Run time optimizations for the Split Step Beam Propagation Method

Matthias Fertig
Konstanz University of Applied Sciences, Alfred-Wachtel Strasse 8, 78469 Konstanz, Germany
matthias.fertig@htwg-konstanz.de

Abstract: This paper proposes three extensions of the standard Split Step Beam Propagation Method (SS-BPM) to optimize run times. The optimizations apply to homogeneous symmetric or inhomogeneous symmetric systems and do not rely on approximations or increase memory consumption. Such run time optimizations are important to shorten product development cycles where a high number of parameter variations need to be investigated to find optimal solutions. Today, massively parallel implementations are first choice to reduce run times while single-thread algorithms sometimes show potential to outperform such parallel implementations. This paper focuses on the optimization of the single-thread implementation of the SS-BPM. The proposed optimizations achieve their run time reductions from symmetries similar to the optimizations applied to the algorithm of the Fourier Transformation. The paper shows that a single-thread implementation of the SS-BPM can outperform a massively parallel implementation in several cases that have practical relevance. To show the general potential of the optimizations and not limit the applicability to selected examples, run times and correctness are investigated for simulations of random homogeneous symmetric and random inhomogeneous symmetric systems traversed by Gaussian beams. The benchmarking is performed for a variation of the grid size and performance is compared to the standard algorithm and a massively parallel implementation. All applications providing such types of symmetries will benefit from the optimizations. The results show that the single-thread implementation can outperform the massively parallel implementation for inhomogeneous symmetric systems. The proposed optimizations achieve a run time reduction of up to 73 percent, are on-demand and apply on-the-fly, show no loss of accuracy, execute in-place and do not consume additional memory.

1. Introduction

Run time and memory consumption are two key indicators for the performance of algorithms. The performance of algorithms is important for the development of photonic devices because thousands of simulations on variations in device parameters are required to find the optimal solution. Such simulations usually contain homogeneities and symmetries that can be used to optimize the simulation performance. Such symmetries typically exist for waveguides, tapers, splitters, couplers, gratings, lenses or free-space propagation between lenses. It is therefore highly desirable to reduce run times and reduce or limit memory consumption during simulations wherever possible. As hardware speeds increase through parallelization and hardware costs reduce due to mass production and general availability, the optimization of single-thread performance tends to be neglected. But even well known and deeply investigated algorithms, such as the SS-BPM, often contain the potential to improve single-thread performance. This paper focuses on the single-thread performance of the SS-BPM and provides three types of optimizations to improve run times up to levels that are competitive with implementations on a massively parallel system such as a GPU.

The contribution of this paper is an optimized single-thread implementation of the SS-BPM for homogeneous symmetric and inhomogeneous symmetric systems. It shows why the massively parallel implementation cannot outperform the single-thread implementation for homogeneous symmetric and inhomogeneous symmetric systems. The optimizations achieve run time reductions between 40 and 73 percent so that the simulation overhead for device optimization reduces by a factor of 1.6 to 4, which is to say that up to four times as many simulations can be performed at the same time without the cost for a massively parallel hardware. The optimizations do not rely on approximations of the standard algorithm, are optional, apply on the fly and do not consume additional memory. All characteristics, the accuracy and the memory usage of the original algorithm are fully retained. The optimizations are introduced for two-dimensional systems and uni-directional propagation to show the basic concepts but the principles are easily applicable to three-dimensional systems and bidirectional propagation. The paper is fur-

thermore a comfortable starting point for engineers who are new in the field of optical Fourier simulators because it provides the basic theory and implementation examples to start quickly and adapt the code to individual needs.

In the original BPM scheme by Feit and Fleck [1], the propagation operator was split into two operators, a homogenous medium propagation in the averaged index and a thin element transmission through the index variation, i.e. the Split Step Beam Propagation Method (SS-BPM). With the application of propagation methods to more general optical components, such as gradient index media, spheric and aspheric lenses, and gratings, there is much interest in removing some of the restrictions of the SS-BPM. A non-paraxial finite difference SS-BPM for non-paraxial propagation using a symmetrized splitting of the propagation operator in a reformulation of the wave equation as a matrix ordinary differential equation has been introduced by Sharma and Agrawal [2]. Another higher-order propagation operator has been used by Hadley [3] to remove the paraxial limitation. Ma and van Keuren presented a three-dimensional wide-angle BPM in [4] for optical waveguide structures. A semivectorial wide angle BPM was presented by Lee and Vogoes in [5]. The BPM was extended to very wide angles in [6]. Several vector extensions of the BPM have been presented by Yamauchi et al. [7], Liu and Li [8], and Wanguemert-Perez and Molina-Fernandez [9]. Yamauchi et al. introduced a modified semivectorial beam propagation method retaining the longitudinal field component, Liu and Li analyzed the polarization modes of rib waveguides with a semivectorial BPM, and Wanguemert-Perez and Molina-Fernandez presented a fully-vectorial three-dimensional extension of the BPM. A third class of extensions is based on the finite element (FE) approach as published by Tsui et al. [10], Stern [11], Pinheiro et al. [12] and Obayya and Rahman [13]. The FE-based extensions of the BPM are optimized for scalar, semivectorial, and full-vectorial fields as well as numerically efficient methods in the order of citation. The fourth and last class of extensions in this brief overview of BPM-based methods is the multigrid approach that is used to reduce computational effort at regions in a system that allow a reduced degree of accuracy. Sewell et al. introduced a multigrid method for electromagnetic computation in [14]. Except for the Padé approximation in [3] none of the BPM-based approaches overcome the limitation that originates from the separation of the operators. Other Fourier propagation methods such as the Wave Propagation Method [15] or the Vector Wave Propagation Method [16] perform a propagation of waves without the aforementioned limitations.

The paper is organized in five sections. Section 2 explains the original BPM algorithm and introduces the nomenclature used in this paper. The run time optimizations are introduced in section 3. There, the standard algorithm is extended by four individual steps: 'GridAnalysis' (3.1), 'HomogOpt' (3.2), 'FreqOpt' (3.3) and 'SpatOpt' (3.4). The optimizations are incrementally benchmarked in section 4. Section 4.3 investigates the improvements for homogeneous symmetric and section 4.4 for inhomogeneous symmetric systems. All benchmarking is performed with random Gaussian beams and random homogeneous symmetric and random inhomogeneous symmetric systems to show the general applicability of the approach. The benchmarking performs on variations of the grid size and provides run times as well as cycle counts to allow performance comparisons with systems running at different clock speeds. Sections 4.5 and 4.6 show the thread timing breakdown for the single-thread implementation on a CPU and the kernel timing breakdown for the massively parallel implementation on a GPU. The results are summarized and conclusions are drawn in section 5.

