programming the second. In our context, however, we use GA for both stages of our solution.

Our main contribution in this work is hence the formulation of a GA-based approach for optimized clustered distributed FDTD. More specifically, contributions are as follows:

- formulation of a GA based solution to the problem of forming optimal computer clustering, taking into consideration machine charactheristics, interconnections and availabilities,
- formulation of a GA based solution to the problem of distributing FDTD workload within a cluster, again taking into consideration machine charactheristics, interconnection and availability.

We organize the rest of the paper as follows: We provide an overview of FDTD in Section 2. In Section 3, we describe our genetic clustering method. In Section 4, we present our FDTD workload balancing solution. In Section 5, we present our experimental results, and finally in Section 6, we conclude the paper.

2. **FDTD Review.** While a wide range of FDTD variants has been described in the literature, in this section, as we seek only to impart a sense of the algorithmic structure of FDTD, we focus on the simplest form, one that simulates the propagation of electromagnetic wave in a vacuum. The Maxwell's equations that describe this propagation are as shown in Equations (1) and (2).

$$\nabla \times \vec{H} = \varepsilon_0 \frac{\partial \vec{E}}{\partial t} \tag{1}$$

$$\nabla \times \vec{E} = -\mu_0 \frac{\partial \vec{H}}{\partial t} \tag{2}$$

The equations above describe the evolution of the electric, $\vec{E}$, and the magnetic component, $\vec{H}$, of an electromagnetic wave over a spatial region over time, t. In the equations, the ε_0 and μ_0 denote respectively the electrical permittivity and magnetivity of the vacuum.

Equations (1) and (2) can be discretized (see for example [8]) in both temporal and spatial domain. Three discrete update equations can be derived from Equation (1), each corresponding to a single component of $\vec{E}$. Similarly, three discrete update equations can be derived from Equation (2) for the computation of the $\vec{H}$ field. The resulting update equation for E_x, as an example, is as shown in Equation (3):

$$E_x^{t+1}(i,j,k) = E_x^t(i,j,k) + \frac{\Delta t}{\varepsilon \Delta s} curl_h \tag{3}$$

where

$$\begin{aligned} curl_h &= (H_z^{t+1/2}(i,j+1/2,k) - H_z^{t+1/2}(i,j-1/2,k) - H_y^{t+1/2}(i,j,k+1/2) \\ &\quad + H_y^{t+1/2}(i,j,k-1/2)) \end{aligned}$$

Equation (3) performs update for the E_x value at position i, j, k. The new E_x value, on the left side of the equation will be at time step $t+1$, derived from an expression involving its value at time step t. The spacing between consecutive time step is given by Δt. The calculation of $curl_h$ in the equation involves a stencil block from the dual field, that is the $\vec{H}$ field. It is important to note that the discretized $\vec{E}$ and $\vec{H}$ values are staggered spatially and temporally. In other words, $\vec{E}$ points are not spatially collocated with the $\vec{H}$ points, and temporally, both fields are updated in an alternate manner.

A further point to note is that the FDTD can be practically applied only within a finite region or spatial volume. An absorbing boundary condition (ABC) must be in place to prevent spurious EM reflection from the boundary of the finite region. A number of ABCs is commonly in use, the most popular of which being the perfectly matched layer proposed by Berenger [3] and its convolutional variant by Roden and Gedney [28]. In this work, we assume the ABC cost to be negligible, due to the relatively thin layers required for its implementation.

In pseudocode, the FDTD updates to be performed at each time step is depicted below:

$for\ i, j, k = 0 \rightarrow IE, JE, KE\ do$

$ex[i, j, k] := ex[i, j, k] + stencil(hy, hz)$

$ey[i, j, k] := ey[i, j, k] + stencil(hx, hz)$

$ez[i, j, k] := ez[i, j, k] + stencil(hx, hy)$

$end\ for$
$send\_blockboundary(e)$
$accept\_blockboundary(e)$

$for\ i, j, k = 0 \rightarrow IE, JE, KE\ do$

$hx[i, j, k] := hx[i, j, k] + stencil(ey, ez)$

$hy[i, j, k] := hy[i, j, k] + stencil(ex, ez)$

$hz[i, j, k] := hz[i, j, k] + stencil(ex, ey)$

$end\ for$
$send\_blockboundary(h)$
$accept\_blockboundary(h)$

The pseudocode above deals may process either the whole FDTD volume or a single FDTD block. In parallel or distributed FDTD, the entire volume is partitioned into blocks, each one of which is to be executed on a different machine using the same FDTD update operations. While the FDTD is easily parallelizable, due to the stencil computation, FDTD update at the boundary layer of a subblock will involve communication with adjacent blocks. The *send_blockboundary* and *accept_blockboundary* in the above pseudocode performs this communication function. In the case whereby the boundary face is an external face, i.e., it does not face another block, the functions will then perform the transfer of data to and from the ABC layers instead.

3. **Stage 1: Resource Clustering.**

We consider in this section the resource clustering stage. As far as we can tell, there has been very few work in the literature that deals with resource clustering. The closest is that in [2]. As in our work, in [2], the authors attempted to create clusters where each cluster focussed on a single task. Their optimization effort focussed on minimizing the intra-cluster communications. Posing the problem as a bin-covering problem, a distributed approximation solution was proposed, where each node autonomously made a decision as to whether or not to join a particular cluster. The solution was hence distributed in nature. In our context, however, we need only a centrally computed solution. Furthermore, we consider more factors – resource availability and processing rate factors, not taken into consideration in [2].

TABLE 1. Availability profile (1 for true, 0 for false)

machines	profile
m_0	$1,0,1,1,0,0,1,1$
m_1	$0,0,0,0,1,1,1,1$
m_2	$0,0,0,0,1,0,1,1$
m_3	$1,1,1,1,0,0,0,1$

3.1. **Resource features.** Given a heterogeneous set of computers, we seek to partition the set into clusters. We assume that for each computer, m_i, we know the following characteristics:

1. The FDTD processing rate, ν_i, i.e., is the number of FDTD cells that can be processed by machine m_i per second. A simple FDTD benchmarking program can be used to obtain this value, and its value is likely to be proportional to the allocated CPU rate, i.e., the portion of CPU dedicated to grid tasks. In our case, users and system administrators will typically set the fraction of CPU rate to be dedicated to grid task, considering factors such as electrical usage and air-conditioning level.
2. The FDTD data transfer rate (i.e., the number of FDTD cells transferable per second), c_{ij}, to another computer m_j. This value can also be obtained using simple benchmarking.
3. Allocated memory space, $|m_i|$, that is the amount of memory dedicated for the distributed FDTD tasks. This value may be pre-set by the users or the administrators.
4. Predicted computer availability, A_i, over a period of time.

The availability for a computer p_i over a period of time of duration T is specified in the form of an *availability profile*, $A_i = a_0, a_1, \ldots, a_{T-1}$, where a_t is the availability (*true* or *false*) of the machine at time t. To predict resource availability, historical availability patterns for the computers available within the grid needs to be collected, and a prediction technique such as those published in [9,15,17], can be used to predict future trends. While prediction is probabilistic and uncertain in nature, within a campus context, such extrapolated data is usually reliably accurate. The reason behind this is due to the usually repeating computer usage patterns induced by university timetables at both undergraduate and postgraduate levels.

To understand the need to consider availability vector when performing clustering, consider the following: If a machine m_i within a cluster has to wait for another machine m_j, m_i would be losing time. The longer or the more frequent the wait, the less productive is a cluster. In the worst possible case, suppose machines m_i and m_j are running interdependent processes and their availability alternates indefinitely – that is when m_i is available, m_j becomes unavailable, and vice versa. In such a situation, the simulation would simply stall. Hence, in a good solution, the availability of the machines within the same cluster is likely to be similar. As an example, suppose we have the availability profile of four machines as shown in Table 1. The profile in the table shows the availability of each machine over an eight 15-minutes period. A reasonable clustering based on availability (alone) would be as follows: $\{m_0, m_3\}$, $\{m_1, m_2\}$.

3.2. **Clustering optimization.** The clustering strategy should be clear by now: we seek to group machines with similar capability and availabilty together. Within each cluster, the computer nodes should be

- as similar as possible in terms of CPU rate and availability vector
- with a maximal average internode data transfer speed

Furthermore, the following constraint needs to be satisfied: the collective memory of the computers within a cluster must be sufficient enough to hold an FDTD workload.

As for the number, K, of clusters to be formed, we adopt a simple strategy, based on the number of FDTD loads, each of size that can fit into the collective memory of the given computers. The number of clusters, K, is computed from the collective memory, $|M|$, and the FDTD load size, v, as follows:

$$K = \frac{|M|}{v} \tag{4}$$

Of course, if each computer can hold the entire FDTD load by itself, we may well have clusters of size 1. In our case, this does not matter, as the total number of computers at our disposal is fixed at M, and if we ignore internode communication, it does not matter if we use m clusters each with n computers, or n clusters each with m computers, where $M = n \times m$. Factoring in the communication cost however, we prefer a cluster size which is tight-fitting, as implied by Equation (4).

