Journal of Computational Design and Engineering 4 (2017) 249–255

Contents lists available at ScienceDirect

Journal of Computational Design and Engineering

journal homepage: www.elsevier.com/locate/jcde

Optimization of patch antennas via multithreaded simulated annealing based design exploration

James E. Richie, Cristinel Ababei *

Dept. of Electrical and Computer Engineering, Marquette University, 1515 W. Wisconsin Ave., Milwaukee, WI 53233, USA

ARTICLE INFO

Article history:
Received 2 May 2017
Received in revised form 8 June 2017
Accepted 14 June 2017
Available online 12 July 2017

Keywords:
Microstrip patch antennas
FDTD
Design space exploration
Simulated annealing
Multithreaded parallelization

ABSTRACT

In this paper, we present a new software framework for the optimization of the design of microstrip patch antennas. The proposed simulation and optimization framework implements a simulated annealing algorithm to perform design space exploration in order to identify the optimal patch antenna design. During each iteration of the optimization loop, we employ the popular MEEP simulation tool to evaluate explored design solutions. To speed up the design space exploration, the software framework is developed to run multiple MEEP simulations concurrently. This is achieved using multithreading to implement a manager-workers execution strategy. The number of worker threads is the same as the number of cores of the computer that is utilized. Thus, the computational runtime of the proposed software framework enables effective design space exploration. Simulations demonstrate the effectiveness of the proposed software framework.

1. Introduction

Microstrip or patch antennas are becoming increasingly popular because they can be printed directly onto circuit boards. Patch antennas are low cost, have a low profile and can be easily fabricated. They are becoming very widespread within the mobile phone market. The design process is affected by many parameters that control the properties of such antennas. To identify the parameters that offer the best performance requires the exploration of the design solution space defined by such parameters. Such exploration implies multiple numerical simulations, which can take long computational runtimes.

In this paper, we present a software framework that implements an automated design space exploration (DSE) for patch antennas. The exploration is done with a multithreaded simulated annealing (SA) optimization algorithm. Each explored design solution is evaluated with the MEEP simulator. To this end, the main contributions of this paper include: (1) The SA based tool that can seek and identify the optimal patch antenna design and (2) The SA algorithm is implemented with a multithread approach, which provides a speed-up of the execution time by a factor of 7.56x when executed on an 8-core processor compared to the execution on only one thread on a single core.

The remainder of this paper is organized as follows. In the next section, we briefly review related literature. Then, we present background information and formulate the problem of optimization for patch antennas. In Section 4, we present the proposed software framework, which consists of a parallel via multithreading implementation of a simulation annealing optimization algorithm to solve the patch antenna design problem. Section 5 reports simulation results obtained on a machine running on a processor with eight cores, followed by a discussion in Section 6. Finally, we conclude and summarize our contributions in Section 7.

2. Related work

With the advent of wireless communication systems, there has been significant work done on the design of antennas. Particularly, research efforts were focused on antennas used in wireless local area network (WLAN) and Bluetooth applications. Instances of such efforts include Gondarenko and Lipson (2008), Hansen, Zheng, Perederey, and Hesselink (2011), Jayasinghe and Uduwawala (2015) and Meng and Sharma (2016) and the references therein. For example, the recent study in Jayasinghe and Uduwawala (2015) presented the design of a compact planar inverted F antenna (PIFA) for 2.4 GHz and 5 GHz bands. In order to optimize the geometry (the shorting pin position and the feed position of their patch antenna), the authors used a genetic algorithm (GA) optimization approach that employed simulations with the Ansys' high frequency structure simulator (HFSS). However, they did not report computational runtimes of their optimization

Peer review under responsibility of Society for Computational Design and Engineering.

* Corresponding author.
E-mail addresses: james.richie@marquette.edu (J.E. Richie), cristinel.ababei@marquette.edu (C. Ababei).

http://dx.doi.org/10.1016/j.jcde.2017.06.004
2288-4300/

approach, which is not publicly available. In addition, HFSS is a commercial tool, which is not free. As another recent example, the study in Meng and Sharma (2016) reported the design of a single feed dual-band miniaturized microstrip patch antenna for WLAN communications. They also use Ansys HFSS and several manual design optimization techniques including the creation of selected slots to the E-shaped inner patch and the adoption of a non-homocentric design style.

In contrast with these works, the core simulator that we use to evaluate each design solution point during separate iterations of the SA algorithm is MEEP (MEEP, 2017; Oskooi et al., 2010). MEEP is a popular open-source free implementation of the finite-difference time-domain (FDTD) method for simulating electromagnetics problems. It can be used to conduct a variety of simulations but for given specific design problems only; it does not include optimization algorithms to seek optimal solutions. Noteworthy is that its implementation is very efficient due to various code optimization techniques. Thus, it serves well the purpose of being an efficient simulator that we can call multiple times to evaluate patch antenna designs.

