UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

"Mat. Luis Manuel Rivera Gutiérrez"

ESTUDIO NUMÉRICO DE LA RESPUESTA ÓPTICA EN GUÍAS DE ONDAS DE CRISTAL FOTÓNICO USANDO PROGRAMACIÓN EN PARALELO CON LOS PROTOCOLOS DE MPI Y CUDA

TESIS

Para obtener el grado de

LICENCIADO EN CIENCIAS FÍSICO MATEMÁTICAS

PRESENTA:

ELIEZER LOZANO TREJO

ASESOR DE TESIS:

Doctor en Ciencias en Óptica
DR. HÉCTOR I. PÉREZ AGUILAR

Morelia, Michoacán, Febrero de 2017

RESUMEN

Con el paso del tiempo, los problemas que han surgido en la ciencia han aumentado en cantidad, así como en dificultad. Lo que a una persona le tomaría días o incluso meses para realizar cálculos numéricos, ahora son resueltos por una computadora en minutos. Sin embargo, en la actualidad los problemas se han vuelto cada vez más difíciles de resolver (esto normalmente era compensado con procesadores más rápidos) que en poco tiempo ya alcanzaron sus límites. Esto implicó buscar otras soluciones para mejorar el rendimiento de las computadoras creando finalmente lo que hoy se tiene: los procesadores multinúcleos. Todo esto provocó un cambio de paradigma en la forma de programación, dando el nacimiento al concepto de paralelización y la creación de diferentes herramientas que han ayudado a que esta actividad se vuelva más sencilla. Algunas de estas herramientas son: MPI (Message Passing Interface), OpenMP (Open Multi-Processing), OpenCL (Open Computing Language) y CUDA (Computer Unified Device Architecture).

En el presente trabajo, se muestra un gran avance para mejorar el rendimiento que se puede alcanzar utilizando algunas de estas herramientas especializadas en el cómputo paralelo. Para lograr esto hemos utilizado el modelo de programación en paralelo bajo distintos protocolos y librerías como son: MPI y ScaLAPACK (Scalable Linear Algebra PACKage) usando los procesadores de la CPU y CULA (Compute Unified Linear Algebra PACKage) usando núcleos de la GPU. Estas librerías fueron aplicadas a un problema particular de álgebra lineal, como es la inversión de una matriz; asimismo, esta herramienta fue utilizada como parte de una técnica numérica (conocida como el Método de la Ecuación Integral) para calcular la respuesta óptica de una guía de ondas de cristal fotónico (PCW) bidimensional.

Los resultados para la inversión de una matriz general en precisión sencilla, muestran que el mejor rendimiento se logró con las subrutinas de CULA obteniendo una rapidez de 98 veces más que las versiones secuenciales utilizando las técnicas LU y Gauss Jordan. Estas formas de programación en paralelo fueron aplicadas al método integral, usando tanto los procesadores como la tarjeta gráfica; así como en la forma secuencial para hacer un comparativo del tiempo de cómputo variando el número de periodos de la PCW. Con la versión secuencial usando la librería MKL de Intel se obtuvo una rapidez de 26 veces más con respecto a la versión secuencial de LAPACK. En cambio con las versiones paralelas con los procesadores de la CPU, usando ScaLAPACK se tuvo una rapidez de 32 veces; así como las versiones de MPI con comunicación No bloqueante y Colectiva fueron 46 y 53 veces más rápido, respectivamente. Por otro lado, para las versiones que usan la programación en paralelo con la GPU, tanto la que nada más invierte la matriz, como la que construye e invierte la matriz se logró una rapidez de 37 y 118 veces más que la versión secuencial de LAPACK. Esto nos permite concluir que con estas técnicas de programación en paralelo bajo el protocolo de MPI y CUDA permiten modelar sistemas complejos en tiempos de cómputo relativamente muy cortos.

Palabras clave: Programación en Paralelo, MPI FORTRAN, ScaLAPACK, CUDA, Método de la Ecuación Integral.

ABSTRACT

With the passage of time, the problems that have arisen in the science have increased in quantity, as well as in difficulty. What a person took days o even months to perform numerical calculations, now are resolved by a computer in minutes. Nevertheless, today the problems have become increasingly difficult to resolve (this normally was compensated by processors faster) that in a little time already they reached its limits. This implies looking for other solutions to improve the performance of the computers by creating what we have today like: multi-core processors. All this has provoked a change of paradigm in the form of programming, giving birth to the concept of parallelization and the creation of different tools that have helped this activity turns simpler. Some of these tools are MPI (Message Passing Interface), OpenMP (Open Multi-Processing), OpenCL (Open Computing Language) and CUDA (Computer Unified Device Architecture).

In the current work, it shows a great advance to improve the performance that can be reached using some of these specialized tools in the parallel computing. In order to achieve this, we have used the model of parallel programming under different protocols and libraries which are: MPI and ScaLAPACK (Scalable Linear Algebra PACKage) using the processors CPU and CULA (Computer Unified Linear Algebra PACKage) using the cores of GPU. These libraries were applied to a particular problem of linear algebra, which is the inverse of a matrix; likewise, this tool was used as a part of a numerical technique (known as the Integral Equation Method) to calculate the optical response of the two-dimensional Photonic Crystal Waveguide.

The results for the inverse of a general matrix in simple precision show that the best performance was achieved by CULA obtaining a 98 times faster than the sequential versions using the LU and Guass Jordan techniques. These forms of programming in parallel were applied to the integral method, using both the processors and the graphical card; as well as in the sequential form to do a comparative of the computing time varying the number of periods of the PCW. With the sequential version using the libraries MKL of Intel there was a speed of 26 times with regard to the sequential version of LAPACK. The parallel versions with the processors of the CPU, using ScaLAPACK gave a speed of 32 times; like the versions of MPI with communication non-blocking and collective were 46 and 53 faster, respectively. On the other hand for the versions using parallel programming with GPU, the one that only inverts the matrix and the one that constructs and inverts the matrix both achieve a speed of 37 y 118 times more than the sequential version of LAPACK. This allows us to conclude that with these techniques of parallel programming under the protocol of MPI and CUDA allow to model complex systems in relatively short computing times.