Given a set of M computers, the K-clustering problem can be posed as a search for a partitioning P that maximizes:

$$\frac{1}{|P|} \sum_{k=0}^{|P|} \left(\sum_{i=0}^{|p_k|} \sum_{j=0, j \neq i}^{|p_k|} C_{ij} \right) \tag{5}$$

where

$$C_{ij} = \frac{(1 - A_{ij}) + (1 - m_{ij})}{c_{ij}} \tag{6}$$

$$A_{ij} = \frac{(A_i - A_j)}{|A|} \tag{7}$$

and

$$m_{ij} = \frac{|m_i| - |m_j|}{m_i} \tag{8}$$

The A_{ij} term in Equation (7) is the Hamming distance between the availability vector for machine p_i and that for p_j. Hence, A_{ij} is larger for closer availability vectors. In a similar way, the m_{ij} term in Equation (8) has a value that increases with a decrease in the difference between $|m_i|$ and $|m_j|$. The denominator term, c_{ij}, in Equation (6) denotes the time it takes to transfer an amount of load from machine m_i to m_j, and acts as the scaling term. Hence, the less the time it takes for the data transfer, the better the overall match between the two machines.

We consider the following two ways to optimize the expression in Equation (5):

- performing greedy clustering,
- using genetic algorithm.

Greedy clustering works as follows:

repeat
randomly, select a 'free' (yet to be clustered) machine, m_i
randomly, rank all other free machines using Equation (6)
form a cluster using m_i and its best $M/K - 1$ peers.
until K clusters are formed

This greedy approach is intuitive and is commonly used in forming computer clusters. The problem with the method, as with many other greedy algorithms [6] in Computer Science, is that the solution obtained is unlikely to be optimal for complex dataset. Nevertheless, the method is fast, and can be a good heuristics for a quick guess.

Next, the GA for this approach assumes a standard form, as follows:

begin
INITIALIZE population with random candidate solutions;
EVALUATE each candidate;
repeat
SELECT parents;
RECOMBINE pairs of parents;
MUTATE the resulting children;
EVALUATE children;
SELECT individuals for the next generation
until no_of_iteration
end

For the GA implementation, we deploy chromosomes of length M, where each gene denotes a cluster index. As an example, the sequence $\langle 1, 1, 2, 2, 2\rangle$, a clustering chromosome for a 5-machines set, denotes that the first 2 machines are to be in cluster 1, while the last 3 machines to be in cluster 2. A similar approach was reported in [12,13,22,24]. Given the linear chromosome structure, uniform crossover, single-point crossover and uniform flip mutation suffice for our purpose. An initial population of random possible chromosomes are first generated. We then evolve it for a number of generations, at each generation, using tournament scheme to select individuals to be involved in producing the next generations.

Note that due to the nature of the problem, GA approaches that evolves center of clusters, as in [18,21,23,35], are not likely to be applicable. While consuming less memory per chromosome, given the multidimensional nature of the problem, such approach cannot be extended in a straightforward manner to address the problem as expressed in Equation (5).

We now consider the performance of greedy algorithm relative to the initial GA population. This is as plotted in Figure 1. For the result in Figure 1, the number of computers involved in the clustering is 100. The maximum FDTD processing speed, the maximum allocated memory space and the maximum interconnection speed is assumed to vary with normal distribution between the minimum and the maximum values shown in Table 2. Availability patterns of 100 time duration, each of them with length of 10 minutes, are randomly generated. The number of GA generations is experimentally fixed at 500, and

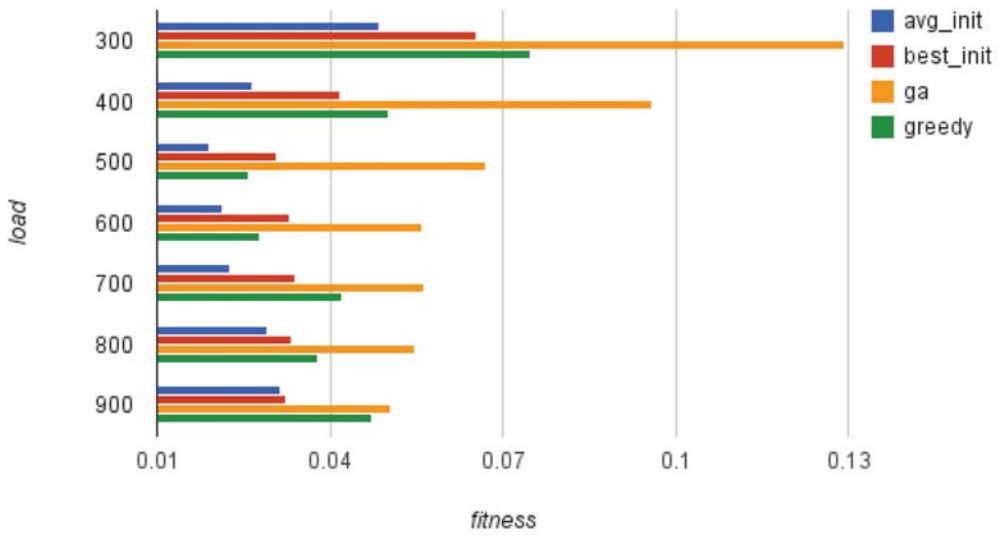

FIGURE 1. Clustering performance

TABLE 2. Min-max setting for result in Figure 1

size/rate	cells or cells/sec
max memory	26301781
max cpu rate	23068672
max network rate	54613
min memory	10520712
min cpu rate	9227468
min network rate	21845

the population size at 500 throughout the evolution. The mutation rate is fixed at 0.01, and the crossover rate at 0.5. We further note that the performance result in this section is more exploratory in intent, rather than rigorous. More rigorous results are presented in Section 5.

As shown in Figure 1, as expected, GA largely outperforms the greedy approach. However, note that the greedy approach obtains result at least close to the best fitness (*best_init*) in the GA initial population. In fact, it obtains a fitness value consistently better than the average fitness (*avg_init*) in the GA initial population.

What this result suggests is that the greedy deterministic approach may be useful in helping GA to converge better. We hence consider the combination of GA with the greedy approach as follows:

Run greedy clustering as pre-optimization stage,
Use its result to seed the GA population after its initialization as follows:
  Assume a seeding probability α
    For each individual in the population
      With a probability α, mark the individual
        If (individual is marked)
          Compute a seed by greedy clustering with a randomly shuffled machine list
            If (the seed is fitter than the selected individual)
              Replace that individual by that seed

One would suspect that seeding would improve the average fitness value in the GA initial population. We examine this, considering the effect of seeding in the case whereby the input load is $500 \times 500 \times 500$. As shown in Figure 2, seeding does appear to improve the average fitness of the initial GA population. In fact, the average fitness increases with the seeding probability. This result is to be expected as we replace an individual within the population with a seed only if the individual is less fit than the seed.

We proceed on to investigate how seeding affects the final fitness value. Plotting the variation in final fitness improvement with various seeding rates, we obtain the graph in Figure 3. One can deduce from the graph that the seeding rate should optimally be of low value, within the range 0.1 to 0.25.

Hence, seeding does seem to be a promising approach to improve the clustering performance. In fact, the benefit of seeding has been reported in a number of works. In [27] for example, apply case-based reasoning to create individuals forming half of the initial population. In [16], seeding is applied to speed up GA-based solution for the rectilinear steiner problem, and in [25], seeded population is considered in the context of a bioinformatic problem. However, there is a certain subtlety in applying seeding in our specific context, and this we explore more rigorously in Section 5.

4. **Stage 2: FDTD Workload Distribution.** In a homogeneous environment, an FDTD workload balancing algorithm can be fairly simple, i.e., simply divide the load