However, if simulations are required to be done for three dimensional structures, such as in the case of numerical electromagnetic calculations in nanophotonics, the computational runtimes of such FDTD simulators can increase significantly due to large number of variable parameters. Moreover, when applying optimization schemes that require a full-field solution at each iteration of the optimization loop, such as genetic algorithms (Gondarenko & Lipson, 2008; Jayasinghe & Uduwawala, 2015), adjoint optimization methods (Hansen et al., 2011), or simulated annealing in the case of this paper, the ability to efficiently find a solution becomes limited by the computational speed of the full-field simulator. To address such computational runtime issues, one can pursue one of the following two approaches. The first approach is to focus on the FDTD simulator itself in order to speed it up via some parallelization technique. In this category, the study in Wahl, Ly-Gagnon, Debaes, Miller, and Thienpont (2013) uses CUDA programming to speed-up a 3D-FDTD solver by running it on computers equipped with Graphical Processing Units (GPUs). The authors benchmark their GPU based implementation against MEEP and reported significant speed-up for computing the absorption efficiency of a metallic nanosphere.

Note that these FDTD simulators are designed to be used for just a single simulation of a design or structure of interest at a time. They do not conduct any optimization in the sense of seeking to identify the best combination of design parameters that would provide the desired design characteristics. Such design parameters are assumed to be known and directly specified as input to these simulators. Therefore, the second approach to address computational runtime issues, applicable in the case of iterative optimizations is to parallelize the optimization algorithm itself. The simulation and optimization framework proposed in this paper falls in this category. We employ MEEP as a point-tool simulator to develop an optimization approach whose objective is to find the best solution for a patch antenna design. To make it computationally efficient, we use multithreading as a parallelization technique, which is implemented using a manager-workers strategy that allows us to run multiple MEEP simulator instances concurrently, thereby speeding up the design solution space exploration. We would like to emphasize that, aside from finding the optimal patch antenna design via the simulated annealing optimization, actually, equally important contributions of our work include: our approach is efficient because it uses the multithreading based speed-up technique, the implementation is versatile in that it can easily be changed to replaced the open-source MEEP simulator with other simulators such as Ansys's HFSS used by previous works, it will be made publicly available as it is constructed with only free tools.

3. Background on patch antennas

3.1. The MEEP model for patch antennas

The MEEP C++ source code is freely available under the GNU GPL license. Documentation is available on the MEEP Wiki pages (MEEP, 2017), including tutorials and reference material. Several examples are also part of the software package. The software can be executed using a script file in the Scheme language, or by writing C++ code that performs the simulation. In this paper, we use the C++ interface.

In MEEP, the fundamental geometric size is the *block*. One typically chooses a distance a that is one block. All quantities are then based off the block size. Each block is broken into cells using the resolution parameter. MEEP also uses *dimensionless* units, $\epsilon_o = \mu_o = c = 1$, where c is the speed of light. All distances are converted by multiplying the number of blocks by a, and frequencies are converted using $f_{meep} = fa/c$, where f_{meep} is the MEEP frequency and f is the frequency in Hz.

A typical microstrip patch antenna consists of a rectangular patch of metal over a dielectric substrate backed by a ground plane. The patch has length L and width W, as shown in Fig. 1. The dielectric height is h, as shown in Fig. 2. Roughly speaking, L is near one half-wavelength. The radiation can be modeled as the fringing of the electric field along thin slots of length W at the two edges separated by the distance L. The radiation pattern of the antenna is typically a broad beam with maximum near broadside. The pattern resembles an array of two elements (the slots) separated by L and fed in phase. The bandwidth of a rectangular patch antenna is typically quite small. Design equations for patch antennas are available and can be found in Balanis (1997) and Stutzman and Thiele (1998).

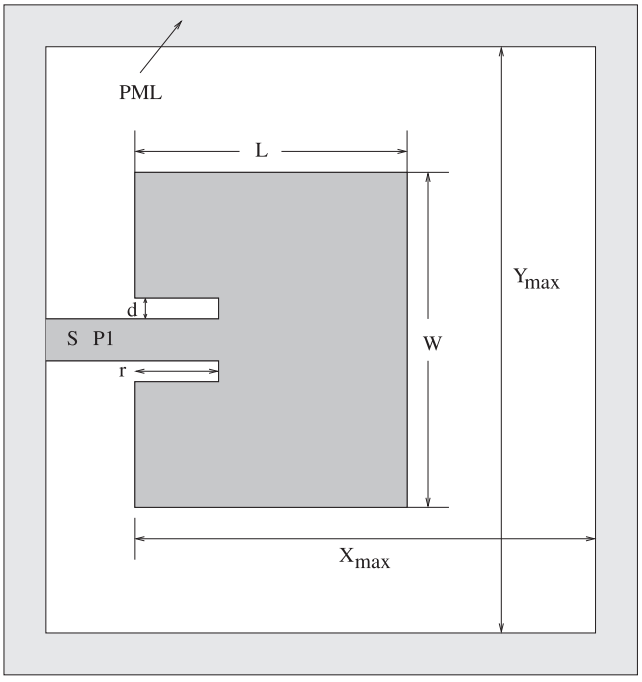


Fig. 1. Geometry and dimensions for the MEEP model of the $L \times W$ patch antenna. The source (S) is dTL from the PML edge; port 1 (P1) is dS from the source; and the patch antenna is dP1 from port 1.