2. Split-Step Beam Propagation Method (SS-BPM)

A discretized two-dimensional system is defined by a spatial refractive index distribution $n(i_x, i_z)$ where $x = i_x \cdot \triangle x$ and $z = i_z \cdot \triangle z$ is the position defined by the index $0 \leq i_x \leq n_x - 1$ and $0 \leq i_z \leq n_z - 1$. The system has an aperture $X = n_x \cdot \triangle x$ and a length $Z = n_z \cdot \triangle x$. A layer in the x-axis of such a system is called homogeneous if the refractive index distribution $n(i_x) = const$ and symmetric if $n(n_x - 1 - i_x) = n(i_x)$. The entire system is called homogeneous if $n(i_x, i_z) = const$ for all i_x and i_z. The BPM algorithm computes the electromagnetic field distribution of a scalar wave along an axis of propagation, i.e. the z-axis in this paper. The x-axis is the lateral axis and, together with the z-axis, spans an orthogonal cartesian coordinate system. The system is split into n_z layers parallel to the x-axis. Each layer parallel to the x-axis is defined by n_x samples. The $n_x \cdot n_z$ samples span the simulation grid of size $(n_x \cdot \triangle x) \cdot (n_z \cdot \triangle z) = X \cdot Z$. The incident field is stored in layer 0 and the BPM iterates the z-axis for $n_z - 1$ steps. It computes the field distribution in layer i_{z+1} from the field distribution in layer i_z, where i_z iterates between $0 \leq i_z < n_z - 2$ and layers are indexed between $0 \leq i_z < n_z - 1$. The electric field in layer i_z is E_{iz} and the refractive index distribution is n_{iz}.

2.1. *Plane wave decomposition ('FFT')*

The BPM is a Fourier method as it derives the spatial field distribution at E_{iz+1} from the plane wave spectrum e_{iz} of layer E_{iz}. The spectrum is obtained from a Fourier Transformation $\mathscr{F}\{\ \}$

$$e_{iz}(k_x) = \mathscr{F}\{E_{iz}(x)\} \tag{1}$$

with $x = i_x \cdot \triangle x$ and $k_x(i_x) = i_x/(n_x \cdot \triangle x) = i_x/x$ where $-n_x/2 \leq i_x \leq n_x/2-1$. $\triangle x$ is obtained from X/n_x where X is the aperture size.

2.2. *Amplitude transformation and average phase shift ('BPM Step-1')*

While propagating a distance $\triangle z = z/n_z$ through layer $iz+1$, each plane wave component $e_{iz+1}(k_x)$ experiences a phase shift $\phi_{a,iz+1}(k_x) = k_{az,iz+1}(k_x) \cdot \triangle z$, where $k^2_{az,iz+1} = k^2_{a,iz+1} - k^2_x$ is the z-component of the propagation vector $\mathbf{k} = (k_x\ k_z)^T$ and a function of k_x. $k_{a,iz+1} = n_{a,iz+1} \cdot k_0$ is the average wave number in layer i_z+1 obtained from the average refractive index in layer i_z+1

$$n_{a,iz+1} = \frac{1}{n_x} \cdot \sum_{ix=0}^{nx} n_{iz+1}(i_x) \tag{2}$$

k_0 is $2\pi/\lambda_0$ and λ_0 is the wavelength of the incident wave at $i_z = 0$ is $E(x,0) = 1 \cdot exp\{j\mathbf{kr}\} = exp\{j(k_x \cdot x + k_z \cdot z)\}$. Here, a unit amplitude is assumed for simplicity but without prejudice to the generality. In the absence of changes of the refractive index and for purely real $n(i_x)$ the absolute of the amplitude remains unchanged.

2.2.1. Amplitude transformation

A boundary is defined by a change in the average refractive index in the z-axis, i.e. $n_{a,iz} \neq n_{a,iz+1}$. In this case the z-components of the propagation vector experience a change, i.e. $k_{z,iz} \neq k_{z,iz+1}$, and the amplitudes $e_{iz}(k_x)$ transform to $f_{iz+1}(k_x)$ according to the Fresnel coefficients of amplitude

$$t_{te} = \frac{2 \cdot k_{z,iz}}{k_{z,iz} + k_{z,iz+1}} \tag{3}$$

$$f_{iz+1}(k_x) = t_{te} \cdot e_{iz}(k_x) \tag{4}$$

for TE-polarized waves. The analysis in this paper focuses on TE-polarized waves but this does not affect the proposed optimizations. TM Fresnel coefficients are applied to simulate TM-modes. The coefficient simplifies to $t_{te} = 1$ if $n_{a,iz} = n_{a,iz+1}$.

2.2.2. Average phase shift

The transmitted amplitude of each plane wave component $f_{iz+1}(k_x)$ propagates a distance $\triangle z$ through layer i_z+1 and experiences a phase shift $\phi_{a,iz+1}(k_x) = k_{az,iz+1}(k_x) \cdot \triangle z$. The plane wave component is then

$$e_{iz+1}(k_x) = f_{iz+1}(k_x) \cdot e^{j \cdot \phi_{z+1}(k_x)} \tag{5}$$

'BPM Step-1' is finished after n_x iterations when amplitude transformations and phase shifts for all plane wave components e_{iz+1} are calculated.

2.2.3. 'NoOpt' (standard) implementation of 'BPM Step-1' [1]

```
for(int ix=0; ix<nx; ix++) {
  kx_sq = pow(KX[ix].real(),2);
  nik0_sq = pow(ni_ave*k0,2);
  ntk0_sq = pow(nt_ave*k0,2);
  if((kx_sq>=nik0_sq)||(kx_sq>=ntk0_sq)) continue;
  kzt=one*sqrt(ntk0_sq-kx_sq);
  if(ni_ave!=nt_ave){
    kzi=one*sqrt(nik0_sq-kx_sq);
    // hint: use TM Fresnel amplitude coefficient for TM modes
    e[ix] = 2*kzi*kzt/(kzi+kzt) * e[ix] * exp(ione*kzt*dz);
  } else {
    e[ix] = e[ix] * exp(ione*kzt*dz);
} } }
```

Listing 1. BPM reference code. Amplitude transformation and phase shift ('BPM Step-1').