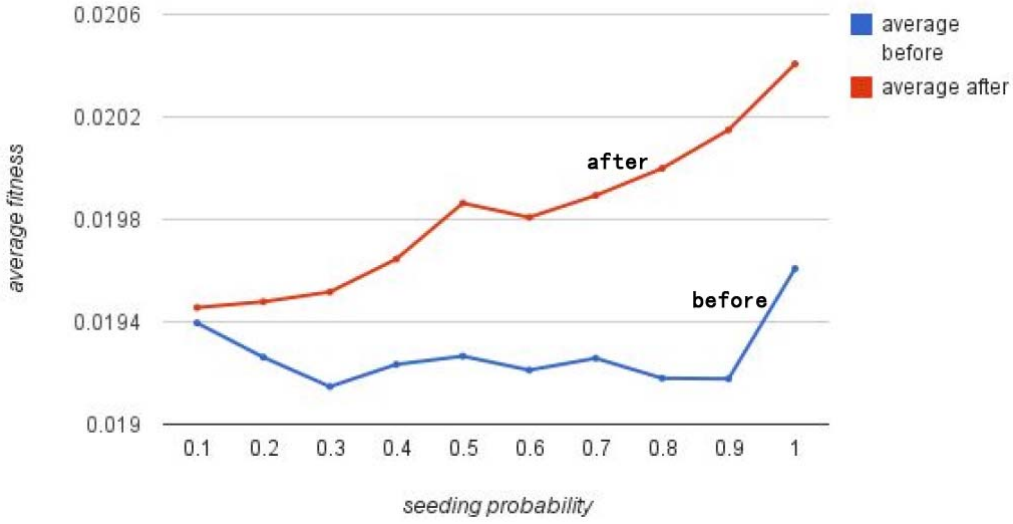

FIGURE 2. Typical effect of seeding on average fitness

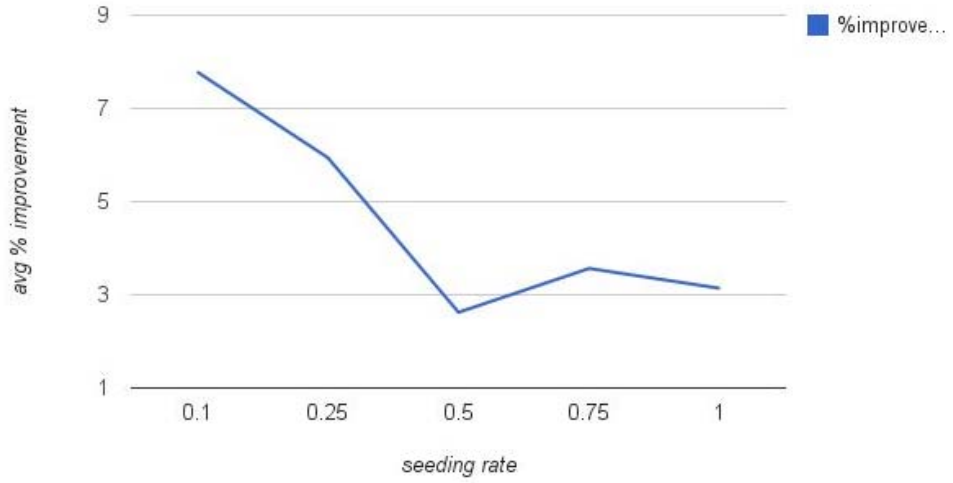

FIGURE 3. Typical effect of seeding on final GA fitness

equally among the available resources. However, in our case the target environment however is a heterogenous, geographically-dispersed, non-dedicated (or opportunistic) grid environment. In such an environment, the technical challenge is due to the following factors:

- Different resources within the grid have different memory and processing capacity. Furthermore, the network bandwidths due to traffics and configurations are different too.
- Each resource is non-dedicated.

The work in [30], is the most recent, that addressed the problem of optimizing the distribution of FDTD workload balancing in a heterogeneous grid. They assumed an environment where the performance of the machines remain constant throughout the execution. They posed the problem of distributing FDTD as that of find the partitioning

Ω that minimizes the following:

$$\left\{ \max_i \left\{ \frac{1}{\nu_i} |\Omega_i| + \sum_{j \neq i} \frac{1}{c_{ij}} |\partial\Omega_i \cap \partial\Omega_j| + \gamma_i |\partial\Omega_i \cap \partial\Omega| \right\} \right\} \tag{9}$$

where $0 \leq i < |P|$, Ω_i is the i-th partition or block, $|\Omega_i|$ the size of the block, $\partial\Omega_i \cap \partial\Omega_j$ is the intersection between 2 blocks, and $\partial\Omega_i \cap \partial\Omega$ the intersection between block i and the computational boundary. γ_i is a constant that relate the size of the ABC boundaries to the cost of execution, and ν_i, as earlier in the paper, the number of FDTD cells that can be processed by machine m_i.

As said in [30], optimization of FDTD distribution based on Equation (9) is non-trivial as the geometry and position of the partitions need to be known first before the overlap between neighboring partitions can be computed. In [30], authors performed the optimization in a two-stage manner. They first determined the partition sizes by solving a simplified version of Equation (9) using a nonlinear constrained optimization method. They then used a partitioning algorithm that recursively subdivide the FDTD domain in such a way that the sizes of the resulting partitions are as close as possible to the optimal sizes computed in the first step.

The following assumptions in [30] render it unsuitable for our purpose:

- In the first stage of the approach in [30], it was assumed that the communication speed from a particular resource to any other resource is the same, that is $\beta_{ij} = \beta_i$. This is certainly untrue for our target grid environment.
- The memory constraint of each of the participating resource was not considered. Even if Equation (9) can be optimized while respecting memory constraints, in the second stage, adjustment of the volume block size to accomodate 3D partitioning tends to result in violation of the constraints.
- A dedicated set of machines is assumed, so the availability is not a concern.

4.1. **GA design for FDTD workload distribution.** Due to the limitations in [30], we consider using a GA-based approach. The basic idea in our approach is to search for an optimal way to subdivide the FDTD load recursively along one of the major axis, until the number of blocks is equal to the number of machines in the cluster. The optimality of a subdivision is a function of the expected number of FDTD iterations that can be executed by the cluster, and this is predicted using a full-scale simulation.

Assuming that the set of neighboring blocks, or simply the neighboring set S_i, for each block Ω has been computed, the pseudocode for the simulation is as shown below:

```
for i = 0 → |Ω| do
  if |Ω_i| < |m_i| then
      return 0
  end if
  status_i ← READY
  procTime_i ← processingTime(m_i, |Ω|)
  numiteration_i ← 0
  commTime_i ← 0
  for k = 0 → |S_i| do
      m_j ← S_ik
      commTime_i ← commTime_i + c_ij
  end for
end for
```

```
for T = 0 → |A| do
  for t = 0 → lengthofperiod do
    for i = 0 → |Ω| do
      if m_i not available then
        continue
      end if
      if status_i is PROCESSING then
        procTime_i ← procTime_i − 1
          if procTime_i is 0
            status_i ← COMMUNICATING
          end if
      end if
      if status_i is COMMUNICATING
        commTime_i ← commTime_i − 1
          if commTime_i is 0
            status_i ← PROCESSING
            numiter ← numiter + 1
          end if
        end if
    end for
  end for
end for
```

4.1.1. *Chromosome design.* We design our chromosome as shown in Figure 4. As can be seen in the figure, the chromosome comprises of a linear sequence of pairs of numbers. Each pair forms a code. A chromosome is hence a linear sequence of codes. The first number in a code is an integer, which indicates the axis to be used for subdivision, while the second, a real number, indicates the position, normalized from 0 to 1, along that axis. Applying the instructions in a chromosome, we obtain a list of FDTD blocks. Hence, to generate m blocks, $m-1$ instruction steps or genes are needed.

The blocks formed by a chromosome is mapped index-wise to the corresponding machines in the machine list. Hence, block ψ_i is mapped to machine m_i. A mismatch may occur, that is, ψ_i, may be mapped to m_i with insufficient memory. To resolve the mismatch, we search for a machine m_j such that $|\psi_i| \leq |m_j|$ and $|\psi_j| \leq |m_i|$. If the search is successful, we perform a swap such that ψ_i is mapped to m_j and ψ_j to m_i. Otherwise, we assume that the mismatch is irrepairable, and a penalty is imposed onto the chromosome. Note that we could have applied other forms of correction, for example, by spreading

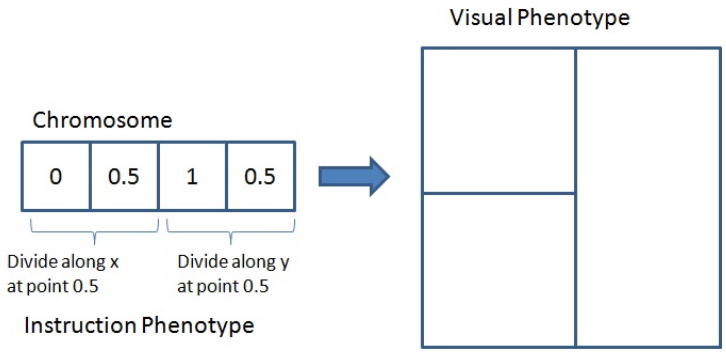

FIGURE 4. Chromosome for load balancing

out the excess load, $||\psi_i| - |m_i||$ over the remaining machines that still have free memory space. However, this will result in multiple chromosomes mapping to the same phenotype, and subsequently a loss of phenotypic diversity in the population.

4.1.2. *Genotype-to-phenotype mapping subtlety to be considered in mutation.* At the essence, given a chromosome as discussed in the previous section, mutation simply involves random perturbation of the gene values. A code within the chromosome comprises of an axis and a split point value. An axis is perturbed by randomly changing its value to any of the other 2 possible values. A split point mutation involves the addition of a small real value to the current split value.

There is a certain subtlety however in mapping the a mutated split point value to the corresponding phenotype. Along the chromosomes, going from the left to the right, a small fixed change in the split point value will be mapped to increasingly large change in the phenotype. To understand this, recall that the chromosome is a list of instruction codes, interpretated from left to right. Each code results in the split of the input volume. As we go to the right, we are splitting smaller and smaller volume subblock. The smaller a volume subblock, the more significant a change in the split value will be. For example, a fixed change of 0.1 means a change of 1 unit along the chosen split axis if the volume subblock being considered has a dimension of 10 along that axis, but a change of 100 units if the volume dimension is instead 1000 along that axis. Therefore, when mutating the split point value, the size of the volume subblock along the chosen axis must be considered, and the perturbation size appropriately scaled.