Keywords: Parallel Programming, MPI FORTRAN, ScaLAPACK, CUDA, Integral Equation Method.

Dedicatorias

Mi dedicación especial en el presente trabajo de tesis está dirigida a mis padres Noé Lozano Laurel y Salud Trejo Cabrera quienes me dieron la vida y han estado conmigo en todo momento. Gracias por todo papá y mamá por darme una carrera para mi futuro y por creer en mí.

También dedico este trabajo a mis tíos Wilbert Molina Toledo y Yazmin Trejo Cabrera, así como a mi hermana Itzanami quien me ayudo a terminar esta tesis y a todos mis familiares quienes me apoyaron todo el tiempo.

Así como a mis amistades más sinceras que estuvieron conmigo en los momentos buenos y malos.

A mis maestros quienes nunca desistieron al enseñarme, a los sinodales quienes estudiaron mi tesis y la aprobaron.

A todos los que me apoyaron para escribir y concluir esta tesis.

Sinceramente

ELIEZER LOZANO TREJO

Agradecimientos

En primer lugar deseo agradecer sinceramente a mi asesor de tesis, Dr. Héctor I. Pérez Aguilar, por su esfuerzo y dedicación.

Agradezco en forma especial a los sinodales, quienes me hicieron las correcciones y observaciones de mi trabajo de tesis.

De igual forma deseo agradecer a mis padres y familiares por el apoyo tanto moral y económico para seguir estudiando y lograr el objetivo trazado.

También agradezco a todos mis amigos que ayudaron a terminar esta tesis.

Atentamente

ELIEZER LOZANO TREJO

Contenido

Página

Contenido (continuación)

Página

Lista de Figuras

Capítulo I

INTRODUCCIÓN

Con el nacimiento de las computadoras, se comenzó una etapa de progresos significativos en las ciencias, desde entonces la computadora ha sido una herramienta indispensable por su fiabilidad y rapidez en procesar grandes cantidades de datos. Una persona que tardaría horas en realizar cálculos, ahora son resueltos por una computadora en segundos. Sin embargo, actualmente existen problemas que demandan todavía mucho más tiempo de procesamiento en la computadora; por ejemplo la simulación del clima, la simulación de mecánica de fluidos, entre otros y solucionar estos problemas puede tardar desde minutos hasta días o meses. Es debido a este tipo de problemas que se produjo un cambio de paradigma en la forma en la que se puede resolver los problemas y este nuevo tipo de paradigma fue la paralelización.

La paralelización ha sido clave no sólo para aumentar el rendimiento en los problemas sino para reducir costos. Todo esto fue gracias a la creación de diferentes herramientas para el desarrollo de programas en paralelo que han ayudado a que esta actividad se vuelva sencilla. Algunas de estas herramientas son: MPI (Message Passing Interface)(Pacheco, 2011), OpenMP (Open Multi-Processing)(Chapman *et al.*, 2008), OpenCL (Open Computing Language)(Kaeli *et al.*, 2015) y CUDA (Computer Unified

Device Architecture)(Cook, 2013).

La paralelización consiste en la subdivisión de un problema, en subproblemas, cada uno de los cuales es resuelto de forma simultánea y por separado, ofreciendo reducciones importantes en cuanto al tiempo de ejecución. Sin embargo, no todos los problemas son aptos para el empleo de la paralelización, y los que sí lo son, no tienen porqué responder de forma positiva a su resolución paralela.

En este trabajo de tesis se plantea la aplicación de la programación en paralelo en diferentes paradigmas para estudiar problemas novedosos, como son los cristales fotónicos.

Los cristales fotónicos fueron propuestos a finales de los años ochenta como una posible solución al control de la emisión espontánea y a la localización de la luz (Yablonovitch, 1987; John, 1987). Para estudiar los cristales fótonicos hace falta el uso de métodos numéricos; entre los que se destacan el método de ondas planas (PWE[1])(Sözüer *et al.*, 1992), el método de Diferencias Finitas en Dominio del tiempo (FDTD[2])(Inan and Marshall, 2011) y el método de la ecuación integral (IEM[3])(Mendoza-Suárez *et al.*, 2006).

El método de la ecuación integral consiste en partir de las condiciones de frontera y del segundo teorema integral de Green, obteniéndose un par de ecuaciones integrales con las cuales se puede determinar el campo dentro y fuera de la guía de ondas en términos de un par de funciones fuente. Debido a que no es posible resolver de manera analítica el sistema de ecuaciones integrales acopladas, recurrimos a una discretización del sistema. Para esto es necesario convertir las ecuaciones integrales obtenidas en un sistema de ecuaciones algebraico que se puede resolver por medio de métodos numéricos

[1]Por sus siglas en inglés, Plane Wave Expansion.

[2]Por sus siglas en inglés, Finite Diference Time Domain.

[3]Por sus siglas en inglés, Integral Equation Method.

matriciales. El numńero de ecuaciones algebraicas que se obtienen depende del número de puntos de muestreo que se requieren para construir los contornos de las regiones que se tienen en el sistema. Resolver la matriz generada por el método numérico matricial así como llenar dicha matriz requiere elevados recursos y tiempos de cómputo.

En este trabajo, se muestran dos aplicaciones de la programación en paralelo. La primera aplicación hace uso de la biblioteca de álgebra lineal para cómputo en paralelo ScaLAPACK, la cual se usa para encontrar la inversa de una matriz. La segunda aplicación hace uso de la programación en paralelo bajo los protocolos de CUDA C y CUDA FORTRAN en la GPU y la Interfaz de Paso de Mensajes (MPI) en la CPU (ver Fig. 1), para presentar un estudio numérico de la respuesta óptica en guías de ondas de cristal fotónico PCW[4] bidimensional finita aplicando el método de la ecuación integral (el cual será descrito en el capítulo 4)(Mendoza-Suárez and Pérez-Aguilar, 2016). Para esto se aplicarán distintos paradigmas de la programación en paralelo y se hará uso de los múltiples recursos con los que cuentan hoy en día una computadora. Todo esto con el fin de reducir los tiempos de cómputo.