2.3. *Inverse Fourier Transformation ('IFFT')*

Using the average refractive index $n_{a,iz+1}$ and not the exact refractive index $n_{iz+1}(x)$ in the phase shift $\phi_{iz+1}(k_x)$ introduces a phase error $\phi_e(x)$ that needs to be corrected in the spatial domain. This imperfect propagated spatial field distribution F_{iz+1} is obtained from an inverse Fourier Transformation $\mathscr{F}^{-1}\{\ \}$

$$F_{iz+1}(x) = \mathscr{F}^{-1}\{e_{iz+1}(k_x)\} \tag{6}$$

2.4. *Spatial phase correction ('BPM-Step2')*

The local phase error $\phi_e(x) = (n_{iz+1}(x) - n_{a,iz+1}) \cdot k_0 \cdot \triangle z$ depends on the spatial refractive index distribution $n_{iz+1}(x)$ and finally the field distribution in layer $iz+1$ is

$$E_{iz+1}(x) = F_{iz+1}(x) \cdot e^{j \cdot \phi_e(x)} \tag{7}$$

2.5. *'NoOpt' (standard) implementation of 'BPM Step-2' [1]*

```
backward->fftNormalized(e,E+((iz+1)*nx);
for(int ix=0; ix<nx; ix++)
   E[(iz+1)*nx+ix] *= exp(ione*(N[(iz+1)*nx+ix]-nt_ave)*k0*dz);
```

Listing 2. BPM reference code. Phase correction (BPM Step-2).

The sequence of steps is 'GridAnalysis', 'FFT', 'BPM Step-1', 'IFFT' and 'BPM Step-2'. The algorithm performs n_z iterations of this sequence until all layers are calculated. It is important to note that the phase shift calculates with the average refractive index $n_{a,iz+1}$ and the phase correction considers perpendicular propagation only. This is called the separation of the propagation operator into a diffraction operator that is applied on the spatial frequency spectrum in equation 5 and a phase correction operator that is applied on the spatial field distribution in equation 7.

3. Run time optimizations

The run time optimizations in this paper utilize symmetries in the spatial frequency vector ('FreqOpt') of the forward propagating or backward propagating wave, symmetries in the spatial index distribution ('SpatOpt') and homogeneities ('HomogOpt'). In this context homogeneous layers are interpreted as a special form of symmetry in the spatial refractive index distribution, where the index is constant.

3.1. *Symmetry and homogeneity analysis ('GridAnalysis')*

The optimizations require a layer analysis step, called 'GridAnalysis' in this paper, to investigate for symmetric and homogeneous layers. Listing 3 shows an efficient way to perform this grid analysis in a half-range loop. This half-range loop is also used to optimize the calculation of the average refractive index $n_{a,iz+1}$.

```
is_homog=true;
is_symmetric=true;
tmp=N[(iz+1)*nx].real();
ni_ave=nt_ave;
nt_ave=0;        // IMPORTANT: initialize nt_ave zero
int nhalf=int(ceil(double(nx/2)));
for(int ix=0; ix<nhalf; ix++){
  if(N[(iz+1)*nx+ix].real()!=N[(iz+1)*nx+(nx-1-ix)].real()){
    is_symmetric = false;
    is_homog = false;
  }
  if(tmp!=N[(iz+1)*nx+ix].real()) is_homog = false;
  if(0==ix) ni_ave=ni_ave+N[iz*nx+ix].real()+N[iz*nx+(nx-1-ix)].real();
  nt_ave=nt_ave+N[(iz+1)*nx+ix].real()+N[(iz+1)*nx+(nx-1-ix)].real();
}
if(0==ix) ni_ave/=nx;
nt_ave=is_homog ? tmp : nt_ave/nx;
```

Listing 3. 'GridAnalysis' average index implementation w/ homogeneity and symmetry indication.

3.2. *Optimization from homogeneous layers ('HomogOpt')*

A special case of spatial symmetry is homogeneity where the spatial refractive index distribution is not only symmetric but constant. In case of homogeneous layers, the phase correction $\phi_e = (n_{iz+1}(x) - n_{a,iz+1}) \cdot k_0 \cdot \triangle z$ becomes zero because $n_{iz+1}(x)$ is equal to $n_{a,iz+1}$. 'BPM Step-2' becomes a multiplication by one and can therefore be skipped. The expected run time improvement for 'HomogOpt' equals the time spent for 'BPM Step-2' minus the time to analyse a layer for homogeneity.

3.3. *Optimization from symmetries in the spatial frequency vector ('FreqOpt')*

The spatial frequency vector $\mathbf{k}$ is symmetric around the center frequency $k_x = 0$, i.e. perpendicular propagation as shown in table 1. The two-dimensional propagation vector $\mathbf{k} = (k_x, k_z)^T$ derives from its transveral component k_x

and the average wave number $k_{a,iz+1}$, with

$$k_x(i) = \frac{i}{n_x \cdot \triangle x} = \frac{i}{x} \quad , \quad -\frac{n_x}{2} \le i \le \frac{n_x}{2} - 1 \tag{8}$$

and where i is mapped to i_x as shown in table 1. The table shows the symmetry in k_x for $nx = 8$ with $k_x(-i_x) = k_x(i_x)$ for $i_x \in \{1, 2, \ldots, n_x/2 - 1\}$. The symmetry does not occur for $i_x = 0$ and $i_x = -n_x/2$ because the zero frequency occurs only once and is counted a positive frequency so that the maximum positive spatial frequency is less than the minimum spatial frequency by one spatial frequency increment $1/X$.

ix	0	1	2	3	4	5	6	7
i	0	1	2	3	-4	-3	-2	-1
$k_x(i)$	0	1/X	2/X	3/X	-4/X	-3/X	-2/X	-1/X
	0	1/X	2/X	3/X	-4/X	$-k_x[3]$	$-k_x[2]$	$-k_x[1]$
	0	1/X	...	$\left(\frac{nx}{2} - 1\right) \cdot \frac{1}{X}$	$-\frac{nx}{2} \cdot \frac{1}{X}$	...	$-k_x[2]$	$-k_x[1]$

Table 1. Symmetry in the spatial frequency vector for $n_x = 8$.

This results in a symmetric z-component of the propagation vector k_z and finally a symmetric phase ϕ_z. Equation 5 transforms to

$$\phi_{z+1}(\pm i_x/X) = \sqrt{n_{a,iz+1}^2 k_0^2 - (\pm k_x)^2} \cdot \triangle z \quad when \quad i_x \neq 0 \ \ and \ \ i_x \neq -n_x/2 \tag{9}$$

$$\phi_{z+1}(0) = n_{a,iz+1} \cdot k_0 \cdot \triangle z \quad when \quad i_x = 0 \tag{10}$$

$$\phi_{z+1}(-0.5/\triangle x) = \sqrt{(n_{a,iz+1} \cdot k_0)^2 - 0.25/\triangle x^2} \quad when \quad i_x = -n_x/2 \tag{11}$$

where the number of iterations for i_x reduce from n_x to $n_x/2 + 1$. Thereby, 'BPM Step-1' in equation 5 reduces to

$$e_{iz+1}(\pm k_x) = f_{iz+1}(\pm k_x) \cdot e^{j \cdot \phi_{z+1}(\pm k_x)} \quad when \quad k_x \neq 0 \ \ and \ \ k_x \neq -n_x/2/X \tag{12}$$

The expected run time improvement for 'FreqOpt' is the time saved by iterating only the half range. Listing 4 shows the implementation of 'FreqOpt' in a half-range loop and two assignments that share the intermediate result by using spatial frequency symmetries.

```
int nhalf=int(ceil(double(nx/2)));
cpx ctmp;
for(int ix=0; ix<=nhalf; ix++) {
 ctmp = 2*kzi*kzt/(kzi+kzt) * exp(ione*kzt*dz); // intermediate result
 if(ix>0 && ix<nhalf) {
    e[ix]    *= ctmp;
    e[nx-ix] *= ctmp;
  } else {
    e[ix]    *= ctmp;
} }
```

Listing 4. Implementation 'FreqOpt' for symmetries in the spatial frequency vector (BPM Step-1).