Hence, the form of the mutation as used in this work is one with non-uniform probability rate within the same chromosome which is specific to the problem of volume partitioning. In fact, there has been many problem-specific operators used in many work on real-world GA application problems. In the work of [34] for example, authors proposed a problem-specific crossover operator, named "conflict-free partition crossover" in formulating a GA-based solution to the compiler register allocation problem. In [4], authors reviewed knowledge-based operators, especially crossover operators, required in solving permutation-based problems. The general emphasis in problem-specific operator design in the literature is on the crossover design. In our context here, however, the mutation is more of concern.

The performance impact of the proposed scaled mutation rate scheme, as proposed in this section, is investigated in Section 5. Statistical analysis of the results, as will be elaborated upon in the section, indicates a significant improvement due to the scheme.

4.1.3. *GA application strategy.* There are two primary concerns in applying GA to the workload distribution problem:

1. Firstly, the concern is with the feasibility of the solutions evolved. While penalty functions may be applied to encourage the evolution of solutions that respect the memory constraints of machines within the input cluster, a sizable fraction of the overall population will likely be infeasible, as shown in Figure 5, and have to be penalized. Ideally, the optimal solution is derived based on a population of legal individuals.
2. The full-scale simulation to be deployed is computationally taxing, and makes practical only smaller GA population sizes.

4.1.3.1. Concern 1: feasibility of solutions. The first concern is addressed by first evolving the GA population with a view of producing legal chromosomes. In this first stage, only the penalty function is applied. A load-resource memory mismatch results in deduction

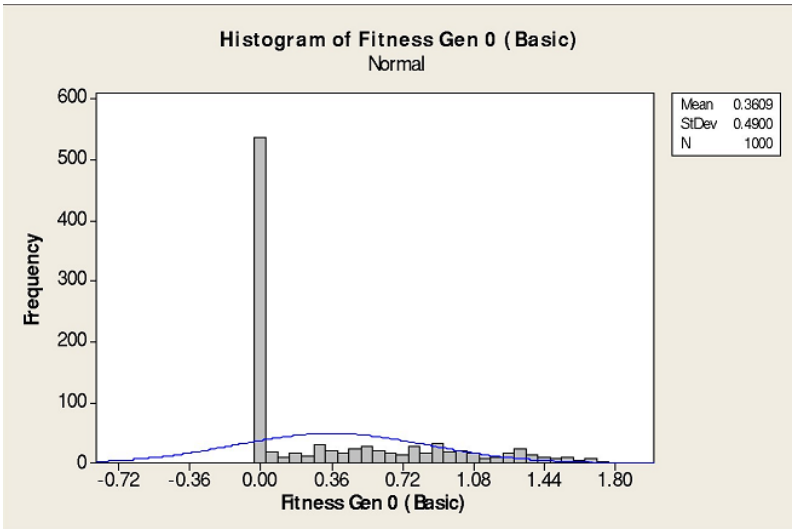

FIGURE 5. Fitness histogram for generation 0

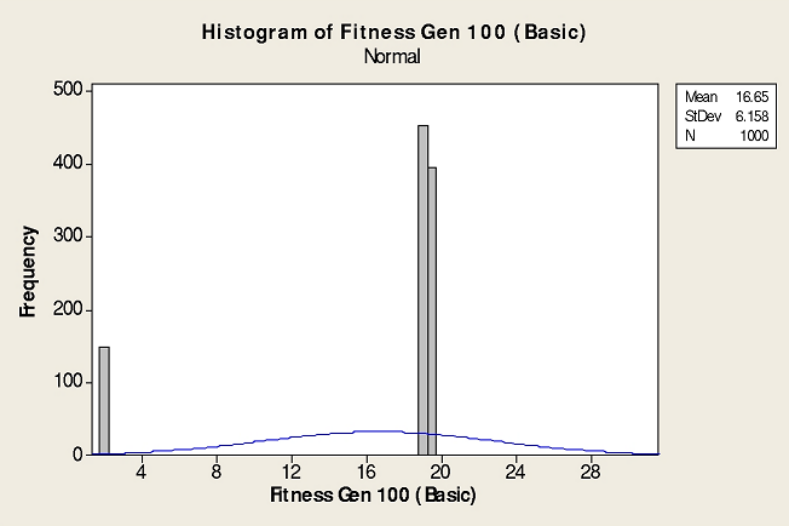

FIGURE 6. Fitness histogram for generation 100

of a small penalty from an initial score value. Only when the average fitness of the population reaches a certain threshold, do we proceed on to the second stage, where a more stringent fitness function is applied. A similar approach was reported in [36]. In this work, they noted that in the early phase, it is more important to optimize the 'feasibility' of solutions rather than its actual fitness. However, they do not totally disregard the actual fitness function in this first stage. Instead they proposed the use of non-dominated individual selection scheme to evolve feasible solutions. In our case, we do not find a need to use sophisticated selection mechanism to evolve feasible individuals. Using the standard tournament selection, our GA implementation does not seem to have any problem in passing by the first stage, and converging to a balanced population comprising largely feasible solutions, with a healthy dose of infeasible ones, as shown in Figure 6. We did probe the use of an alternative selection mechanism, Fitness Uniform Selection Scheme (FUSS) [14], and noted no significant improvement in performance, indicating, as suggested in [19], that the load balancing problem is not likely to be a highly deceptive problems.

4.1.3.2. Concern 2: need for surrogate fitness function. The second concern, the computationally taxing nature of the full simulation-based fitness function, is addressed by running the GA with a surrogate function using a large population, before rerunning it with the full-simulation based evaluation using a smaller population. The use of surrogate function has been considered in several other GA work. For example, in the work of [5], they proposed the use of Markov network to approximate various fitness functions. In [31], authors proposed the use of an adaptive fitness approximation GA (ASAGA) to overcome the problem of selecting appropriate approximation model. ASAGA adaptively chooses and adapt the appropriate model type. In [20], authors developed a framework for determining optimal surrogate functions. Most of the major work on surrogate-based GA to date requires building up approximation models. A problem with this approach, as noted in [29], is that a large number of exact fitness evaluations, in the first place, has to be performed to develop the model. In this work, for the FDTD workload subdivision problem, we avoid this problem, and instead directly use Equation (9) as the surrogate function. Further, we use the surrogate as a mean of producing seeds for a latter full-scale evolutionary optimization process, rather than as a complete replacement of the full-simulation evaluation.

Adaptation of Equation (9) as a surrogate function requires that for each machine in our data set, we compute it is average network transfer rate. Further, we need to ensure, through penalty function, that the optimization of Equation (9) takes memory constraints into consideration.

4.1.3.3. 2-substage load distribution. Based on the the way we are addressing the two concerns in applying GA to workload balancing, it is clear that our approach requires running the second-stage GA in 2 substages. The substages are as follows:

(1) large-population GA with evaluation function based on surrogate and penalty.
(2) small-population GA with evaluation function based on full-simulation and penalty.

The input from the first substage goes on to be part of the population for the second substage.

Note that the same penalty function applies in both stages. Since constraint violation translate to lower fitness value, the essence of our constraint-handling approach is similar to that in [7]. In our tournament selection scheme, for two randomly selected solutions, the followings rules are implicit:

(1) a feasible solution is selected rather than an infeasible solution.
(2) between two feasible solutions or between two infeasible solutions, the one with the higher fitness value is selected

5. **Experimental Results.** We investigate the efficacy of our proposed multi-stage GA approach using 3D cubical FDTD volumes with dimension D^3 where where D is a multiple of 100, with values from 300 to 900. The size of each of the load used are as shown in Table 3.

Each experimentation is repeated multiple times, typically 15 or 20 times, each time using a randomly generated set of 100 computers. As before, the maximum FDTD processing speed, the maximum allocated memory space and the maximum interconnection speed for each set is assumed to vary with normal distribution between the minimum and the maximum values shown in Table 1. Availability patterns of 100 time periods, each of which with length of 10 minutes, are randomly generated.

5.1. **Clustering results and analysis.** We present in this section the result for our GA-based resource clustering approach. The number of GA generations is fixed at 500, and the population size at 500 throughout the evolution. The mutation rate is fixed

at 0.01, and the crossover rate at 0.5. We have used tournament selection scheme to select better individuals for the next generations. Table 4 shows the performance of our clustering approach for different FDTD workloads. A 2-tailed t-test with a confidence interval of 95% was performed, comparing the results of GA with and without seeding. The significance values computed are as shown in Table 5.

First, we consider the performance for a workload of $300 \times 300 \times 300$. From the result, it is clear that GA vastly outperforms the greedy approach. The question is whether or not seeding does have any actual impact. As can be seen in Table 5, there is a significant difference between the performance score when the seeding rates are 0.1, 0.25 and 0.75. Based on the difference value, for a load of $300 \times 300 \times 300$ the best seeding rate seems to be 0.25.