OpenMP OpenMPI Cg, CUDA, OpenCL

Figura 1. OpenMP (Open Multi-Processing), MPI (Message Passing Interface), CUDA (Computer Unified Device Architecture).

[4]Por sus siglas en inglés, Photonic Crystal Waveguides.

I.1. Estructura de la tesis

Este trabajo de tesis está estructurado en seis capítulos que a continuación describiremos.

En el capítulo II se presentarán los conceptos básicos relacionados con la programación en paralelo, así como un pequeña introducción de los distintos lenguajes y bibliotecas que se usarán. De esta forma se pretende explicar conceptos importantes que servirán para el presente trabajo.

En el capítulo III se mencionan las características de la biblioteca ScaLAPACK, así como por que fue desarrollada y cuales fueron sus objetivos. Al final del capítulo se presentará una aplicación de ScaLAPACK a un problema particular de álgebra lineal.

En el capítulo IV se describe una técnica rigurosa para modelar la interacción de la luz con una PCW finita. La técnica se conoce como el Método de la Ecuación Integral. En particular, se aplicará para estudiar la propagación de la luz a través de una PCW calculando la respuesta óptica como función de la frecuencia.

En el capítulo V desarrollan los algoritmos bajo los paradigmas de ScaLAPACK y CULA para calcular la respuesta óptica del sistema de estudio. Además se muestran los resultados numéricos calculando los tiempos de cómputo de los diferentes paradigmas que se consideraron en este trabajo.

Finalmente en el capítulo VI se presentan las conclusiones más importantes sobre los resultados obtenidos en el desarrollo de la tesis de investigación.

Capítulo II

PROGRAMACIÓN EN PARALELO CON MPI Y CUDA

En este capítulo se presentan los conceptos básicos relacionados con la programación en paralelo, así como una pequeña introducción de los distintos lenguajes y bibliotecas que se usan para la programación en paralelo.

II.1. Descripción general en paralelo

La computación en paralelo es una técnica de programación en la que muchas instrucciones se ejecutan simultáneamente. Se basa en el principio de que los problemas grandes se pueden dividir en partes más pequeñas que se pueden resolverse de forma concurrente.

Para poder paralelizar un problema y mejorar el rendimiento, es necesario conocer algunos conceptos fundamentales sobre paralelización y conocer el ambiente de trabajo tales como:

- Arquitectura de la computadora (Hardware).
- Soporte del sistema operativo.

- Herramienta de software para la implementación.

En la computación en paralelo, el software y hardware están estrechamente relacionados. A fin de lograr la ejecución paralela del software, el hardware debe proporcionar una plataforma que admita la ejecución simultánea de varios procesos o múltiples subprocesos.

En este capítulo se describen algunos conceptos fundamentales sobre la programación en paralelo y el ambiente de trabajo necesario para llevar a cabo un óptimo desarrollo de un programa en paralelo (Pérez Hernández, 2015; Sánchez López, 2016).

II.1.1. Conceptos y terminologías de la programación en paralelo

- **Tarea**: Sección lógica discreta de trabajo computacional. Suele ser un programa o un conjunto de instrucciones que es ejecutable por un procesador.

- **Tarea en paralelo**: Tarea que puede ser ejecutada de forma segura (produciendo resultados correctos) por múltiples procesadores simultáneamente.

- **Ejecución en serie**: Ejecución de un programa de forma secuencial; es decir, una expresión en cada instante de tiempo. No obstante, todas las tareas paralelas tendrán secciones de un programa paralelo que deben ser ejecutadas en serie.

- **Ejecución en paralelo**: Ejecución de un programa por más de una tarea, siendo cada tarea capaz de ejecutar la misma expresión o una diferente, y todas en un mismo instante de tiempo.

- **Algoritmos paralelos**: Son métodos para resolver problemas computacionales en máquinas paralelas.

- **Memoria compartida**: Desde el punto de vista del hardware, la memoria compartida describe una arquitectura de ordenador, donde todos los procesadores tienen acceso directo a una memoria física común. Desde el punto de vista del software, describe un modelo donde todas las tareas paralelas tienen la misma representación de la memoria, que pueden direccionar y acceder directamente a las mismas localizaciones de una memoria lógica, sin preocuparse donde se encuentra la memoria física.

- **Memoria distribuida**: En el sentido del hardware, la memoria distribuida se refiere a un acceso a memoria basado en red para una memoria física que no es común. Como modelo de programación, las tareas sólo pueden ver lógicamente la memoria de la máquina local y deben utilizar comunicaciones para acceder a la memoria de otras máquinas donde se ejecutan el resto de tareas.

- **Comunicaciones**: Típicamente, las tareas paralelas necesitan intercambiar datos. Hay varios caminos para realizar esto, tales como tener una memoria compartida en bus o sobre una red. Sin embargo, en la actualidad, el hecho de intercambiar datos se le denomina comunicaciones, independientemente del método empleado.

- **Sincronización**: Es la coordinación de tareas paralelas en tiempo real, a menudo asociado a las comunicaciones. La forma en que se implementa la sincronización entre tareas es poniendo un punto de sincronismo dentro del código de una aplicación, de modo que las tareas no continuarán con su trabajo hasta que todas las tareas hayan llegado al mismo punto de sincronismo. La sincronización implica la espera de, al menos una tarea y, por lo tanto, puede causar el incremento del tiempo de ejecución de la aplicación paralela. El tema del sincronismo es muy importante y es uno de los temas más delicados a tratar a la hora de escribir

aplicaciones en paralelo.

- **Granularidad**: En la computación en paralelo, la granularidad es una medida cualitativa de la tasa de tiempo de computación entre tiempo de comunicaciones. En el límite, granularidad gruesa (fina) es cuando grandes (pequeñas) cantidades de trabajo computacional son realizadas en medio de eventos de comunicaciones.

- **Concurrencia**: Define la ejecución simultánea de dos o más procesos en uno o más procesadores.

- **Escalabilidad**: Capacidad de un sistema paralelo (hardware o software) para demostrar un incremento proporcional en la velocidad de ejecución en paralelo con el aumento del número de procesadores. Un factor importante que influye en la escalabilidad es la forma en que se ha diseñado el código de la aplicación paralela a ejecutar.