3.4. Optimization from symmetries in the spatial refractive index distribution ('SpatOpt')

Symmetries in the spatial index distribution n_{iz+1} do not always occur and need to be detected. This is again performed in a half-range loop as shown in listing 3. In case of symmetry in the spatial index distribution, equation 7 becomes

$$\phi_{e,iz+1}(\pm x) = (n_{iz+1}(\pm x) - n_{a,iz+1}) \cdot k_0 \cdot \triangle z \tag{13}$$

and the number of iterations in listing 2 reduces from n_x to $n_x/2$. The expected run time improvement for 'SpatOpt' is the time saved by iterating only the half range. Listing 5 shows the implementation of 'SpatOpt'. In case of symmetry in the spatial index distribution the code in listing 2 is executed in a half-range loop while otherwise the conventional loop applies.

```
if(is_symmetric){
   for(int ix=0; ix<nhalf; ix++) {
    ctmp = exp(ione*(N[(k+1)*nx+ix]-nt_ave)*k0*dz);
    E[(k+1)*nx+ix]          *= ctmp;
```

```
4     E[(k+1)*nx+(nx-1-ix)] *= ctmp;
5 }   } else
6     for(int ix=0; ix<nx; ix++){
7     E[(k+1)*nx+ix] *= exp(ione*(N[(k+1)*nx+ix]-nt_ave)*k0*dz);
```

Listing 5. Implementation 'SpatOpt' for spatial refractive index symmetries (BPM Step-2).

4. Benchmarking

The optimizations for 'GridAnalysis', 'HomogOpt', 'FreqOpt' and 'SpatOpt' are investgated with two types of benchmarks, a random forward propagating Gaussian beam of unit amplitude propagation through random homogeneous symmetric and random inhomogeneous symmetric systems. Each benchmark iterates the grid size five times from 32x128 up to 2048x8192 samples. Every iteration performs 50 runs on a CPU and a GPU for a system size $4\lambda \cdot 4\lambda \cdot 4\lambda$, where λ is a random wavelength and waist of the Gaussian beam in a range between $500nm$ and $1500nm$. Thereby a number of 500 runs per benchmark and a total number of 1000 runs is performed to analyze the run time optimizations. In a second series of runs the CPU-thread and GPU-kernel time breakdown are analyzed to show the improvements of the individual steps of optimization.

4.1. Implementation and run times

The optimizations extend the conventional code and do not apply any approximations. They apply on-the-fly in case of homogeneous symmetric or inhomogeneous symmetric layers. The benchmarks are performed on a CPU with $2.1GHz$ clock frequency and a GPU with $1.13GHz$ core frequency and 1024 compute cores. The CPU code is single-threaded and does not benefit from any parallel code execution. The CPU-thread applies 'GridAnalysis' to indicate layer homogeneity and symmetry. The GPU-kernel applies a reduced version of 'GridAnalysis' to indicate homogeneous layers and to apply 'HomogOpt'. The GPU-kernel does not apply 'FreqOpt' or 'SpatOpt' because the relevant code runs in parallel on the GPU cores anyway. Syncronization points of the GPU-kernel are 'FFT' and 'IFFT' while 'BPM Step-1' and 'BPM Step-2' run parallel. The device frequency ratio is $1.13/2.1 = 0.54$. All GPU run times exclude latencies for copying data from host-to-device and from device-to-host. The copy latencies are between $2 \cdot 0.08ms$ for a 32x128 grid and $2 \cdot 156ms$ for a 2048x8192 grid and depend on the performance of the hardware interface. This copy overhead is excluded from run time analysis to show the performance of the parallel SS-BPM algorithm.

4.2. Accuracy and memory consumption

All proposed optimizations have been benchmarked against the standard program code, i.e. 'NoOpt'. The relative error in the electric field distribution is below 0.01 percent per sample for all runs in the benchmarks, i.e. $-40dB$. In order to average-out occasional run time glitches from task switches or other interruptions of the operating system, all run times are average values obtained from 50 runs per iteration. The optimizations perform in-place and require no additional memory. Individual simulations utilize memory for the complex-value electric field and real-value refractive index, i.e. $3 \cdot (n_x \cdot n_z) \cdot 4$ Bytes for single precision and $3 \cdot (n_x \cdot n_z) \cdot 8$ Bytes for double precision. If material absorption is supposed to be simulated, the refractive index has to be complex-valued but to reduce memory consumption for storing the system the refractive index is real-valued in this case. This adds up to $384MB$ memory. With this setting the maximum grid size on the GPU is $2048x8192$ samples due to available on-board memory.

4.3. Homogeneous symmetric systems benchmarking

Figure 1 shows the run times and percentual run time reductions over grid size for a benchmarking of random homogeneous symmetric systems traversed by Gaussian beams on the CPU in comparison to the GPU. Solid lines indicate run times and are read against the left y-axis while dashed lines indicate percentual run time improvements and are read against the right y-axis. X-axis and y-axis are linear scaled. Results are shown for the reference code without any optimization ('NoOpt'), the optimization for homogeneous systems ('HomogOpt'), the combined optimization for homogeneous systems and symmetry in the spatial frequency vector ('HomogOpt+FreqOpt') and for the implementation on the GPU. The parallel implementation on the GPU uses 'HomogOpt' as well.