For an FDTD workload of $400 \times 400 \times 400$, as can be seen from the significance values in Table 5, the differences seems to be negligible, between seeded and unseeded GA. In other words, seeding does not help in the case of an FDTD workload of this size. A similar result is obtained for an FDTD workload of $500 \times 500 \times 500$.

The situation is more positive for seeding in the case of an FDTD workload of size $600 \times 600 \times 600$. As indicated in Table 5, there is a significant performance difference between GA with no seeding and GA with seeding rates of 0.1, 0.25 and 0.75. Similarly, seeding seems to be beneficial for an FDTD workload of $700 \times 700 \times 700$. For an FDTD workload of $800 \times 800 \times 800$, however, the result obtained is surprising as GA without seeding (with mean .0548286) actually outperforms GA with seeding 0.25 and 1.0, as can be seen in Table 4 and in the significance values shown in Table 5.

Finally, for an FDTD workload of $900 \times 900 \times 900$, as indicated in the tables, seeding does not bring significant difference in the resulting performance. In fact, an interesting observation that can be made from Table 4 is that the performance difference between the greedy approach and GA-based methods is less (but still significant, with significance value of 0.0298) compared with that for smaller FDTD loads.

As shown in this section, seeding benefit seems to depend on both the workload and the seeding rate. For the largest workloads, that of sizes $800 \times 800 \times 800$ and $900 \times 900 \times 900$, seeding seems to bring no effect. This is likely due to the ineffectiveness of the greedy algorithm used to generate the seeds. In all, however, a seeding rate of 0.1 seems to be the safest option, as at the very least, its performance is not significantly worse than that of an unseeded GA.

5.2. **Load balancing based on surrogate fitness function.** As discussed in Section 4.1.3, for the load balancing stage, we perform a 2-substage optimization. In the first phase, a surrogate function based on Equation (9) [30] is used to evaluate each candidate load-balancing solution. Since the chromosomes used, a sequence of split codes, are

TABLE 3. Volume size for D^3 load, assuming 24 bytes per cell

D	Number of cells	*Volume (MBytes)*
300	27000000	617.98
400	64000000	1464.84
500	125000000	2861.02
600	216000000	4943.85
700	343000000	7850.65
800	512000000	11718.75
900	729000000	16685.49

TABLE 4. Fitness scores for clustering

Load	Measure	Greedy	No Seed	Seeding Rate				
				0.1	0.25	0.5	0.75	1.0
300^3	Mean	.0713231	.1310968	.1326767	.1340032	.1318901	.1329446	.1322749
	Std Dev	.0028727	.0021347	.0017210	.0018685	.0015264	.0021070	.0023479
400^3	Mean	.0499110	.0982985	.0981946	.1009446	.0981732	.0994386	.0989339
	Std Dev	.0082562	.0041648	.0048529	.0039827	.0052687	.0055425	.0064056
500^3	Mean	.0265770	.0706665	.0692137	.0706865	.0701461	.0697806	.0710531
	Std Dev	.0065858	.0047343	.0042349	.0030689	.0053325	.0040543	.0037248
600^3	Mean	.0303977	.0587311	.0601778	.0603163	.0596500	.0601137	.0600630
	Std Dev	.0056142	.0019601	.0014886	.0010832	.0015760	.0011649	.0015437
700^3	Mean	.0448715	.0554987	.0574946	.0572647	.0570652	.0572530	.0572472
	Std Dev	.0124122	.0028864	.0008634	.0005057	.0005857	.0007204	.0011080
800^3	Mean	.0370232	.0548286	.0546098	.0537228	.0541613	.0543691	.0014850
	Std Dev	.0033298	.0010559	.0013779	.0014034	.0013423	.0532479	.0008378
900^3	Mean	.0493121	.0511232	.0516863	.0513298	.0515099	.0511947	.0512364
	Std Dev	.0044781	.0017430	.0018039	.0015753	.0016087	.0015168	.0015605

TABLE 5. T-test between clustering GA with and without seed

Load	Seeding Rate				
	0.1	0.25	0.5	0.75	1.0
300^3	0.021	0.00	0.230	0.024	0.170
400^3	0.947	0.086	0.941	0.525	0.750
500^3	0.35	0.989	0.776	0.585	0.808
600^3	0.018	0.011	0.152	0.024	0.051
700^3	0.005	0.027	0.032	0.026	0.038
800^3	0.618	0.026	0.137	0.344	0.00
900^3	0.366	0.745	0.520	0.905	0.856

hierarchical in nature, the mutation rate should vary at each level, along with the chromosome. The mutation rate should be lowest at the highest level, that is the root, and highest at the lowest level of the hierarchy. Specifically, the mutation probability rate, p_i, for each code, code_i, along the chromosome varies as follows:

$$p_i = p_{base} + i \cdot s \tag{10}$$

Hence, the primary experimental variable is the mutation probability intra-chromosome scaling factor, s, used in the mutation. In our experiments, we vary the scaling factor, using the following values for s : 0.01, 0.05 and 0.1. Further, we study as well the effect of the following two extensions:

(1) feasibility preservation: a mutation is reattempted for as long as it causes a feasible chromosome to become unfeasible.
(2) hill-climbing: a mutation is reattempted for as long as it causes a reduction in fitness.

Each of the figures reported in this section, as in the previous section, was obtain through multiple repetitions (15 to 20). For each load, we consider a single clustered resource pool, and subject the load to GAs with different scaling factor for the mutation. We then use paired t-test to compare the performance results, between GA with scaled mutation rate and GA with a flat mutation rate. We further note that we do not consider the performance measures for load of size $300 \times 300 \times 300$. The reason is that most of the

TABLE 6. Fitness scores for surrogate GA

Load	Measure	No scaling	scaling rate			fp	hc
			0.01	0.05	0.1		
400^3	Mean	33.3202176	33.3296765	33.3445706	33.4102000	33.3386706	33.3029706
	Std Dev	1.9154159	1.9295433	1.9213180	1.9698149	1.9520860	1.9099438
500^3	Mean	21.8757764	21.9065235	18.9642529	21.9020118	18.9795847	16.0774953
	Std Dev	1.9502061	1.9382335	1.4700516	1.9568354	1.4848580	1.2000089
600^3	Mean	23.4223706	23.5532000	23.5200352	23.4690647	23.5857823	23.1826412
	Std Dev	1.9146713	1.9399559	1.9341907	1.9303716	1.9483175	1.9133611
700^3	Mean	24.8240824	25.3901353	25.1115294	25.1601882	25.0460647	24.2824000
	Std Dev	.4641067	.4904735	.2733610	.1608548	.4459373	.5617687
800^3	Mean	23.3983235	23.8981176	24.0033176	24.0728059	23.9981353	24.0414647
	Std Dev	.303451642	.27519376	.358154325	.1921679034	.523278514	.266695723
900^3	Mean	24.9756118	25.7712353	26.4510353	26.5882529	26.2889824	25.9858059
	Std Dev	.5234993	.5285765	.5242757	.3709963	.7208470	.7515927

TABLE 7. Paired t-test between GA with and without scaled mutation

Load	Scaling rate			fp	hc
	0.01	0.05	0.1		
400^3	0.138	0.138	0.000	0.510	0.275
500^3	0.008	0.041	0.004	0.041	0.002
600^3	0.023	0.139	0.465	0.016	0.009
700^3	0.000	0.016	0.001	0.182	0.128
800^3	0.000	0.001	0.000	0.000	0.000
900^3	0.000	0.000	0.000	0.000	0.001

clusters formed for this load comprises of only 2 computers, resulting in a load balancing solution which is too trivial for consideration here.

As in the previous section, the number of GA generations is fixed at 500, and the population size at 500 throughout the evolution. The mutation rate is set to be at 0.01, and the crossover rate at 0.5. And again, we have used tournament selection scheme to select better individuals for the next generations.

As shown in Tables 6 and 7, for a workload of $400 \times 400 \times 400$, a scaling factor of 0.1 leads to a significance difference (increase) in performance compared with a GA with unscaled mutation rate. For all other scaling factors, the significance values are insignificant. One would expect the two extensions above – feasibility preservation and hill-climbing – to result in the best performance, but as can be seen here, this is not the case. More discussion on this will be in the subsections to follow.