- **Rendimiento**: Es la proporción entre el resultado que se obtiene y los medios que se emplearon para alcanzar al mismo. En nuestro caso esto es de suma importancia, debido a que entre más procesadores se involucran en la solución de un determinado problema, mejor será el tiempo de procesamiento, pero también incrementará el tiempo de comunicación entre los procesadores involucrados. En el tiempo de cómputo, Amdahl define los límites de aceleración que una aplicación puede alcanzar (Amdahl, 1967). En el tiempo de comunicación, propone varios modelos, donde algunos estiman el tiempo con un conjunto de parámetros y otros parámetros más detallistas, para los patrones de comunicación punto a punto y colectiva.

Ley de Amdahl

En cualquier programa paralelo es imposible dejar que el código secuencial deje de existir, el cual no puede ser paralelizable, y por lo tanto debe ser ejecutado por un sólo procesador. Si la fracción del problema inherentemente secuencial es f, entonces el tiempo de ejecución de la parte secuencial del programa es $f \times t_1$ y el tiempo en ejecutar la parte paralelizable en p procesadores es $(1-f) \times t_1/p$. Luego el tiempo de ejecución del programa paralelizado será:

$$t_p = f \times t_1 + \frac{(1-f) \times t_1}{p} \tag{1}$$

y por lo tanto, la aceleración será:

$$S_p = \frac{t_1}{t_p} = \frac{1}{f + \frac{(1-f)}{p}}. \tag{2}$$

De lo cual se observa, que aunque se contara con una infinidad de procesadores, la aceleración estará limitada por $1/f$, como puede verse de la siguiente relación:

$$\lim_{p\to\infty} S_p = \lim_{p\to\infty} \left[\frac{1}{f + \frac{(1-f)}{p}} \right] = \frac{1}{f}. \tag{3}$$

De manera análoga, la eficiencia de un programa se acercaría a cero conforme se aumenta el número de procesadores y el programa paralelizado será más susceptible de perder el rendimiento del sistema,

$$\lim_{p\to\infty} E_p = \lim_{p\to\infty} \left[\frac{S_p}{p} \right] = 0. \tag{4}$$

La mejora en la velocidad de ejecución de un programa como resultado de la paralelización está limitada por la porción del programa que no se puede paralelizar como se muestra en Fig. 2.

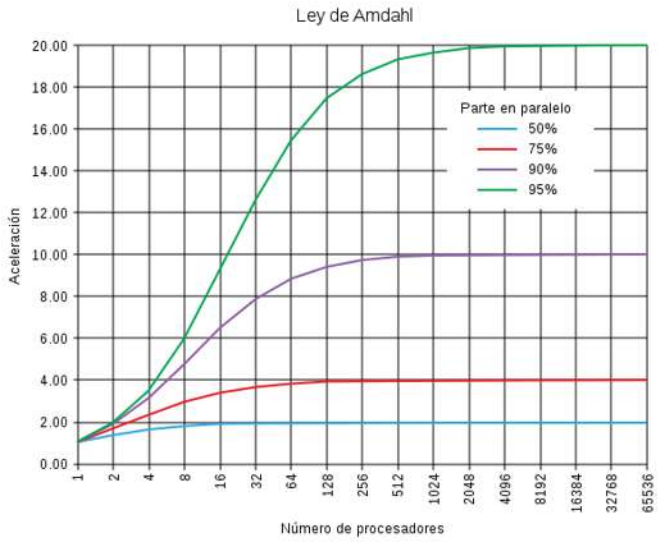

Figura 2. Representación gráfica de la ley de Amdahl.

II.2. Hardware

Existen dos tipos de hardware para llevar a cabo la computación en paralel. Uno de ellos es la unidad de procesamiento central (CPU[5]) y el otro es la unidad de procesamiento gráfico (GPU[6]). Cada uno con diferentes filosofías de diseño como se muestra en la Fig. 3.

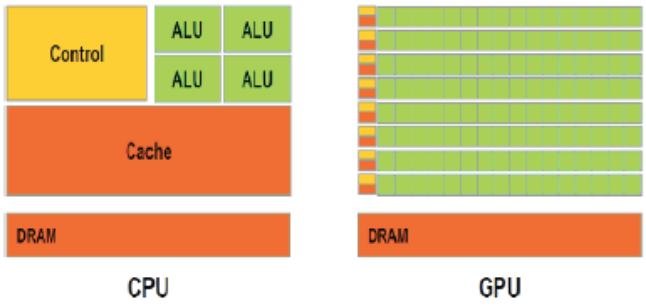

Figura 3. CPU y GPU, diferencia de filosofía de diseño.

[5]Por sus siglas en inglés, Central Processing Unit.

[6]Por sus siglas en inglés, Graphics Processor Unit.

Para más información relacionada con la arquitectura del CPU y GPU consultar la Ref. (Sánchez López, 2016).

II.3. Plataformas de programación en paralelo

Los lenguajes de programación en paralelo, las bibliotecas, las APIs[5] y los modelos de programación en paralelo han sido creados para la programación de computadoras con arquitectura en paralelo. Hay muchos lenguajes y modelos de programación en paralelo. Los que más se utilizan son MPI(Pacheco, 2011) y OpenMP(Chapman *et al.*, 2008) para el caso de CPU, y para GPU existen CUDA(Cook, 2013) y OpenCL(Kaeli *et al.*, 2015). Éste último se desarrolló con el fin de poder escribir programas que se puedan ejecutar a través de plataformas heterogéneas.

II.3.1. OpenMP

OpenMP es una API para la programación multiproceso de memoria compartida en múltiples plataformas. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca y variables de entorno que facilitan la descripción de una o más porciones de un programa que deben ser ejecutadas por varias unidades de procesamiento de manera simultánea. Una gran ventaja que ofrece es que de una manera muy simple se pueden paralelizar programas secuenciales sin grandes cambios en el código fuente.