The 'HomogOpt+FreqOpt' optimization achieves a maximum run time reduction of up to 73 percent as shown in figure 1 when compared to 'NoOpt'. 'HomogOpt' contributes a maximum reduction of 68 percent and 'FreqOpt' between five and six percent for the shown grid sizes and up to 18 percent for a 64x256 grid (not shown here). The run time reduction over all grid sizes is between 58 and 73 percent. Improvements rise quickly and start to saturate early at 73 percent as shown in figure 1. Figure 1 shows that the 'GPU+HomogOpt' outperforms the unoptimized

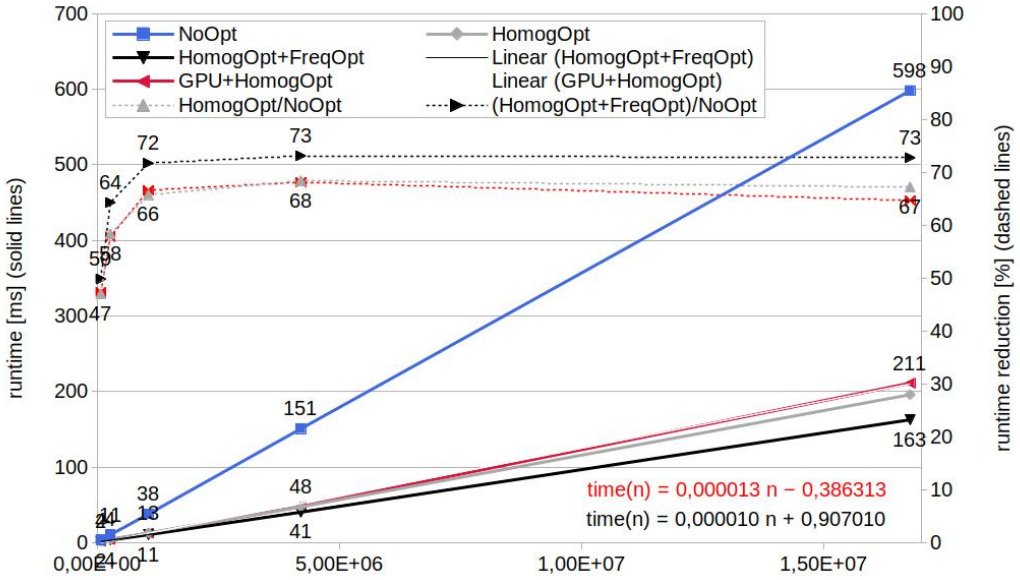

Fig. 1. Run time and percentual reduction over grid size for Gaussian beams traversing random homogeneous symmetric systems.

algorithm 'NoOpt' in run time for all grid sizes and that 'HomogOpt+FreqOpt' achieves even faster run times than 'GPU+HomogOpt' for all grid sizes.

Table 2 shows the number of compute cycles from random homogeous symmetric systems benchmarking. Compute cycles are obtained from run time multiplied by clock frequency. The table shows that the number of compute cycles for 'GPU+HomogOpt' is 30 to 43 percent less than on the CPU for 'HomogOpt+FreqOpt' for all grid sizes. When compared to 'NoOpt', the number of compute cycles on the GPU is smaller by up to 83 percent than on the CPU as depicted in figure 3. Since the GPU runs at lower frequency, the single-thread algorithm on the CPU is faster as long as the CPU over GPU compute cycle ratio does not exceed the frequency ratio of 0.54. The linear extrapolation of numbers in figure 1 for 'GPU+HomogOpt' (red) and 'HomogOpt+FreqOpt' (black) shows that the single-thread implementation on the CPU would outperform the massively parallel implementation on the GPU in run times for larger grids.

Despite the GPU over CPU clock frequency ratio of 0.54, the single-thread implementation of 'NoOpt' has significantly higher run times on the CPU when compared to the parallel implementation of 'GPU+HomogOpt' on the GPU while at the same time 'HomogOpt+FreqOpt' on a single compute core outperforms the 'GPU+HomogOpt' on up to 1024 compute cores by five to six percent (fig. 1).

Table 2. Compute cycle numbers in mega-cycles for Gaussian beams traversing random homogeneous symmetric systems.

n_x	n_y	NoOpt[1] [MCyc]	HomogOpt[2] [MCyc]	Delta vs. [1] [%]	FreqOpt +SpatOpt[3] [MCyc]	Delta vs. [1] [%]	GPU[4] +HomogOpt [MCyc]	Delta vs. [3] [%]
128	512	8.36	4.42	-47	4.19	-50	2.36	-43
256	1024	22.32	9.29	-58	7.97	-64	5.05	-36
512	2048	80.10	27.44	-66	22.64	-72	13.30	-36
1024	4096	324.20	100.00	-69	85.10	-73	54.20	-36
2048	8192	1,332.00	411.60	-67	342.00	-74	238.10	-30

4.4. Inomogeneous symmetric systems benchmarking

Figure 2 shows the run times and percentual run time reductions from a benchmarking of random inhomogeneous symmetric systems traversed by Gaussian beams on the CPU in comparison to the GPU. The figure has the same structure as figure 1.

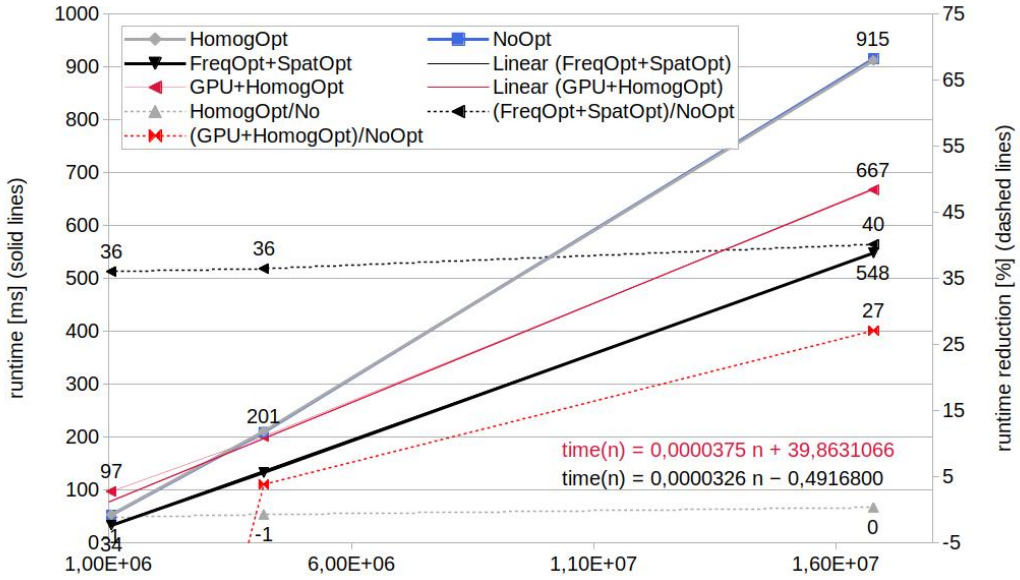

Fig. 2. Run times and percentual reductions over grid size for inhomogeneous symmetric systems.