A different situation can be seen in the context of FDTD workload of size $500 \times 500 \times 500$. While a scaling factor of 0.01 results in a significant (but slight) improvement from a non-scaled version, there is significant degradation for scaling factor of 0.05, and for the feasibility-preserving and hill-climbing extension. For an FDTD workload of size $600 \times 600 \times 600$, favorable figure are obtained for mutations scaling of 0.01, with and without feasibility preserving extension. In the context of workload of size $700 \times 700 \times 700$, the tables show significantly better performance for mutation scaling factor of 0.01, 0.05 and 0.1. There is however no significant difference when it comes to feasibility-preserving and hill-climbing mutation. For an FDTD workload of $800 \times 800 \times 800$. Scaling factor of 0.01, 0.05 and 0.1, and the feasibility-preserving scheme show significantly better results

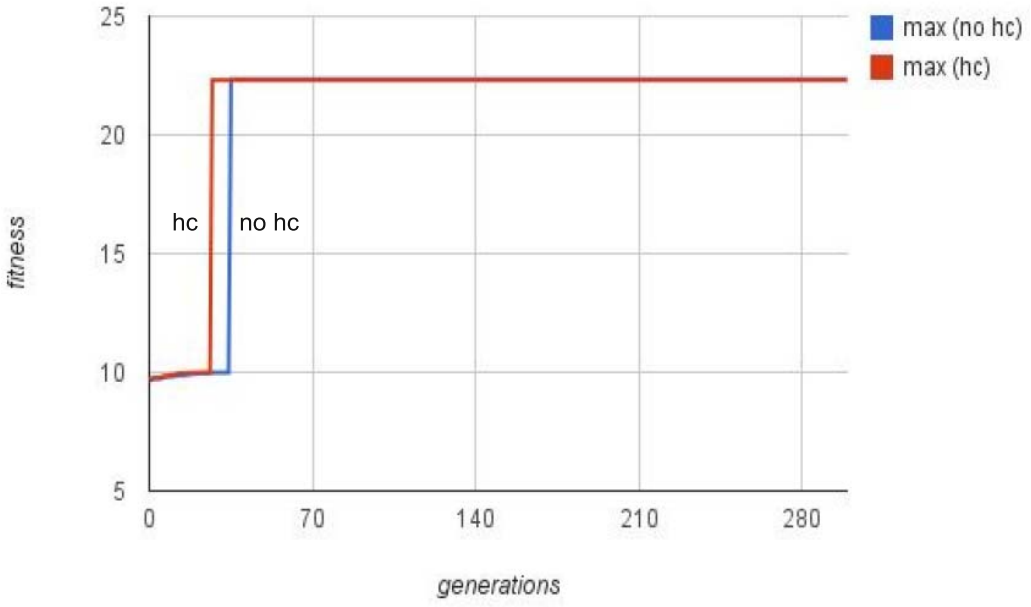

FIGURE 7. Fitness evolution for $500 \times 500 \times 500$

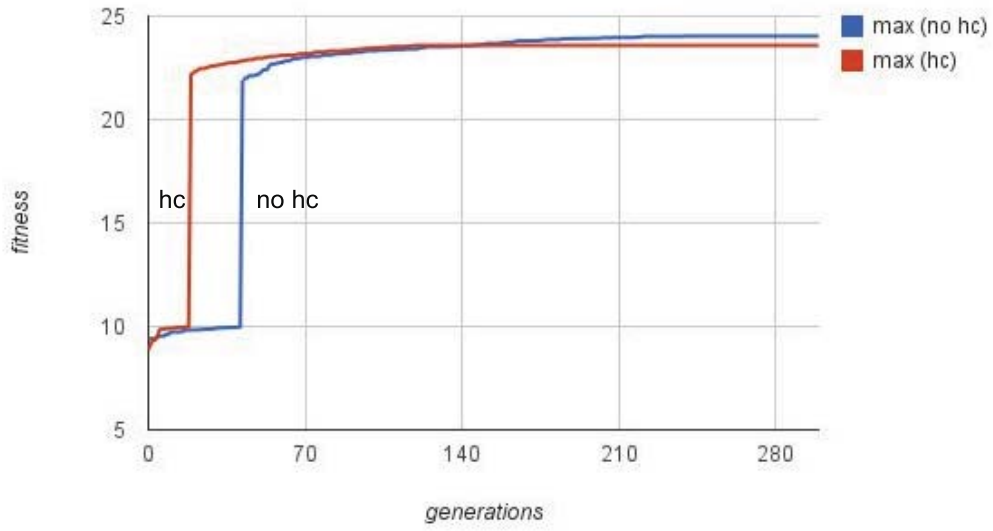

FIGURE 8. Fitness evolution for $600 \times 600 \times 600$

compared with the zero-scaling scheme. Finally, in the case of an FDTD workload of size, $900 \times 900 \times 900$, all the mutation attempts with non-zero scaling, including that with hill-climbing and feasibility-preservation, show significantly better performances compared with non-scaled mutation.

Based on the results presented in this section, we can derive the following conclusion: Feasibility preservation rarely help in improving performance of GA with scaled mutation rate. Hill-climbing does not help at all. The apparent lack of performance boost from both forms of extensions, can be explained with reference to the plot in Figures 7 and 8. The figures show evolution of best fitness for GA with (hc) and without hill-climbing (no hc), in the case of $500 \times 500 \times 500$ and $600 \times 600 \times 600$ workloads. It can be seen that hill-climbing can actually help in accelerating convergence. However, the final fitness value, as shown in the graphs, is significantly better or different compared with that when

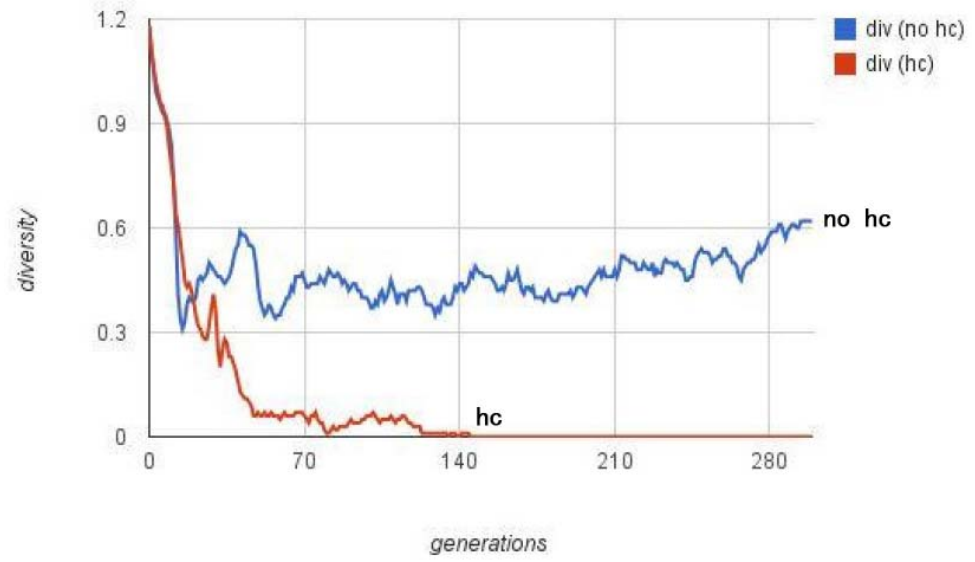

FIGURE 9. Diversity evolution for $500 \times 500 \times 500$

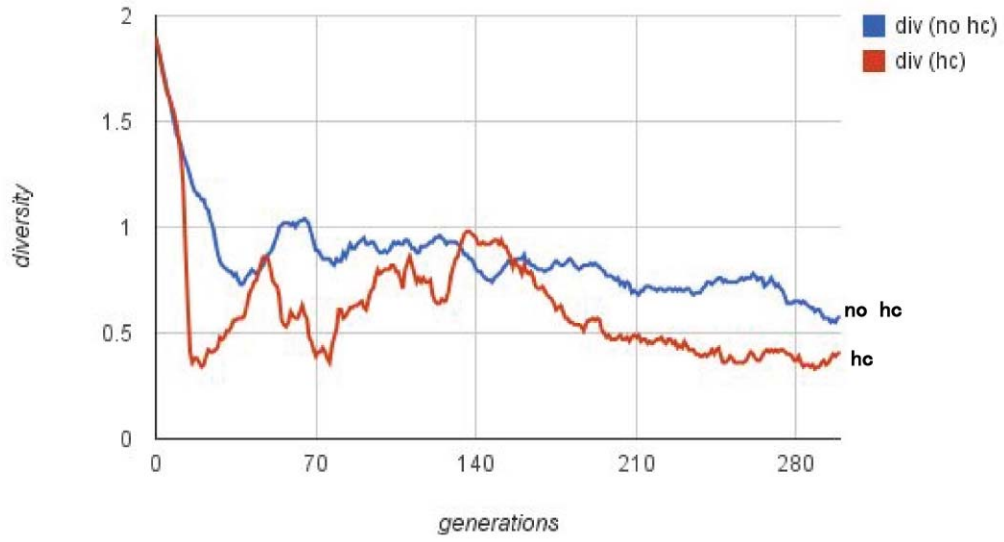

FIGURE 10. Diversity evolution for $600 \times 600 \times 600$

hill-climbing is not deployed. In fact, as can be seen in diversity plots in Figures 9 and 10, hill-climbing may lead to more rapid loss of diversity, which may in turn result in inferior performance. The observation and explanation here applies to our all other test volumes, and to the feasibility-preservation scheme as well.

Hence, the best form of mutation for the surrogate-based load-balancing optimization appears to be that with seeded scaling factor between 0.01 and 0.1. Interestingly, the impact of scaled mutation rate seems to be higher for the bigger volumes. Of course, the conclusion we draw here is based on the experimentations done so far. More work will be done to further validate our observations here.