II.3.2. MPI

Interfaz de paso de mensajes (MPI) es un modelo de comunicación ampliamente usado en computación paralela. Una de las ventajas que ofrece es que, la sintaxis se ha

[5]Por sus siglas en inglés, Application Programming Interface.

estandarizado y cuenta con una amplia cantidad de funciones que se encuentran en su biblioteca. Esto garantiza la portabilidad de los programas paralelos; sin embargo, los resultados pueden variar de una implementación a otra. Para más detalles de como implementar programas usando MPI consultar la Ref. (Pérez Hernández, 2015, 30–51).

SCALAPACK

ScaLAPACK (Scalable Linear Algebra PACKage) es una biblioteca de alto rendimiento diseñada para la computación heterogénea. Sus rutinas son un rediseño de LAPACK (Linear Algebra PACKage) para computadoras paralelas de memoria distribuida. Resuelve sistemas lineales densos, problemas de mínimos cuadrados, problemas de valores propios y de valores singulares entre otros, todo esto haciendo uso de la distribución cíclica en bloques y algoritmos de bloque dividido. Esto nos garantiza que el programa se ejecuta tan rápido como sea posible y que funcionará en cualquier máquina donde BLAS, LAPACK y BLACS estén disponibles (Blackford *et al.*, 1997).

II.3.3. CUDA

CUDA (Computer Unified Device Architecture) es una plataforma de computación en paralelo que presenta un nuevo modelo de programación y un conjunto de instrucciones, que en conjunto nos permite implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad. El desarrollador puede usar una variación del lenguaje de programación C para codificar los algoritmos. También es compatible con otros lenguajes de programación como: C++, Python, FORTRAN y Java. En este trabajo usaremos dos lenguajes de programación los cuales son: CUDA C y CUDA FORTRAN. Para mayor detalle consultar la Ref. (Sánchez López, 2016, 24–38).

CULA

CULA (EM, 2009) es una implementación de LAPACK pero rediseñada para funcionar en plataformas NVidia CUDA masivamente en paralelo sobre GPUs. CULA ofrece a los programadores dos diferentes interfaces de manipulación de datos: la interfaz de host y la interfaz del device. En otras palabras, la interfaz de host funciona con datos ubicados en la memoria host y la interfaz del device funciona en los datos que ya se encuentran en el acelerador de GPU.

II.3.4. OpenCL

OpenCL (Open Computing Language) es tanto una interfaz de programación de aplicaciones y de un lenguaje de programación, lo cual nos permite escribir programas con paralelismo de datos y de tareas, que se ejecutan a través de plataformas heterogéneas, como son: CPU, GPU, DSP (Digital Signal Processors), FPGAs (Field-Programmable Gate Arrays), entre otros. En la presente tesis no se hará uso de OpenCL ya que por la falta de librerías optimizadas en el área de soluciones de ecuaciones lineales resulta impráctico su uso. Esperamos en un futuro estar incorporando esta gran herramienta en el área de simulaciones de sistemas periódicos como son los cristales fotónicos.

Capítulo III

MODELO DE PROGRAMACIÓN DE SCALAPACK

ScaLAPACK es una biblioteca de alto rendimiento, de rutinas de álgebra lineal para memoria distribuida, paso de mensajes y computadoras MIMD; además, es la continuación del proyecto LAPACK. Ambas bibliotecas contienen rutinas para resolver sistemas de ecuaciones lineales, problema de mínimos cuadrados y problemas de valores propios.

Los objetivos de ambas bibliotecas son los mismos, los cuales son: la eficiencia (que se ejecute lo más rápido posible), la escalabilidad, fiabilidad (incluyendo márgenes de error), portabilidad (que pueda correr en todas las máquinas con arquitectura paralela), flexibilidad (que los usuarios puedan construir rutinas) y facilidad de uso (realizando la interfaz de LAPACK y ScaLAPACK para que fueran lo más parecido posible). Estos objetivos se llevaron acabo gracias a que se limitaron la mayoría de las dependencias de la máquina a dos bibliotecas estándar, las cuales son: BLAS (Basic Linear Algebra Subprograms) (Anderson *et al.*, 1999) y BLACS (Basic Linear Algebra Communication Subprograms) (Blackford *et al.*, 1997, 5–6).

LAPACK puede ser ejecutado en cualquier máquina que cuente o esté disponible BLAS y ScaLAPACK de igual forma puede ser ejecutada en cualquier máquina que

cuente o que esté disponible BLAS y BLACS.

Actualmente ScaLAPACK está escrita en FORTRAN (con la excepción de una pocas rutinas auxiliares simbólicas de valores propios que están escritas en C para explotar la aritmética IEEE). Para mayor detalle de la aritmética IEEE consultar la Ref. (Blackford *et al.*, 1997, 124–125).

Además de los tipos de problemas que se mencionaron anteriormente que pueden ser resueltos con ScaLAPACK, también se puede manejar muchos cálculos asociados tales como factorizaciones de matrices, entre otros.

Al final del capítulo se muestra la aplicación de ScaLAPACK en un problema de álgebra lineal (inversión de una matriz), el cual tiene una gran importancia para nosotros ya que lo usaremos para aplicar un método numérico riguroso que se conoce como el Método de la Ecuación Integral.

III.1. Estructura

Así como LAPACK, las rutinas de ScaLAPACK se basan en algoritmos de partición de bloques, para minimizar la frecuencia de movimiento de datos entre los diferentes niveles de jerarquía de memoria. Los bloques fundamentales de ScaLAPACK son versiones de memoria distribuida de BLAS de Nivel 1, Nivel 2 y Nivel 3, denominado BLAS Paralela o PBLAS (Blackford *et al.*, 1997, 5–6) y BLACS. En las rutinas de ScaLAPACK, la mayoría de las comunicaciones interprocesador se produce dentro de las librerías de PBLAS.

ScaLAPACK contiene rutinas de controlador para resolver problemas de tipo estándar, rutinas para realizar una tarea de cálculo y las rutinas auxiliares para realizar una determinada subtarea o cálculo común de bajo nivel. Cada rutina de tipo controlador normalmente llama una secuencia de rutinas. Tomadas en conjunto, las rutinas pueden

realizar un rango más amplio de tareas que las que cubre las rutinas de tipo controlador. Las rutinas auxiliares pueden ser de utilidad para analistas numéricos o desarrolladores de software, ya que se pueden encontrar los códigos fuentes de las mismas con gran detalle en la página web de ScaLAPACK (Tennessee *et al.*, 2012).