In case of inhomogeneous symmetric layers the optimization 'HomogOpt' is useless because 'BPM Step-2' is required to perform the phase correction. The lines for 'NoOpt' and 'HomogOpt' in figure 2 therefore coincide and 'HomogOpt/NoOpt' shows a degradation of up to one percent. This degradation is caused by the overhead in 'GridAnalysis'. 'FreqOpt+SpatOpt' achieves a total run time reduction of 36 to 40 percent while 'GPU+HomogOpt' achieves a 27 percent reduction only. The run time improvement with 'FreqOpt+SpatOpt' scales well with the grid size and shows quite a constant level of improvement (black dashed line in fig 2) while 'GPU+HomogOpt' shows a significant drop in performance for $n_x < 1024$ (red dashed line in fig 2). The run times of 'FreqOpt+SpatOpt' and 'GPU+HomogOpt' show an increasing gap so that the advantage of 'FreqOpt+SpatOpt' tends to improve for larger grids. The linear extrapolations of run times for 'GPU+HomogOpt' (red formula in fig 2) and 'FreqOpt+SpatOpt' (black formula in fig 2) confirm this trend. The gradient for 'GPU+HomogOpt' is less than the gradient for 'FreqOpt+SpatOpt'.

Table 3. Compute cycle numbers for Gaussian beams traversing random inhomogeneous symmetric systems in mega cycles.

n_x	n_y	NoOpt[1] [MCyc]	HomogOpt[2] [MCyc]	Delta vs. [1] [%]	FreqOpt +SpatOpt[3] [MCyc]	Delta vs. [1] [%]	GPU[4] +HomogOpt [MCyc]	Delta vs. [3] [%]
128	512	9.15	9.24	+1	6.53	-29	27.04	+313
256	1024	29.39	29.66	+1	19.42	-35	54.34	+180
512	2048	110.60	111.80	+1	70.78	-37	109.40	+55
1024	4096	439.00	442.10	+1	279.00	-37	226.60	-19
2048	8192	1,920.00	1,913.00	< 1	1,150.00	-40	751.70	-35

Table 3 gives the number of compute cycles for inhomogeneous symmetric random benchmarking. The figures show that there are 19 to 35 percent less compute cycles on the GPU than on the CPU for 'FreqOpt+SpatOpt' for $n_x \geq 1024$. For $n_x < 1024$ 'GPU+HomogOpt' performs up to 313 percent more cycles when compared to 'FreqOpt+SpatOpt'. Figure 3 (right) shows that the number of compute cycles on the GPU is up to 60 percent smaller than for 'NoOpt'. The implementation of 'GridOpt' is efficient as it scales well with the grid and constantly reduces the performance by one percent or less as shown in table 3.

Despite the GPU over CPU clock frequency ratio of 0.54, the single-thread implementation of 'NoOpt' shows significantly higher run times on the CPU when compared to the parallel implementation of 'GPU+HomoOpt' on the GPU while at the same time 'FreqOpt+SpatOpt' on a single compute core outperforms 'GPU+HomogOpt' on up to 1024 compute cores by 13 percent (fig. 2).

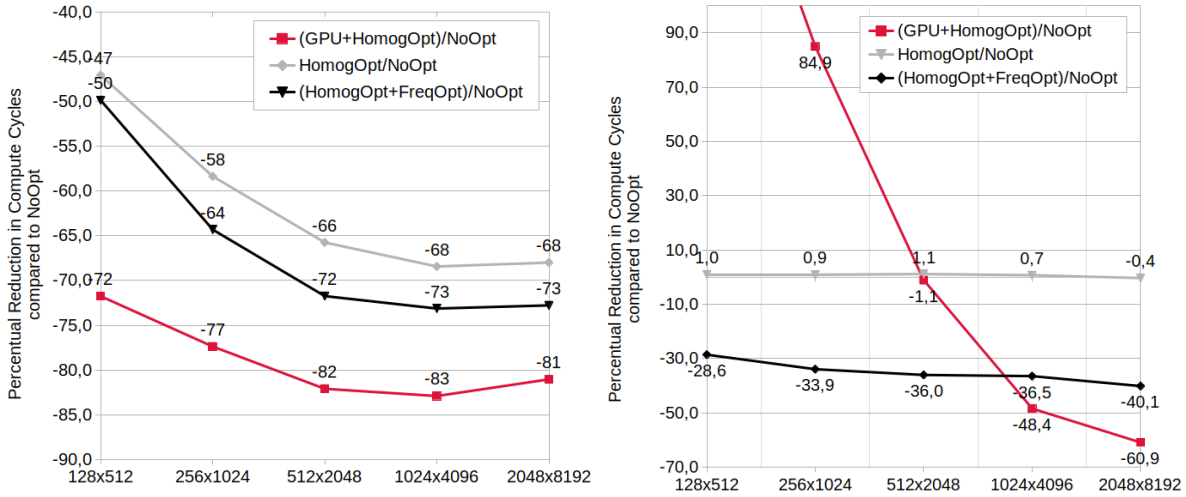

Fig. 3. Percentual reduction in compute cycles compared to 'NoOpt' over grid size for Gaussian beams traversing random homogeneous symmetric (left) and random inhomogeneous symmetric (right) systems benchmarking. The number of compute cores on the CPU is one and on the GPU 1024.

4.5. CPU thread timing breakdown

Figure 4 shows the timing breakdown of the single-thread implementation on a CPU. The latencies for 'Grid-Analysis', 'FFT', 'BPM Step-1', 'IFFT' and 'BPM Step-2' over the level of optimization 'NoOpt', 'HomogOpt' and 'HomogOpt+FreqOpt' for homogeneous symmetric or 'FreqOpt+SpatOpt' for inhomogeneous symmetric systems are shown. The analysis is performed on a 2048x8192 grid size.

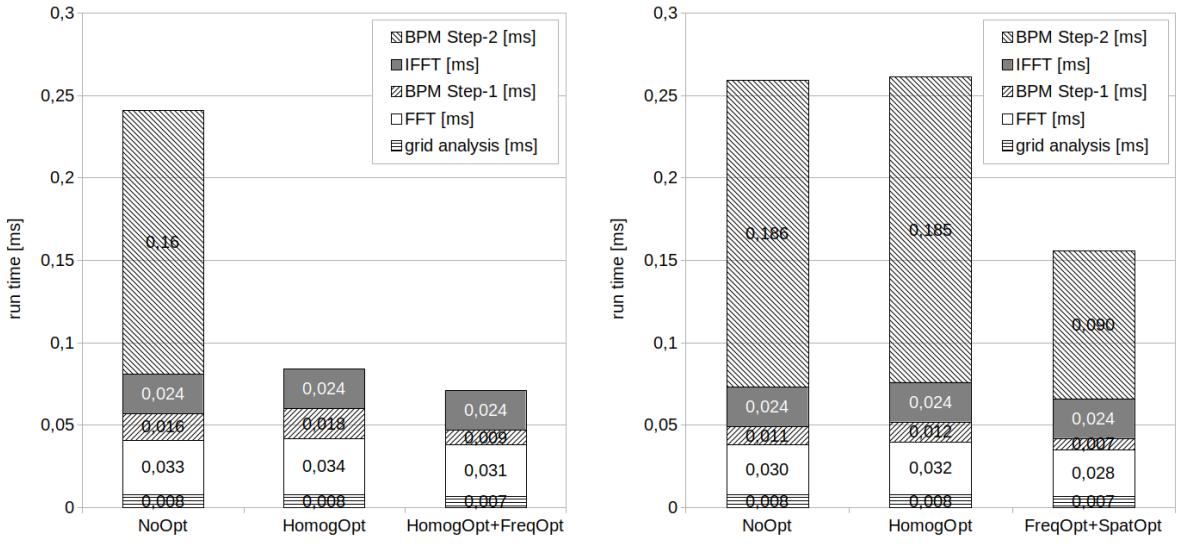

Fig. 4. CPU single-thread timing over level of optimization for Gaussian beams traversing random homogeneous symmetric systems (left) and random inhomogeneous symmetric systems (right) on a 2048x8192 grid.