5.3. **Load balancing based on simulation-based fitness function.** The use of surrogate GA in this paper is meant to produce seeds for a smaller population GA that uses full-simulation for the evaluation of its individuals. In this section, we present the performance results for both seeded and unseeded full-simulation GA. We assume a population

TABLE 8. Performance of simulation-based GA

Load	Measure	Number of Seeds			
		0	**5**	**15**	**30**
400^3	Mean	16.0293867	6886.6666667	7216.8666667	7346.2000000
	Std Dev	10.7292388	870.7542434	139.9269537	168.9937446
500^3	Mean	9.7062180	5605.86666667	5600.00000000	5600.00000000
	Std Dev	.1212901	15.4820941	0	0
600^3	Mean	9.5686220	3596.8000000	3755.4666667	3743.6000000
	Std Dev	.1872287	136.8305521	59.82458485	117.8108654
700^3	Mean	7.4635940	6040.4000000	6445.0666667	6351.5333333
	Std Dev	.7873739	315.5172171	547.0791881	261.9320704
800^3	Mean	9.5686220	3596.8000000	3755.4666667	3743.6000000
	Std Dev	.1872287	136.8305521	59.8245848	117.8108654
900^3	Mean	3.7743073	5955.0000000	6003.6000000	5879.1333333
	Std Dev	3.2278751	147.2582765	192.4632806	248.4827866

TABLE 9. Paired t-test between GA with number of seed = 30 and others

Load	Number of Seeds		
	0	**5**	**15**
400^3	0.000	0.051	0.012
500^3	0.000	0.164	-NA-
600^3	0.000	0.005	0.727
700^3	0.000	0.014	0.604
800^3	0.000	0.114	0.250
900^3	0.000	0.404	0.216

size of 50 for the full-simulation GA, and we use the following number of seeds: 5, 15 and 30, evolving the GA population over just 50 generations (to minimize computational cost). The results are shown in Tables 8 and 9.

As can be seen in the tables, for a load of $400 \times 400 \times 400$, the results for seeded full-simulation GAs are obviously better than unseeded GA. In fact, the unseeded GA could not even find feasible solutions. As explained in Section 4.1.3, our GA implementation first evolve based only on minimization of the penalty function. Each individual starts with a score of 20. Violation of memory constraint results in deduction from the score. If no violation happens, the score is then added with the number of FDTD iterations performed over a fixed time period. Apparently, with the limited population budget and number of generations, the unseeded GA could not even find a feasible solution.

Due to the clear performance superiority of seeded GA compared with the unseeded version, we choose to have the significance values in Table 9 which addresses a different question: Taking 30 as the maximum number of seeds, we consider the question of whether or not a lesser number of seeds will result in a significantly (based on paired t-test) different performance. Table 9 shows the paired t-test significance values of the differences between GA with 30 seeds and that with lesser number of seeds. As can be seen in the table, for a $400 \times 400 \times 400$ workload, there is a significant difference between 30-seeds GA and GAs with 15 seeds or no seed at all.

For a load of $500 \times 500 \times 500$, that table shows that again, unseeded GA failed to find a feasible solution. Further note that 30-seed and 15-seed GA produced the same constant

value (with 0 standard deviation). As shown in Table 9, no t-test can be performed between the values for the 2 GAs. For a load of $600 \times 600 \times 600$, unseeded GA has a very much lower score. The score for 30-seed GA is significantly better than that for 5-seed GA. The same conclusion can be derived for $700 \times 700 \times 700$. For a load of $800 \times 800 \times 800$, again unseeded GA is shown to perform poorly, as in the previous cases. As can be seen in the table, there is no significant differences between a 30-seed GA and GAs with lesser number (5 and 15) of seeds. The same conclusion can be derived for $900 \times 900 \times 900$.

A conclusion that we can draw from the results in this section is as follows: Seeded full-simulation GA certainly perform better than the unseeded counterpart. Further, the actual number of seeds matters less for the largest volumes considered here ($800 \times 800 \times 800$ and $900 \times 900 \times 900$) compared with the case for smaller volumes.

An interesting question to ponder upon at this stage is as to whether or not there was actually any fitness improvement in the simulation-based GA during its iterations. Was

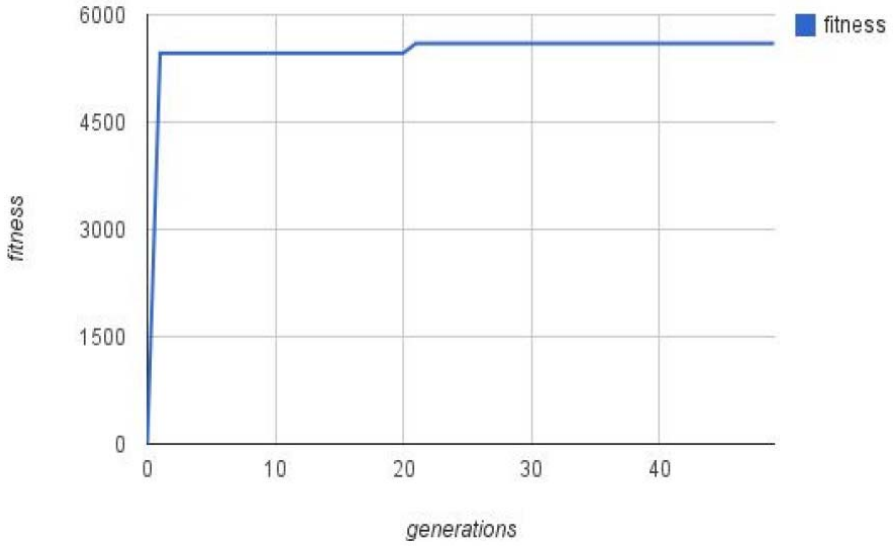

FIGURE 11. Fitness evolution for $500 \times 500 \times 500$

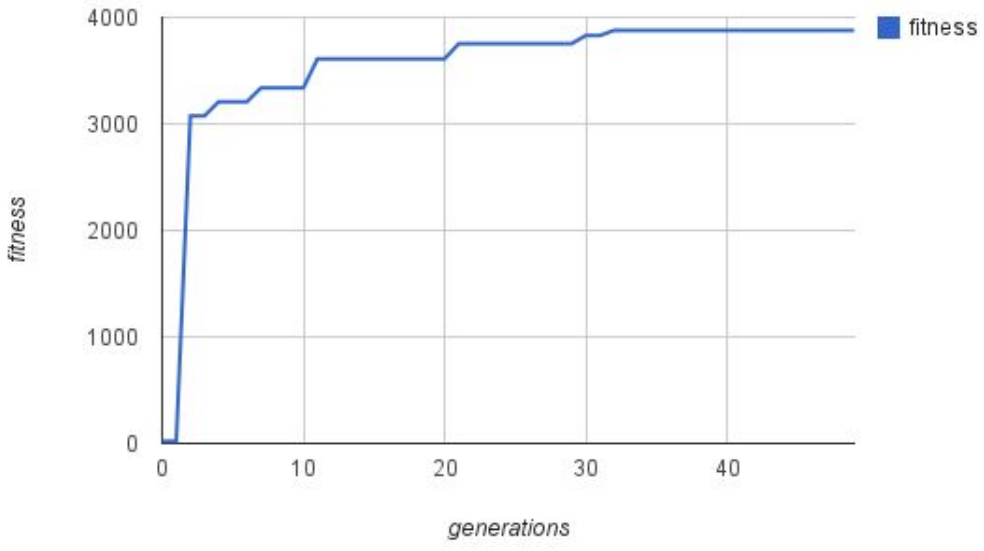

FIGURE 12. Fitness evolution for $600 \times 600 \times 600$

the final fitness value obtained due to the simulation-based GA or solely due to the results obtained from the previous surrogate substage? To address this question, it is insightful to look at the fitness plots for the various workloads. Figures 11 and 12 show the fitness evolution for $500 \times 500 \times 500$ and $600 \times 600 \times 600$ workloads. As can be seen in the graph, the fitness values in the simulation-based GA does improve with increasing generations. We made similar observations in the case of other workloads considered in this paper.

6. **Conclusion.** We have presented in this paper the clustering and distribution of FDTD workload over a large set of computers. Given a large set of computers, as is typically the case in the context of a campus grid, we consider the problem of forming clusters of computers, each of which is to run a single FDTD at a time. The optimization required is two-fold: We need to optimize the clustering, and we need as well to optimize the distribution of FDTD workload within a cluster.

For both optimizations, we proposed some hybrid methods which are centered on GA. For both, we show the benefit of population seeding. Specifically, for the clustering optimization, we use a greedy deterministic method that works quickly to produce fit possible solutions, to be used as seeds for the full GA-based optimization, while for the workload distribution problem, we use a large-population surrogate GA to generate the seeds for a smaller-population full-simulation-based GA. It is worth repeating again that here in this paper, we use surrogate not as a replacement of the actual evaluation function, as is generally the case in existing work, but to prepare the stage for the actual GA optimization.