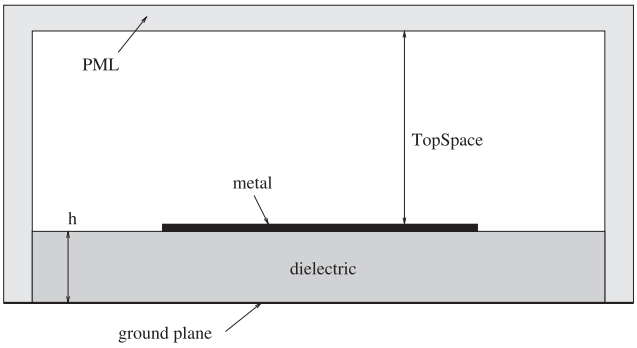


Fig. 2. Cross section of the MEEP model.

The MEEP model for the patch antenna is shown in Figs. 1 and 2. The patch is fed using a 50Ω microstrip line. The antenna impedance at the edge of the antenna is typically around 250Ω, and the impedance falls to zero as the feed point moves to the center of the patch. To match the antenna impedance to the 50Ω feed line, a recess gap is cut into the patch antenna by a distance r, as shown in Fig. 1.

The cross section shown in Fig. 2 includes a ground plane and the dielectric substrate. The substrate has a relative permittivity of 2.33 and a loss tangent $\delta = 0.0011$. The height h is 0.1575 cm. The metallization for the feed line and patch have a thickness of a single cell (0.3 mm in our simulations). The model extends above the antenna by TopSpace = 0.8 cm. On all sides of the model, except the ground plane, there is a perfectly matched layer (PML) to absorb waves and avoid reflections off the ends of the model.

Simulation of the patch antenna requires three general tasks: defining the geometry and materials in the model; specifying the source or incident field; and declaring the desired output data. In MEEP, the size of the structure is defined and the geometry is specified using material functions. Material functions are passed a location within the structure and return the material parameter at that location. Dielectric functions are used to describe both the substrate and the metallization (using $\epsilon = -\infty$). We assume perfectly conducting metal for the patch and microstrip.

The source is a Gaussian pulse of the electric field in the z direction (perpendicular to the plane of Fig. 1). The source location is under the feed line at the "Source" location indicated in the figure. A MEEP flux plane is defined in the cross section of the substrate at the Port 1 location. The flux plane is slightly wider than the feed line and is entirely within the substrate. The output is the net power passing through the flux plane over the frequency range specified by the Gaussian pulse source.

To collect reflection coefficient data $|S_{11}|$, a reference MEEP simulation is performed once. The reference model consists of no patch antenna and the 50 Ω line extends across the entire model space. The source launches a wave that is absorbed at both ends by the PML layer. The power measured at port 1 for the reference simulation is the input power to the antenna, P_{in}. The reference simulation data is collected and stored in a file that is read by every patch antenna simulation.

Each simulation of a candidate patch antenna also collects the net power at port 1. Net power means the power flowing toward the antenna (equal to P_{in}) minus the power that is reflected by the antenna (P_{rfl}). Denote the power data for an antenna simulation as P_{ant}. The power reflection coefficient is the power reflected normalized by the power incident on the antenna. Therefore, since $P_{ant} = P_{in} - P_{rfl}$, the reflection coefficient (in dB) denoted as $|S_{11}|$ is computed by Balanis (1997) and Stutzman and Thiele (1998):

$$|S_{11}| = \frac{P_{in} - P_{ant}}{P_{in}} \quad (1)$$

Table 1
Dimensions used in MEEP model of patch antenna.

Fixed dimensions	Distance (cm)
X_{max}	9.8
Y_{max}	17.0
$dP1$	0.63
dS	0.47
dTL	0.32
PML thickness	0.16

For example, consider a matched antenna. The power reflected would be 0. Therefore, $P_{in} = P_{ant}$ and $|S_{11}|$ is zero. If the antenna is not matched, the reflected power reduces P_{ant} and $|S_{11}|$ increases. Note that the use of power means that no phase information is available.

As an illustrative example, we simulated a patch antenna design whose model parameters are shown in Figs. 1 and 2. The recess distance r has a gap on each side of the line of width d. The distances X_{max} and Y_{max} are chosen so that there is at least $6h$ of additional ground plane (Garg, Bhartia, Bahl, & Ittipiboon, 2001). The fixed dimensions in the MEEP model are listed in Table 1.

The results of an example MEEP simulation are shown in Fig. 3. This is just an illustrative example, whose center frequency is $f_c = 2400$ MHz and the bandwidth is $BW = 63$ MHz (2.6%). The bandwidth is calculated as the frequency bandwidth given by the two intersection points between the plot from Fig. 3 and the horizontal line at the −6 dB threshold. Such a resonance frequency is used for example in Bluetooth wireless communications. In this particular example (a simple textbook design), the dimensions of the patch antenna are $L = 4$ cm, $W = 4.9$ cm, $r = 1$ cm, and $d = 0.2$ cm. The figure shows $|S_{11}|$, the reflection coefficient in dB for the frequency range of 2.1 GHz to 2.7 GHz.