The results in figure 4 (left) show that the latency for 'BPM Step-2' removes completely for homogeneous symmetric systems, thereby reducing the run time by 65 percent from 0.241*ms* to 0.084*ms*. 'FreqOpt+HomogOpt' reduces the overall run time by another six percent from 0.084*ms* to 0.071*ms* and 'BPM Step-1' by 50 percent from 0.018*ms* to 0.009*ms*. Latencies for 'GridAnalysis', 'FFT' and 'IFFT' show some small variation that is independent of 'HomogOpt' or 'FreqOpt'.

The results in figure 4 (right) show that the overall latency for 'HomogOpt' increases by one percent from

0.259ms to 0.261ms. The increase is caused by additional service code for 'HomogOpt' in 'GridAnalysis' and another if-clause between 'IFFT' and 'BPM Step-2'. With 'FreqOpt+SpatOpt' a total run time improvement of 40 percent from 0.261ms to 0.156ms is achieved. The latency for 'BPM Step-1' improves by 42 percent from 0.012ms to 0.007ms and for 'BPM Step-2' by 51 percent from 0.185ms to 0.09ms.

4.6. GPU kernel timing breakdown

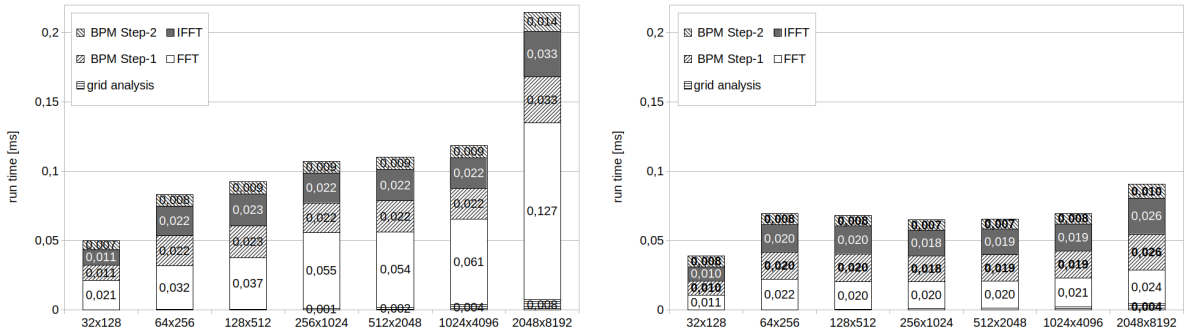

Fig. 5. GPU (1.13GHz, 1024 cores, massively parallel) timing breakdown for the first iteration (left) and the $n_z/2$-th iteration (right) with 'GPU+HomogOpt' and Gaussian beams traversing random symmetric inhomogeneous grids.

Figures 5 and 6 show the timing breakdown for one iteration for inhomogeneous symmetric layers with optimzations 'FreqOpt' and 'SpatOpt' on the GPU and CPU. The figures have the same linear scale on the y-axis and show the latencies in milliseconds for the computation steps 'GridAnalysis', 'FFT', 'BPM Step-1', 'IFFT' and 'BPM Step-2' over the grid size n_xxn_z. Both figures show the timing breakdown for the first iteration of a run on the left side and for the $n_z/2$-th iteration of a run on the right side.

Figure 5 shows the timing breakdown for the GPU kernel. The latencies for the first iteration are significantly higher than for the $n_z/2$-th iteration because the caches and registers of the hardware are not loaded yet. The latency for 'GridAnalysis' is negligible for all grid sizes and the latencies of 'BPM Step-1' and 'BPM Step-2' change with the number of samples as expected. The latencies for 'FFT' reduces significantly between the first and $n_z/2$-th iteration but still consumes a significant amount of computation time for all grid sizes. 'IFFT' is quite constant for the first and $n_z/2$-th iteration, probably because the internal data structures for the Fourier Transformation is set up with the first 'FFT' in the first iteration so that it is already available 'IFFT'. The latencies for 'BPM Step-1' and 'BPM Step-2' reduce by up to 30 percent from the first iteration to the $n_z/2$-th iteration on the GPU.

The timing breakdown on the CPU in figure 6 shows significantly lower latencies per iteration. While the latencies for 'GridAnalysis' are almost identical, the latencies for 'FFT' and 'IFFT' on the GPU are up to $0.54 \cdot 0.127ms/0.033ms \approx 2.1$ times higher for the first iteration and still $0.54 \cdot 0.024ms/0.008ms = 1.6$ times higher for the $n_z/2$-th iteration than on the CPU as shown in figure 6, where $1.13GHz/2.1GHz \approx 0.54$ is frequency ratio between GPU and CPU. The latencies for 'BPM Step-1' are lower on the CPU for all grid sizes when compared to the GPU. The latency for 'BPM Step-2' on the GPU is 5.4 times lower in the first iteration and still 4.5 times lower in the $n_z/2$-th iteration when compared to the CPU.

The latencies on the GPU in figure 5 show a small dependency on the grid size while the latencies on the CPU change significantly with the number of samples. The latencies per iteration on the CPU are lower than on the GPU for all grid sizes under investigation and change significantly from the first to the $n_z/2$-th iteration. The analysis of the last iteration (not shown here) did not show a better performance.

5. Conclusions

The paper demonstrates that the algorithm of the SS-BPM is well suited for single-thread optimizations up to a point where it can compete with a massively parallel implementation.

The optimized single-thread implementation of the SS-BPM shows faster run times than the parallel implementation for homogeneous symmetric systems as shown in section 4.3 and for inhomogeneous symmetric systems as shown in section 4.4.