The method proposed in this work is specific to the problem of workload balancing for FDTD (and other numerical computations that requires similar volume partitioning). Unavoidably, its lack of generality may limits its applicabllity. However, as has been confirmed in the well-known work in [38], a general-purpose universal optimization strategy is impossible, and we have to accept that the optimal strategy is one which has been specialized to the structure of the specific problem under consideration.

Future work includes actual implementation and empirical experimentation with the distributed FDTD, using the clustering and workload distribution strategy as proposed in this paper. Further, we believe that the GA methods proposed here deserve further empirical work that deploy even more rigorous experimentation and statistical analysis.

REFERENCES

[1] E. Abenius, *Time-Domain Inverse Electromagnetic Scattering Using FDTD and Gradient-Based Minimization*, Master Thesis, Department of Numerical Analysis and Computer Science, Royal Institute of Technology, Sweden, 2004.

[2] O. Beaumont, N. Bonichon, P. Duchon and H. Larchevêque, Distributed approximation algorithm for resource clustering, *Proc. of the 15th International Colloquium on Structural Information and Communication Complexity, SIROCCO*, Switzerland, pp.61-73, 2008.

[3] J. Berenger, A perfectly matched layer for the absorption of electromagnetic waves, *Journal of Computational Physics*, vol.114, pp.185-200, 1994.

[4] C. Bierwirth, D. C. Mattfeld and H. Kopfer, On permutation representations for scheduling problems, *Proc. of the 4th International Conference on Parallel Problem Solving from Nature, PPSN*, Germany, pp.310-318, 1996.

[5] A. Brownlee, O. Regnier-Coudert, J. McCall and S. Massie, Using a markov network as a surrogate fitness function in a genetic algorithm, *Proc. of the IEEE Congress on Evolutionary Computation CEC*, Spain, pp.1-8, 2010.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, The MIT Press, 2nd Edition, 2001.

[7] K. Deb, An efficient constraint handling method for genetic algorithms, *Computer Methods in Applied Mechanics and Engineering*, vol.186, no.2/4, pp.311-338, 2000.

[8] A. Elsherbeni and V. Demir, *The Finite Difference Time Domain Method for Electromagnetics with MATLAB Simulations*, SciTech Publishing Inc, 2009.

[9] V. Galtier, K. Mills, Y. Carlinet, S. Bush and A. Kulkarni, Predicting and controlling resource usage in a heterogeneous active network, *Proc. of the 3rd Annual International Workshop on Active Middleware Services*, USA, pp.35-44, 2001.

[10] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, 1989.

[11] C. Guiffaut and K. Mahdjoubi, A parallel fdtd algorithm using the mpi library, *IEEE Antennas and Propagation Magazine*, vol.43, no.2, pp.94-103, 2001.

[12] E. R. Hruschka, R. J. G. B. Campello and L. N. de Castro, Improving the efficiency of a clustering genetic algorithm, *Proc. of the 9th Ibero-American Conference on Advances in Artificial Intelligence – IBERAMIA*, Mexico, pp.861-870, 2004.

[13] E. R. Hruschka and N. F. F. Ebecken, A genetic algorithm for cluster analysis, *Intelligent Data Analysis*, vol.7, no.1, pp.15-25, 2003.

[14] M. Hutter and S. Legg, Fitness uniform optimization, *IEEE Transactions on Evolutionary Computation*, vol.10, no.5, pp.568-589, 2006.

[15] S. Hwang, E. J. Im, K. Jeong and H. Park, An idle computer cycle prediction service for computational grids, *Proc. of the 4th International Conference on Computational Science*, Poland, pp.116-123, 2004.

[16] B. A. Julstrom, Seeding the population: Improved performance in a genetic algorithm for the rectilinear steiner problem, *Proc. of the 1994 ACM Symposium on Applied Computing, SAC'94*, USA, pp.222-226, 1994.

[17] N. Kapadia, C. Brodley, J. Fortes and M. Lundstrom, Resource usage prediction for demand-based network-computing, *Proc. of the 17th IEEE Symposium on in Reliable Distributed Systems*, USA, pp.372-377, 1998.

[18] P. Kudová, Clustering genetic algorithm, *Proc. of the 18th IEEE International Workshop on Database and Expert Systems Applications (DEXA 2007)*, Germany, pp.138-142, 2007.

[19] S. Legg, M. Hutter and A. Kumar, Tournament versus fitness uniform selection, *Proc. of the IEEE Congress on Evolutionary Computation (CEC-2004)*, USA, pp.2144-2151, 2004.

[20] D. Lim, Y. Jin, Y. S. Ong and B. Sendhoff, Generalizing surrogate-assisted evolutionary computation, *IEEE Transactions on Evolutionary Computation,* vol.14, no.3, pp.329-355, 2010.

[21] H. Lin, F. Yang and Y. Kao, An efficient ga-based clustering technique, *Tamkang Journal of Science and Engineering*, vol.8, no.2, pp.113-122, 2005.

[22] Y. Lu, S. Lu, F. Fotouhi, Y. Deng and S. J. Brown, Fgka: A fast genetic k-means clustering algorithm, *Proc. of the 2004 ACM Symposium on Applied Computing, SAC'04*, USA, pp.622-623, 2004.

[23] U. Maulik and S. Bandyopadhyay, Genetic algorithm-based clustering technique, *Pattern Recognition*, vol.33, no.9, pp.1455-1465, 2000.

[24] C. A. Murthy and N. Chowdhury, In search of optimal clusters using genetic algorithms, *Pattern Recognition Letters*, vol.17, no.8, pp.825-832, 1996.

[25] R. Pham, *A Seeded Genetic Algorithm for RNA Secondary Structural Prediction with Pseudoknots*, Master Thesis, San Jose State University, 2008.

[26] O. Ramadan, Three dimensional mpi parallel implementation of the pml algorithm for truncating finite-difference time-domain grids, *Parallel Computing*, vol.33, no.2, pp.109-115, 2007.

[27] C. L. Ramsey and J. J. Grefenstette, Case-based initialization of genetic algorithms, *Proc. of the 5th International Conference on Genetic Algorithm, ICGA*, USA, pp.84-91, 1993.

[28] J. Roden and S. Gedney, Convolution pml (cpml): An efficient fdtd implementation of the cfspml for arbitrary media, *Microwave and Optical Technology Letters*, vol.27, no.5, pp.334-339, 2000.

[29] M. Schmidt and H. Lipson, Coevolution of fitness predictors, *IEEE Transactions on Evolutionary Computation*, vol.12, no.6, pp.736-749, 2008.

[30] R. Shams and P. Sadeghi, On optimization of finite-difference time domain (fdtd) computation on heterogeneous and gpu clusters, *Journal of Parallel Distributed Computing*, vol.71, no.4, pp.584-593, 2011.

[31] L. Shi and K. Rasheed, Asaga: An adaptive surrogate-assisted genetic algorithm, *Proc. of the 10th Annual Conference on Genetic and Evolutionary Computation GECCO'08*, USA, pp.1049-1056, 2008.

[32] K. U. Stucky, W. Jakob, A. Quinte and W. Süß, Tackling the grid job planning and resource allocation problem using a hybrid evolutionary algorithm, *Proc. of the 7th International Conference on Parallel Processing and Applied Mathematics, PPAM'07*, Germany, pp.589-599, 2008.

[33] T. Tometzki and S. Engell, Systematic initialization techniques for hybrid evolutionary algorithms for solving two-stage stochastic mixed-integer programs, *IEEE Transactions on Evolutionary Computation*, vol.15, no.2, pp.196-214, 2011.

[34] H. R. Topcuoglu, B. Demiröz and M. T. Kandemir, Solving the register allocation problem for embedded systems using a hybrid evolutionary algorithm, *IEEE Transactions on Evolutionary Computation*, vol.11, no.5, pp.620-634, 2007.

[35] Y. Wang, Fuzzy clustering analysis by using genetic algorithm, *ICIC Express Letters*, vol.2, no.4, pp.331-337, 2008.

[36] Y. Wang, Z. Cai, Y. Zhou and W. Zeng, An adaptive tradeoff model for constrained evolutionary optimization, *IEEE Transactions on Evolutionary Computation*, vol.12, no.1, pp.80-92, 2008.

[37] Y. M. Wang and W. C. Chew, An iterative solution of two-dimensional electromagnetic problem, *International Journal of Imaging System Technology*, vol.1, no.1, pp.100-108, 1989.

[38] D. Wolpert and W. Macready, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, vol.1, no.1, pp.67-82, 1997.

[39] F. Xhafa, L. Barolli, J. Kolodziej and S. Khan, Genetic algorithms for energy-aware scheduling in computational grids, *Proc. of the IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC*, Spain, pp.17-24, 2011.

[40] F. Xhafa, J. Carretero and A. Abraham, Genetic algorithm based schedulers for grid computing systems, *International Journal of Innovative Computing, Information and Control*, vol.3, no.5, pp.1053-1071, 2007.

[41] X. Yao, An empirical study of genetic operators in genetic algorithms, *Microprocessing and Microprogramming*, vol.38, no.1-5, pp.707-714, 1993.

[42] K. Yee, Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media, *IEEE Transactions on Antennas and Propagation*, vol.14, no.3, pp.302-307, 1966.