III.2. Componentes

ScaLAPACK tiene dos tipos de componentes (ver Fig. 4):

- Los **locales** están conformados por las bibliotecas que son llamadas por cada procesador y cuyos argumentos se encuentran almacenados en cada procesador.
- Los **globales** están formados por las bibliotecas o rutinas paralelas, cuyos argumentos incluyen matrices, vectores entre otros y que se encuentran distribuidos a través de múltiples procesadores.

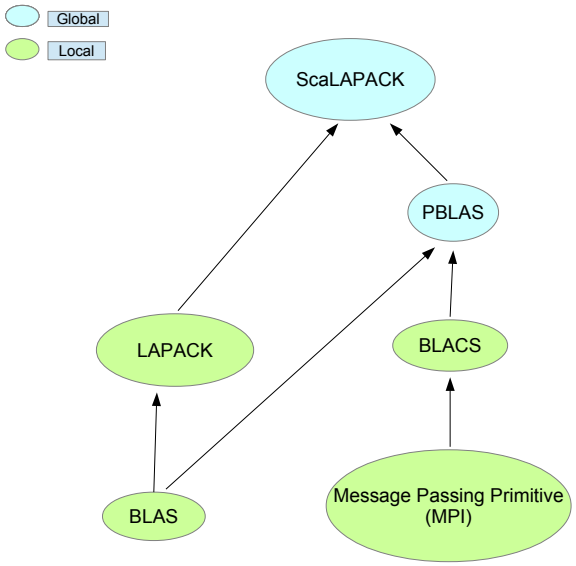

Figura 4. Descripción gráfica de la jerarquía de ScaLAPACK.

III.2.1. BLAS

Basic Linear Algebra Subprograms (BLAS) es una biblioteca que contiene subrutinas para cálculos de álgebra lineal. Las rutinas BLAS se clasifican en tres niveles, que corresponden tanto al orden cronológico de su definición y su publicación, como al grado del polinomio en la complejidad del algoritmo.

Nivel 1

Este nivel tiene definido sólo las operaciones vectoriales en matrices, como el producto punto, las normas de vectores y la suma de vectores de forma generalizada, entre otras operaciones. Por ejemplo,

$$y \leftarrow \alpha x + y.$$

Nivel 2

El nivel 2 contiene las operaciones de matriz-vector que incluye entre otras cosas, multiplicación generalizada Matriz-Vector (GEMV),

$$y \leftarrow \alpha Ax + \beta y.$$

Así como para encontrar la solución de la ecuación lineal de la forma:
$Tx = y$, siendo T triangular.

Nivel 3

En este nivel están definidas las operaciones entre matrices, incluyendo la multiplicación generalizada de matrices (GEMM), de la forma

$$C \leftarrow \alpha AB + \beta C.$$

También cuenta con rutinas para resolver sistemas de la forma

$B \leftarrow \alpha T^{-1}B$, siendo T una matriz triangular.

III.2.2. PBLAS

Parallel Basic Linear Algebra Subprograms (PBLAS). Como BLAS demostró ser de gran utilidad para LAPACK. Los desarrolladores de ScaLAPACK optaron por construir su versión de BLAS paralelo (PBLAS), cuya interfaz sea lo más parecida a BLAS. Esto permitió que el código de ScaLAPACK sea bastante similar a LAPACK.

PBLAS sólo implementa el Nivel 2 y 3 de BLAS, y depende del nivel 1 de BLAS, cuyas operaciones se realizan de manera local en cada procesador. Para esto únicamente se le añadió una nueva rutina, la cual fue la transposición de una matriz, ya que ésta es una operación bastante complicada en un entorno de memoria distribuida. PBLAS opera en matrices distribuidas en un esquema cíclico de bloques en 2D. Esto será de gran importancia más adelante y se explicará con más detalle en su momento.

III.2.3. BLACS

Basic Linear Linear Algebra Communication Subprograms (BLACS) es una biblioteca de paso de mensajes diseñada para álgebra lineal. BLACS incluye rutinas de envío y recepción síncronas para comunicar una matriz o submatriz de un proceso a otro, o para calcular reducciones globales como sumas, máximos y mínimos entre otros. BLACS fue creado para hacer que las aplicaciones de álgebra lineal sean más fáciles de programar y más portátiles.

III.3. Portabilidad y Eficiencia

La estrategia que se usa en ScaLAPACK para combinar la portabilidad y la eficiencia se basa en construir todo el software de tal forma que éste realice llamadas a los subprogramas de PBLAS. Éste a su vez se apoye de los subprogramas de BLAS para el cómputo local y de BLACS para las comunicaciones entre procesadores.

La eficiencia de ScaLAPACK depende del uso de algoritmos de partición en bloques y de las implementaciones eficientes de BLAS y BLACS para cada arquitectura. BLAS y BLACS forman una interfaz de bajo nivel entre ScaLAPACK y la arquitectura de la máquina. Por encima de este nivel todo el software de ScaLAPACK es portátil.

III.4. Inversión de Matriz

La capacidad de invertir grandes matrices con rapidez y precisión es un paso esencial en una amplia gama de problemas numéricos; por ejemplo, la solución de ecuaciones lineales, la representación en 3D, filtrado de imágenes, entre otros.

III.4.1. Función GESV de ScaLAPACK

ScaLAPACK cuenta con la rutina *GESV* que calcula la solución de un sistema de ecuaciones lineales. La función para precisión simple y matrices reales es *PSGESV*. El prototipo de la rutina es: PSGESV (INT N, INT NRHS, FLOAT A0, INT IA, INT JA, INT DESCA, INT IPIV, FLOAT B0, INT IB, INT JB, INT DESCB, INT INFO).

Entrada

- **N** (global): Número de columnas de la matriz A (global) y el número de filas de la matriz solución B (global).