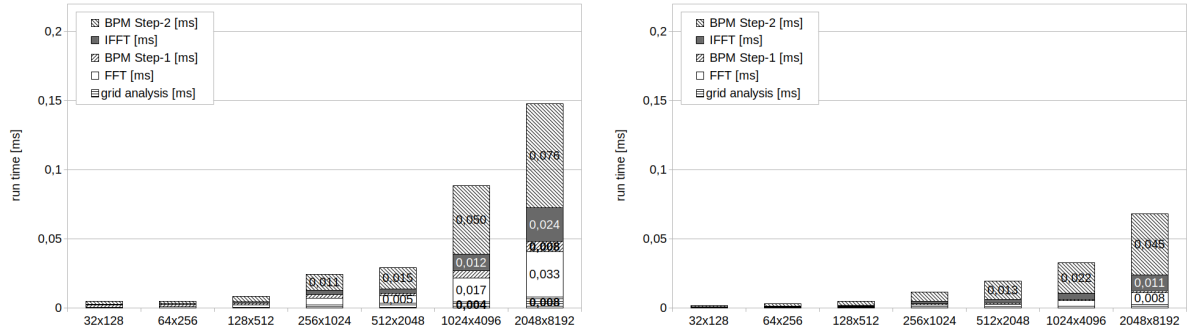

Fig. 6. CPU (2.1GHz, 1 core, single-thread) timing breakdown for the first iteration (left) and the $n_z/2$-th iteration (right) with 'FreqOpt+SpatOpt' and Gaussian beams traversing random symmetric inhomogeneous grids.

The complexities of the algorithms are in $\mathcal{O}(n)$ and the gradients of the linear extrapolations show an increasing gap in run times as shown in sections 4.3 and 4.4. Hence, the run time improvements with the proposed optimizations tend to increase with the grid size under the given circumstances in comparison to the massively parallel implementation.

The proposed optimizations for the SS-BPM achieve a maximum run time reduction of 73 percent for homogeneous symmetric systems as shown in section 4.3 and 40 percent for inhomogeneous symmetric systems as shown in section 4.4 while the massively parallel implementation achieves a maximum run time reduction of 68 percent for homogeneous symmetric and of 27 percent for inhomogeneous symmetric systems.

The comparison of clock execution cycles for homogeneous symmetric systems shows that the massively parallel implementation of the SS-BPM performs between 30 and 40 percent less cycles than the optimized single-thread implementation as shown in section 4.3.

The comparison of clock execution cycles for inhomogeneous symmetric systems shows that the massively parallel implementation of the SS-BPM performs up to 313 percent more cycles for small grids and at least 17 percent less cycles for large grids than the optimized single-thread implementation as shown in section 4.4.

The results demonstrate that the single-thread implementation scales better with the number of samples than the parallel implementation as shown in section 4.4. A parallel implementation of the SS-BPM should be considered if the clock frequency ratio between GPU and CPU allows to compensate the gap in execution cycles and if the sequential program codes of the massively parallel implementation show comparable performance to the single-thread implementation.

The integrated grid analysis algorithm consumes only one percent of performance as shown in section 4.4 so that a separate pre-processing step would not significantly improve run times. This makes the optimizations applicable to systems that change their characteristics during simulation.

The proposed optimizations operate in-place and do not allocate additional memory. They are not based on approximations and retain the accuracy and memory footprint of the original algorithm. Hence, the optimizations achieve run time reductions up to a level of a massively parallel system without any cost.

Disclosures The author declares no conflicts of interest.

References

1. M. Feit, J. Fleck, *Light propagation in graded index fibers*, Appl. Opt., vol. 24, pp. 3390-3998, 1978.
2. Sharma, Anurag & Agrawal, Arti, *Non-paraxial Split-step Finite-difference Method for Beam Propagation*, Optical and Quantum Electronics, 38. 19-34. 10.1007/s11082-006-0019-4, Feb 2006.
3. G.R.Hadley, *Wide-Angle beam propagation method using Padé approximant operators*, Opt Lett, Vol 17, No. 20, pp 1426-1428, 1992.
4. Changbao Ma and Edward Van Keuren, *A three-dimensional wide-angle BPM for optical waveguide structures* in OPTICS EXPRESS, (22 January 2007), Vol. 15, No2.
5. P. Lee, E. Vogoes, *Three-dimensional semi vectorial wide-angle beam propagation method*, J. Lightwave Tech., vol. 12, no. 2, pp. 215-225, Feb. 1994.

6. T. Anada, T. Hokazono, T. Hiraoka, J. Hsu, T. Benson, P. Sewell, *Very-wide-angle beam propagation methods for integraded optical circuits.*, IEICE Trans Electron, Vol E82-C, No. 7, pp. 1154-1158, 1999.
7. Junji Yamauchi, Yuta Nito and Hisamatsu Nakano, *A modified semivectorial beam propagation method retaining the longitudinal field component* in Integrated Photonics and Nanophotonics Research and Applications, (Optical Society of America, 2008), paper IWB5.
8. P. Liu and B.J. Li, *Semivectorial beam-propagation method for analyzing polarized modes of rib waveguide.*, IEEE J. Euantum Electron, Vol. 28, pp 778-782, April 1992.
9. J. Wanguemert-Perez, I. Molina-Fernandez, *A novel Fourier based 3D full-vectorial beam propagation method*, Optical Quantum Electronics, vol. 36, pp. 285-301, Kluwer Academic Publishers 2004, Netherlands.
10. Y. Tsuji, M. Koshiba, N. Takimoto, *Finite element beam propagation method for anisotropic optical waveguides.*, Journal Lightwave Technology, Vol. 17, No. 4, pp.723-828, Apr. 1999.
11. M. Stern, *Semivectorial polarized finite difference method for optical waveguides with arbitrary index profiles*, IEEE Proceedings, Vol. 135, Pt.J, No 1, Feb 1988.
12. H. Pinheiro, A. Barbero, H. Hernandez-Figueroa, *Full-vectorial FE-BPM approach for the analysis of anisotropic medium with off-diagonal permittivity terms*, Mic. Opt. Tech. Letters, V.25, No 1, pp. 12-14, April 2000.
13. S.Obayya, B. Rahman, *New vectorial numerically efficient propagation algorithm based on the finite element method.*, IEEE Journal Lightwave Technology, Vol. 18, No. 3, pp. 409-415, Mar 2000.
14. P. Sewell, J. Wykes, A. Vukovic, D. W. P. Thomas, T.M.Benson, C. Christopoulos *Multi-grid interface in computational electromagnetics*, Elec. Lett. Vol.40 No.3, pp 162-163, 2004.
15. K.-H. Brenner and W. Singer, *Light propagation through microlens: a new simulation method* in Applied Optics 32, (1993), 4984-4988
16. M. Fertig and K.-H. Brenner, *Vector wave propagation method*, Journal of the Optical Society of America (JOSA) A, vol. 27, pp. 709-717, Apr 2010