3.2. Patch antenna design space

The model that we use in this paper has four key dimensions that affect the performance of the antenna, including: length L, width W, recess r, and gap width or depth d. The length L of the patch strongly influences the resonant frequency; variations in the width W will vary the edge impedance, Z_A. As the edge impedance changes, the recess distance changes to remain near the 50 Ω feed point. The gap width d maintains isolation between the antenna and the feed line.

To sample the space of solutions, a range of values for each of the four parameters has been chosen, as listed in Table 2. The ranges of these parameters were selected based on our prior experience and familiarity with this antenna design. While these ranges

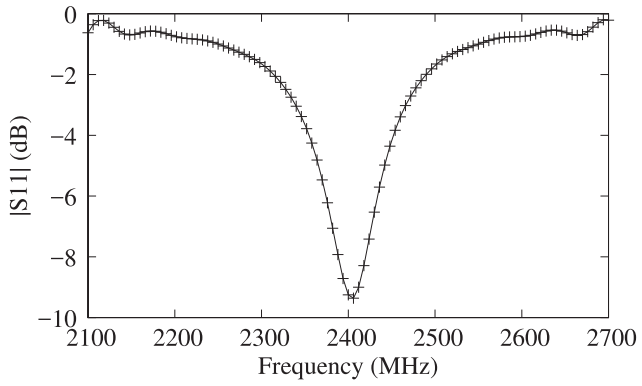


Fig. 3. Sample data from a single patch antenna MEEP simulation example showing the reflection coefficient $|S_{11}|$ vs. frequency.

Table 2
Parameter Ranges for Antenna Optimization.

Parameter	Range (cm)	Delta (cm)
Length L	3.8–4.2	0.05
Width W	2.0–7.25	0.75
Recess r	0.0–2.1	0.3
Depth d	0.05–0.4	0.05

are selected such that they make sense, they should also be constrained also by the geometry of the space where the actual antenna is to be used. These four parameters with their ranges of values effectively define the solution space, which we explore using the simulated annealing (SA) based design space exploration (DSE) implemented by the software framework presented in the next section.

4. Proposed software framework for automated design space exploration for patch antennas

In this section, we present details on the proposed simulation and optimization framework. We first describe its block diagram. Then, we discuss the simulated annealing algorithm used for the optimization of the search process. Finally, we present details on the multithreaded implementation for speed up.

4.1. Block diagram of proposed framework

In this paper, we propose to search for the optimal design of a patch antenna like the one described in the previous section using an automated design space exploration. The top-level block diagram of the proposed software framework is shown in Fig. 4.

The core of the proposed software framework is the simulated annealing algorithm (described later in the next subsection) shown in the center of the diagram from Fig. 4. This algorithm consists of an optimization loop, where during each iteration a new solution point from the design space is generated and evaluated. The design solution space is defined by the antenna parameters that are provided as permissible ranges, which need to be swept during the search. Thus, a solution point is completely determined by actual values of these parameters, selected from within their ranges of permissible values. These ranges are provided by the user and are represented by the box at the top of the diagram in Fig. 4.

The evaluation of a solution point is done by running a complete MEEP simulation. The MEEP simulation is done by invoking an instance of the MEEP simulator (MEEP, 2017; Oskooi et al., 2010). At the end of each MEEP simulation we have calculated the center frequency, f_c, and the bandwidth, BW, for the current design point characterized by a given set of design parameter values. The center frequency and the bandwidth will then be utilized to calculate the *cost* of the current design point, which in turn will be used inside the simulated annealing algorithm to derive a probability to accept this design solution.

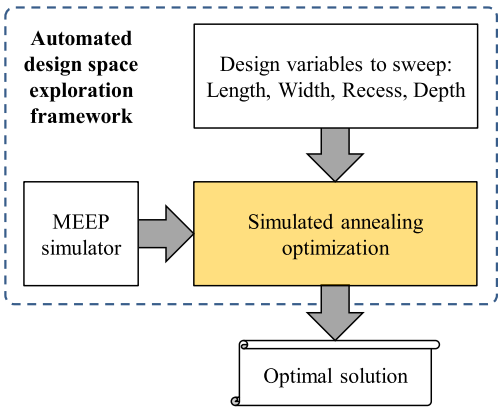


Fig. 4. Proposed software framework for optimization of patch antennas.

4.2. Simulated annealing optimization

We have chosen simulated annealing (SA) for our optimization because of its popularity for solving multi-objective problems and because it is relatively easy to implement. SA is a probabilistic algorithm that has been around for several decades (Cerny, 1985; Kirkpatrick, Gelatt, & Vecchi, 1983). It is especially known for the inherent ability of finding the global minimum in optimization problems that also have multiple local minima. To work with only one total cost, the SA approach expresses the total cost of a multi-objective optimization problem as a weighted sum of the individual costs.