- **NRHS** (global): Número de columnas de la matriz solución B (global).
- **A0** (local): Matriz de coeficientes de $(MXLLDA, MXLOCC)$.
- **IA** (global): Índice de la fila en la matriz A (global) que indica la primera fila de $A(IA : IA + M - 1, JA : JA + N - 1)$.
- **JA** (global): Índice de la columna en la matriz A (global) que indica la primera columna de $A(IA : IA + M - 1, JA : JA + N - 1)$.
- **DESCA** (global y local): Arreglo descriptor de la matriz A.
- **IPIV** (global y local): Matriz de dimensión $(MXLOCR+NB)$, guarda los índices de las filas que son permutadas.
- **B0** (local): Matriz de coeficientes de $(MXLLDB, MXRHSC)$, de entrada contiene el lado derecho del problema y a la salida la matriz de soluciones X.
- **IB** (global): Índice de la fila en la matriz B (global) que indica la primera fila de $B(IB : IB + N - 1, JB : JB + NRHS - 1)$.
- **JB** (global): Índice de la columna en la matriz B (global) que indica la primera columna de $B(IB : IB + N - 1, JB : JB + NRHS - 1)$.
- **DESCB** (global y local): Arreglo descriptor de la matriz B.
- **INFO** (global y local): Indica si la rutina tuvo éxito o no.

Salida

$X = A^{-1} \times B$.

Implementación de la función GESV

Queremos calcular la matriz inversa real y simple de A de tamaño $N \times N$, entonces usaremos la función $PSGESV$. Tomando la ecuación de salida $X = A^{-1} \times B$, hacemos que $B \to I$, de esta forma obtenemos que: $X = A^{-1} \times I$, que es la matriz inversa de A. De esta manera, necesitamos que la matriz A sea invertible.

Así, para este trabajo implementamos un programa en FORTRAN que calcula la inversión de matriz usando la función $PSGESV$ de ScaLAPACK. En el programa 1 se muestra el algoritmo.

****

Programa 1. Inversión de matriz con ScaLAPACK

****

```
PROGRAM MAIN

------------------------------------------------------------------------
```

SECCIÓN 1.- Inicialización de variables, arreglos y declaración de rutinas externas

```
------------------------------------------------------------------------

INTEGER             DLEN_, IA, JA, IB, JB, M, N, MB, NB, RSRC,
$                   CSRC, MXLLDA, MXLLDB, NRHS, NBRHS,
$                   MXLOCR, MXLOCC, MXRHSC,RE,CO

PARAMETER          (DLEN_=9,IA=1,JA=1,IB=1,JB=1,RE=5,CO=5,
$                   M = 5120,N = M,MB=64,NB=64,RSRC = 0,
$                   CSRC = 0, MXLLDA =( N/RE )+(  M-1 ),
$                   MXLLDB=( N/RE ) + ( MB-1 ),NRHS=N,
$                   NBRHS= (N/RE ) + ( MB-1  ),
$                   MXLOCR=( N/RE ) + ( MB-1 ),
$                   MXLOCC =(N/CO + (NB-1), MXRHSC = (N/CO) + (NB-1))

INTEGER             INFO, MYCOL, MYROW, NPCOL, NPROW
INTEGER             DESCA ( DLEN_ ), DESCB ( DLEN_ ), IPIV (MXLOCR+NB),
$                   TC ( CO,N ) ,TR ( RE,N  ), I, J

REAL                A( N,N ), AO( MXLLDA, MXLOCC ),
```

```
$                   B( N,N ), BO( MXLLDB, MXRHSC ),

EXTERNAL            BLACS_EXIT, BLACS_GRIDEXIT, BLACS_GRIDINFO,
$                   DESCINIT, MATINIT, PSGESV,
$                   SL_INIT,DISTRIBUCIONCICLICA

DATA                NPROW / RE / , NPCOL / CO /
```

SECCIÓN 2.- Llamado a las rutinas externas e internas

```
CALL SL_INIT( ICTXT, NPROW, NPCOL )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
CALL DISTRIBUCIONCICLICA( TC, NPCOL, N, NB )
CALL DISTRIBUCIONCICLICA( TR, NPROW, N,NB )
```

SECCIÓN 3.- Llenado de la matriz A y B

```
DO I = 1, N
DO J=1,N
    IF( I.EQ. J ) THEN
      B( I, J ) = 1.0
    ELSE
      B( I, J ) = 0.0
    END IF
    A( I, J )= REAL( 1 ) / REAL( J + I - 1 )
  END DO
END DO
```

SECCIÓN 4.- Llamado a la rutina que distribuye la matriz A y B en los procesadores y llamado a la rutina PSGESV la cual es la encargada de calcular la inversa de la matriz

---

```
IF( MYROW.EQ.-1 )
```

```
$   GO TO 10
CALL DESCINIT( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA,
$                INFO )
CALL DESCINIT( DESCB, N, NRHS, NB, NB, RSRC, CSRC, ICTXT,
$                MXLLDB, INFO )
CALL MATINIT( AO, DESCA, BO, DESCB,A,B,N,NPCOL,NPROW,TC,TR)
CALL PSGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB,
$                INFO )

CALL BLACS_GRIDEXIT( ICTXT )
10 CONTINUE
CALL BLACS_EXIT( 0 )
STOP
END
```

****

A continuación explicaremos en más detalle cada parte del Programa 1.

Sección 1

En esta sección del programa se inicializan las variables, arreglos y se declaran las rutinas externas que se utilizarán en dicho programa. A continuación explicaremos para que se utilizará cada una de las variables y arreglos que se encuentran en esta sección del programa.

Variables	**Definición**
DLEN_	Almacena el tamaño que tendrán los arreglos $DESCA$ y $DESCB$.
IA	Índice de la fila en la matriz A, que indica la primera fila de A.
JA	Índice de la columna en la matriz A, que indica la primera columna de A.
IB	Índice de la fila en la matriz B, que indica la primera fila de B.
JB	Índice de la columna en la matriz B, que indica la primera columna de B.