The pseudocode of the SA algorithm is shown in Fig. 5. SA is essentially an iterative algorithm that conducts a random search through multiple solutions toward the best solution. It starts with one or more initial solutions. Then, it continuously generates new solutions from the previous ones. This generation is achieved by calling the function *GenerateNeighborSolution*() in Fig. 5. New solutions are initially accepted even if their overall cost increased compared to previous solutions. As the algorithm progresses, such solutions are less likely to be accepted and only solutions that improve the cost are more likely to be accepted. In this way, if the search initially gets trapped in local minima, the algorithm escapes by allowing the acceptance of worse solutions also during the first part of the annealing process.

The *temperature* variable inside the algorithm is a critical component as it is employed in calculating the probability to accept newly generated solutions. It is updated during each iteration by a call to the function *CalculateTemperature*(), which in our implementation is subtracting a constant amount from the previous value. However, this could be changed such that the annealing is steeper in the beginning and more leveled towards the end. The acceptance probability is computed such that to be high during the early iterations and to be low during the final iterations of the algorithm. In this way, during the early iterations, even solutions that do not improve the total cost can be accepted. This, in turn, helps to move away and escape getting trapped in potentially local minima solutions.

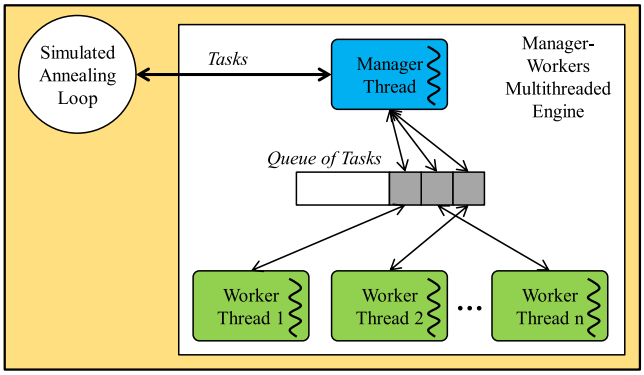
The process of generating new solutions during the iterative search involves *moves* done on the previously generated solutions. The meaning of a move depends on the application at hand, but it is usually some form of local alteration or change of previous solutions. For instance, the process of swapping the visiting order of any two successive cities can represent a move in the well known traveling salesman problem (TSP). In the TSP, a solution is defined as the order in which all the cities will be visited. Changing the order in which two cities are visited is an effective move, which is also easily implemented in computer programs.

In the case of our problem in this paper, a solution is defined as a given set of selected values for the parameters to sweep. This set of values represents a point in the four dimensional solution space, which needs to be explored. To generate a new solution point, we randomly select one of the four design parameters and change its values to a different one (also selected randomly) from the permissible ranges. Each new solution must be evaluated by calculating its *total cost*, which requires a new MEEP simulation to be able to calculate individual costs first. The MEEP simulation is performed by a call to the function *RunMEEPSimulation*(), which, under the hood of the multithreaded framework described later in Fig. 6,

Algorithm: Simulated Annealing

In: Define cost, $Cost$; Cooling strategy, $CalculateTemperature$; Initial high temperature, T_{max}; Max number of iterations, M
Out: Best solution, S_{best}
$k \leftarrow 0$
$S_{current} \leftarrow GenerateInitialSolution()$ // radomly
$RunMEEPSimulation(S_{current})$
$CalculateCosts(S_{current})$
$S_{best} \leftarrow S_{current}$
while $k < M$ **do**
  // Implement a *move* by randomly selecting one of the four parameters and also randomly selecting a new value for it
  $S_k \leftarrow GenerateNeighborSolution(S_{current})$
  $RunMEEPSimulation(S_k)$
  $CalculateCosts(S_k)$
  $T_{current} \leftarrow CalculateTemperature(k, T_{max})$
  if $Cost(S_k) \leq Cost(S_{current})$ **then**
    // Downhill move: accept it all the time
    $S_{current} \leftarrow S_k$
    if $Cost(S_k) \leq Cost(S_{best})$ **then**
      $S_{best} \leftarrow S_k$
    end if
  else
    // Uphill move: accept it sometimes only
    $r \leftarrow Random(0, 1)$
    if $r < Exp(\frac{CostS_{current} - CostS_k}{T_{current}})$ **then**
      $S_{current} \leftarrow S_k$
    end if
  end if
  $k \leftarrow k + 1$
end while
Return S_{best}

Fig. 5. Pseudocode of the simulated annealing (SA) algorithm for finding the best patch antenna design characterized by the optimal set of parameters that minimize both $Cost_{CenterFreq}$ and $Cost_{BW}$.



Multithreaded simulated annealing optimization

Fig. 6. The simulated annealing optimization block from Fig. 4 is implemented on top of a manager-workers multithreaded engine.

translates into the execution of the MEEP simulator instance on one of the worker threads. We express the total cost as the weighted summation of the individual costs:

$$Cost = \alpha * Cost_{CenterFreq} + (1 - \alpha) * Cost_{BW} \quad (2)$$

where $\alpha \in [0, 1]$ is a weighting parameter, which can be used to give higher priority to any of the two individual cost components. For example, if we wanted to give a higher priority to the center frequency as an objective, we could use $\alpha = 0.75$. In this way, the weight for the bandwidth individual cost will be 0.25. Because in our case both individual costs are equally important, we use $\alpha = 0.5$. In the above equation, the center frequency and bandwidth individual costs are calculated at the end of the MEEP simulation for the current design solution point. More specifically, the $Cost_{CenterFreq}$ is defined as the absolute difference between the desired center frequency and the center frequency of the current design solution point. The $Cost_{BW}$ is defined as the absolute difference between an ideal large bandwidth and the current bandwidth. By defining this cost also as a difference or *distance* from an ideal bandwidth we effectively transform the objective of bandwidth maximization into a minimization problem. Noteworthy, in the actual implementation, the above costs are normalized so that their absolute numerical values are always in the range $(0, 1)$. The normalization is necessary because individual costs may have values that are not comparable as a range and that could result in situations where an individual cost could overwhelm the others. We do normalization by taking the ratio between individual costs as calculated from the data obtained from the MEEP simulation and their maximum possible value. The calculation of normalized individual costs as well of the total cost of a given solution is done by calling the function *CalculateCosts*() in Fig. 5. Randomness is achieved by using the standard function *rand*() available with the C++ compiler; this function is seeded differently with the current time during each execution of our tool.

Another key aspect in implementing an effective SA algorithm is the ability to initially generate solutions that may be far from the existing solutions while gradually restricting the *distance* between the parent and child solutions. In this way, a good coverage of the solution space and rapid convergence is ensured. In our case, that is achieved in the following way. The new values for the randomly selected parameters can be selected from vicinities of their current values and these vicinities can be allowed to be large at the beginning of the annealing process and then be restricted to smaller values during subsequent iterations.

4.3. Parallelization via multithreading

One of the main contributions of this paper is that we provide a parallel implementation of the the simulated annealing based design space exploration (DSE) algorithm in order to reduce the overall computational runtime by taking benefit of the readily available multicore processors today. Despite the fact that parallel computing has been thought of for a long time (Gill, 1958; Wilson, 1994) it is still a challenging task to find the most appropriate parallelization technique and application transformation that would maximize the benefit of parallelism. Common parallelization techniques include distributed computing, multithreading (Ababei, 2009; Andrews, 1999; Yang, 1991), and graphics processing units (GPUs) (Owens, 2007; Owens et al., 2008).

Here, we opt for a multithreaded implementation of the main loop of the simulated annealing algorithms because of two reasons. First, it does not require us to modify the MEEP source code, which has been already optimized for runtime. This has also the benefit that the overall proposed framework can be easily extended by replacing the MEEP tool with another if so desired. Second, the

simulated annealing algorithm itself through its iterative nature lends itself nicely to parallelization via multithreading. Indeed, we can launch multiple threads to perform multiple MEEP simulations concurrently in order to evaluate multiple design solution points, thereby speeding up significantly the solution space search process.

The multithreaded implementation of the design solution space exploration through the simulated annealing optimization algorithm is shown in Fig. 6. In this figure, we describe in greater detail the central block from the diagram in Fig. 4 in order to illustrate the mechanics of the multithreading approach.

The proposed multithreaded implementation uses a manager-workers multithreading approach. The *manager* thread is responsible with the communication between the multithreading engine and the host application, which in this case is the simulated annealing loop, as shown in Fig. 6. The host application creates *tasks*, which are taken by the manager thread and placed into a *queue*. Each task basically represents a patch antenna design solution point, which we need to evaluate through a MEEP simulation as part of the design solution space exploration process. Running a MEEP simulation instance and collecting the results from the simulation is done via tasks, which facilitate the cost calculations discussed in the previous section and Fig. 5. The tasks from the queue then are processed individually by the *worker* threads. In our implementation, we create a number of worker threads equal to the number of cores on the processor that the user's machine has. In this way, we maximize the benefits of parallelization. The worker threads, basically, run separate MEEP simulations concurrently for the corresponding design solution points on all cores of the processor. After a given worker thread finishes a task from the queue, the manager thread dispatches the result back to the host application. In our case, the host application is responsible with implementing the logic of the simulated annealing algorithm that we described in the previous section and illustrated in Fig. 5. Thus, it is the host application that interprets the results received from the manager thread and decides if a certain design solution becomes the *current* and/or the *best* solution so far. The host application continues to generate tasks that are passed to the manager thread until a maximum number of tasks has been created or the annealing temperature has reached zero or no improvement in solution quality (measured through cost) has been observed over a prescribed number of recent solution evaluations.

5. Simulations results

In this section, we present simulation results that we obtained using the proposed software framework. All our simulations are done on a machine that has an Intel Xeon CPU E5-1620 processor, 3.60 GHz x 8 cores, 16 GB memory, and runs Linux Ubuntu 14.04 operating system. We set-up the optimization problem for the patch antenna design, where the design parameters that define the solution space are as shown in Table 2 discussed in Section 3. In other words, during the design solution space exploration done as part of the simulated annealing optimization algorithm, we sweep four design parameters that include antenna length, width, recess, and depth. The maximum number of iterations inside the SA algorithm is set to 100. The desired (i.e., ideal) characteristics for the patch antenna are a center frequency of $f_c = 2400$ MHz and a bandwidth of $BW = 100$ MHz. These ideal values are used in the cost calculations inside the SA algorithm. The execution of our tool is done with the maximum number of possible threads on the machine that is used. In our case, that is a number of 8 threads, equal to the number of available cores. The total execution time measured as *wall time* is 6.95 days. The total *cpu time* is 52.59 days. This means that running the multithreaded simulation

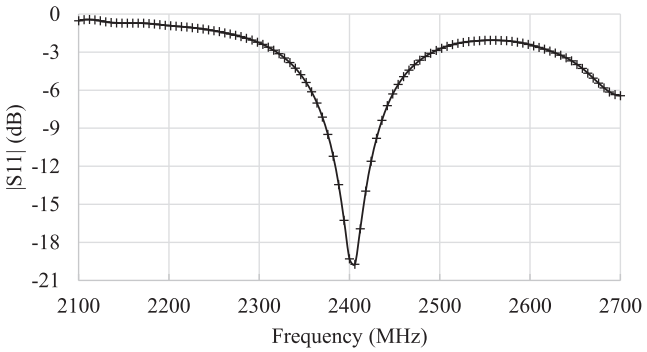


Fig. 7. Reflection coefficient $|S_{11}|$ vs. frequency for the patch antenna MEEP simulation of the best solution found during the design space exploration.

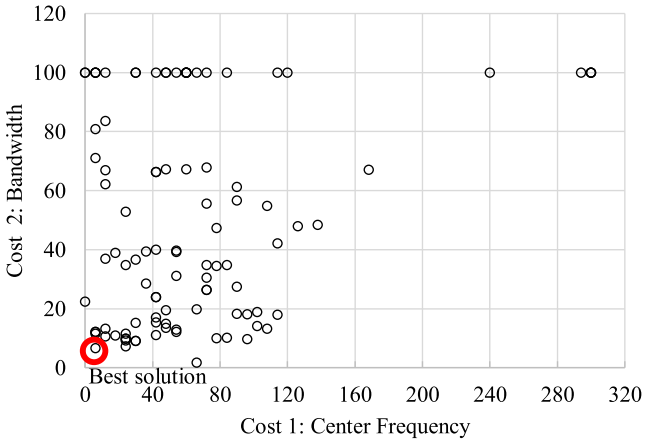


Fig. 8. SA evaluated 100 different solution points during the SA based design space exploration process. The *best solution* is indicated with a red circle.

provided a speed up of 7.56x on the 8 core processor. The best solution found at the end of the design space exploration is characterized by a center frequency $f_c = 2406$ MHz and bandwidth of $BW = 93.41$ MHz. The actual values of the design parameters corresponding to this best solution are: length $L = 3.95$ cm, width $W = 7.25$ cm, recess $r = 0.6$ cm, and depth $d = 0.15$ cm. The sample data corresponding to this best solution is shown in Fig. 7.

Our simulation and optimization framework is orchestrated to record detailed information about the evaluated solution points as well as the evolution of the cost inside the simulated annealing algorithm. For example, Fig. 8 shows all solution points evaluated during 100 iterations. The x axis of this plot represents the center frequency cost and the y axis represents the BW cost, both measured as absolute distances from the ideal/desired values. The best solution point is the one indicated by the data point on the bottom left-hand side of this plot.

Finally, Fig. 9 shows how the normalized cost varied during the annealing process. This plot provides useful information about the convergence of the annealing process. In our context, we say that the algorithm converged when the change in the cost value does not change significantly anymore. We can see that the algorithm converged roughly after about 80 iterations. The cost value does continue to fluctuate even after, but these fluctuations are small and they would continue for more iterations due to the discrete nature of the problem, which works with four design parameters taking discrete values from pre-defined ranges. This information can be utilized to gain further insights into how one could calibrate and fine-tune the design space exploration to further improve

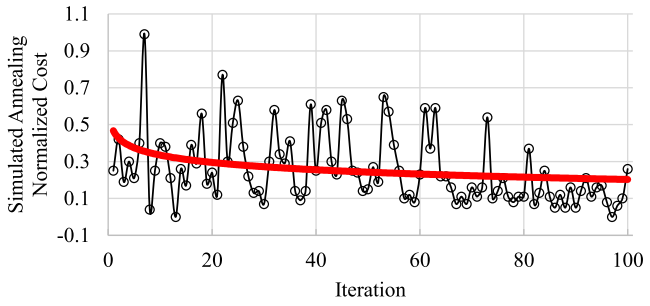


Fig. 9. Variation of the normalized cost values during the design space exploration. The red line is a logarithmic trend-line.

solution quality and to reduce total execution time. Such fine-tuning can be done for example by implementing different types of *moves* inside the simulated annealing algorithm, or different temperature cooling strategies, etc.

6. Discussion

Noteworthy, the design space exploration (DSE) is very effective in finding excellent desired solution points for relatively a small number of 100 iterations allowed inside the simulated annealing algorithm. The best solution found is satisfactorily close to the target one. Furthermore, we note that this type of DSE formulation is very parallelization friendly to the multithreading parallelization approach that we employed. Our simulation provided a speed up of 7.56x on the 8 core processor.

The entire design solution space exploration framework presented in this paper is implemented in C++. In addition, it is implemented to be extendable in the following sense. On one hand, the MEEP simulator is a plug-and-play simulator. It can be replaced by any other simulator if that is desired. Thus, if one needs to use a different FDTD simulator with features that MEEP may not have, replacing the MEEP core simulator with another simulator is easily done. On the other hand, the type of design that is investigated can also be changed easily by changing the source code of a dedicated class inside the source code. We hope that this versatile software framework will foster further research into electromagnetic systems, which increasingly depend on complex simulations.

7. Conclusion

We presented a software tool for the automated design optimization of microstrip patch antennas. The optimization consists of a parallel implementation of simulated annealing based design space exploration. The solution space is defined by four different design parameters that can take discrete values from prescribed permissible ranges. During each iteration of the simulated annealing loop, we employ the popular MEEP simulation tool to evaluate explored design solutions. To speed up the design space exploration, the software framework is developed to run multiple MEEP simulations concurrently. This is achieved using multithreading that uses a number of worker threads equal to the number of cores of the computer that is utilized. Simulation results demonstrate the effectiveness of the proposed tool in finding optimal designs in significantly shorter times, i.e., 7.56x faster when executed on an 8-core processor compared to the execution on only one core. The tool will be released to the public domain.

Conflict of interest

None declared.

References

Ababei, C. (2009). Parallel placement for FPGAs revisited. In *ACM/IEEE int. symposium on field programmable gate arrays (FPGA)*.

Andrews, G. R. (1999). *Foundations of multithreaded, parallel, and distributed programming*. Addison Wesley.

Balanis, Constantine A. (1997). *Antenna theory: analysis and design* (2nd ed.). New York, NY: John Wiley & Sons.

Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications, 45*, 41–51.

Garg, R., Bhartia, P., Bahl, I., & Ittipiboon, A. (2001). *Microstrip antenna design handbook* (1st ed.). Norwood, MA: Artech House.

Gill, S. (1958). Parallel programming. *The Computer Journal, 1*(1), 2–10.

Gondarenko, A., & Lipson, M. (2008). Low modal volume dipole-like dielectric slab resonator. *Optics Express, 16*(22), 17689–17694.

Hansen, P., Zheng, Y., Perederey, E., & Hesselink, L. (2011). Nanophotonic device optimization with adjoint FDTD, CLEO: 2011 – Laser applications to photonic applications, OSA technical digest (CD). Optical Society of America.

Jayasinghe, J. M. J. W., & Uduwawala, D. N. (2015). A novel multiband miniature planar inverted F antenna design for bluetooth and WLAN applications. *International Journal of Antennas and Propagation, 2015*(970152), 1–6.

Kirkpatrick, S., Gelatt, C. D., Jr, & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

MEEP (2017) – A free finite-difference time-domain (FDTD) simulation software package, MIT. <http://ab-initio.mit.edu/wiki/index.php/Meep>.

Meng, F., & Sharma, S. (2016). A single feed dual-band (2.4 GHz/5 GHz) miniaturized patch antenna for wireless local area network (WLAN) communications. *Journal of Electromagnetic Waves and Applications, 30*(18), 2390–2401.

Oskooi, A. F., Roundy, D., Ibanescu, M., Bermel, P., Joannopoulos, J. D., & Johnson, S. G. (2010). Meep: A flexible free-software package for electromagnetic simulations by the FDTD method. *Computer Physics Communications, 181*(3), 687–702.

Owens, J. (2007). GPU architecture overview. In *ACM SIGGRAPH 2007 courses*. New York.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Philips, J. C. (2008). GPU computing. *Proceedings of the IEEE, 96*(5), 879–889.

Stutzman, Warren L., & Thiele, Gary A. (1998). *Antenna theory and design* (2nd ed.). New York, NY: John Wiley & Sons.

Wahl, P., Ly-Gagnon, D.-S., Debaes, C., Miller, D. A. B., & Thienpont, H. (2013). B-CALM: An open-source multi-GPU-based 3D-FDTD with multi-pole dispersion for plasmonics. *Progress in Electromagnetics Research, 138*, 467–478.

Wilson, G. V. (1994). The history of the development of parallel computing. <http://ei.cs.vt.edu/history/Parallel.html>.

Yang, G. (1991). Paraspice: A parallel circuit simulator for shared-memory multiprocessors. In *IEEE/ACM design automation conference (DAC)*.