JB	Índice de la columna en la matriz B, que indica la primera columna de B.
M	Número de filas de la matriz A.
N	Número de columnas de la matriz A y número de filas de la matriz.
MB	Número de filas del bloque de la matriz A.
NB	Número de columnas del bloque de la matriz A y número de filas del bloque de la matriz B.
RSRC	Número de la fila del procesador sobre el cual se distribuye la primera fila de la matriz.
CSRC	Número de la columna del procesador sobre el cual se distribuye la primera columna de la matriz.
MXLLDA	Dimensión máxima de la submatriz $A0$.
MXLLDB	Dimensión máxima de la submatriz $B0$.
NRHS	Número de columnas de la matriz solución B.
NBRHS	Número de columnas de la submatriz $B0$.
MXLOCR	Número máximo de filas de la submatriz $A0$ en cualquier proceso.
MXLOCC	Número máximo de columnas de la submatriz $A0$ en cualquier proceso.
RE	Número de procesos por fila.
CO	Número de procesos por columna. Nota : $RE \times CO$ debe ser igual al número total de procesadores usados.
INFO	Almacena la información de si la rutina $PSGEV$ tuvo éxito.
MYCOL	Almacena el número de columna del proceso donde se está corriendo.
MYROW	Almacena el número de fila del proceso donde se está corriendo.
NPCOL	Almacena el número total de columnas de los procesos.
NPROW	Almacena el número total de filas de los procesos.

DESCA Es un arreglo que almacena toda la información referente a la matriz A.

DESCB Es un arreglo que almacena toda la información referente a la matriz B.

IPIV Guarda los índices de las filas que son permutadas.

TC Arreglo que almacena la información de los índices de las columnas para realizar la distribución cíclica.

TR Arreglo que almacena la información de los índices de las filas para realizar la distribución cíclica.

I Contador.

J Contador.

A0 Submatriz de A, la cual será distribuida a cada proceso.

B0 Submatriz de B, la cual será distribuida a cada proceso.

Las rutinas serán explicadas a grandes rasgos en cada parte donde son utilizadas.

Sección 2

SL_INIT: Inicializa el proceso de mallado, usando un ordenamiento por "row-major".

BLACS_GRIDINFO: Esta rutina sirve para consultar la malla de procesos e identificar las coordenadas de cada proceso (*Myrow, Mycol*).

DISTRIBUCIONCICLICA: Esta rutina calcula los indices de las columnas o de las filas para realizar la distribución cíclica que es necesaria para poder utilizar la librería *PSGESV*, (ver apéndice A).

Sección 3

En esta parte del programa realiza el llenado de la matriz A para que sea invertible y la matriz B sea la identidad.

Sección 4

La matriz A y B deben ser distribuidas en la malla de procesos antes de que se pueda llamar a alguna rutina de ScaLAPACK. Cada matriz que se va a distribuir en la malla de procesos se le debe de asignar un *descriptor de matriz*.

DESCINIT: Esta rutina inicializa el descriptor de cada matriz, el cual se almacena en $DESCA$ para el caso de A y $DESCB$ para la matriz B.

MATINIT: Es la rutina encargada de hacer la distribución de las matrices A y B a cada proceso, (ver apéndice A).

PSGESV: Esta es la rutina de ScaLAPACK encargada de calcular la inversa de la matriz A.

BLACS_GRIDEXIT: Esta rutina libera la malla de procesos que se utilizó.

BLACS_EXIT: Es la rutina encargada de finalizar el programa.

III.4.2. Función GESV de CULA

Para mostrar las diferencias que existen entre la programación en ScaLAPACK y CUDA, a continuación se mostrará la implementación de un programa en CUDA FORTRAN que calcula la inversión de una matriz usando la función $SGESV$ de CULA. En el programa 2 se muestra el algoritmo.

****

Programa 2. Inversión de matriz con CULA

****

SECCIÓN 1.- Inicialización de variables y arreglos

```
PROGRAM MAIN
```

```
USE CULA_STATUS
USE SUBROUTINES
IMPLICIT NONE

REAL, DIMENSION ( :, : ), ALLOCATABLE :: A, B
INTEGER N, INFO, I, J, STATUS, NRHS
N = 1024
NRHS = N

ALLOCATE ( A ( N, N ), B ( N, NRHS ) )
```

SECCIÓN 2.- Llenado de la matriz A y B

```
DO I= 1,N
  DO J = 1, N
    IF ( I == J ) THEN
      B ( I, J ) = 1.0
    ELSE
      B ( I, J ) = 0.0
    END IF
    A ( I, J) = REAL ( 1 ) / REAL( J + I - 1 )
  END DO
END DO
```

SECCIÓN 3.- Llamado a la rutina que inicializa CULA y a la que se encarga de hacer los preparativos para usar la rutina SGESV

```
STATUS = CULA_INITIALIZE()
CALL CULA_CHECK_STATUS( STATUS )
CALL DO_CULA_DEVICE_TEST( N, NRHS, A, B )
END PROGRAM
```

SECCIÓN 4.- Inicio del la rutina encargada de llamar a la rutina SGESV, incialización de variables y arreglos

```
------------------------------------------------------------------------

MODULE SUBRUTINES
CONTAINS

SUBROUTINE DO_CULA_DEVICE_TEST( N, NRHS, AIN, BIN )
  USE CULA_LAPACK_DEVICE_PGFORTRAN
  INTEGER, INTENT(IN) :: N, NRHS
  REAL, ALLOCABLE, DIMENSION( :, : ),INTENT( IN ) :: AIN,BIN
  REAL,DIMENSION( :, : ),ALLOCATABLE :: A, B, ANS
  INTEGER STATUS

  REAL, DEVICE, DIMENSION ( :, : ), ALLOCATABLE :: A_DEV,B_DEV
  INTEGER, DEVICE, DIMENSION ( : ), ALLOCATABLE :: IPIV_DEV

  ALLOCATE( A ( N, N ), B ( N, NRHS ), ANS( N, NRHS ) )
  A( 1 : N, 1 : N ) = AIN
  B( 1 : N, 1 : NRHS ) = BIN


------------------------------------------------------------------------
```

SECCIÓN 5.- Transferencia de las matrices A y B a la memoria de la GPU, llamado a la rutina SGESV y envió de la solución a la memoria de la máquina

```
------------------------------------------------------------------------

  ALLOCATE( A_DEV ( N, N ), B_DEV ( N, NRHS ), IPIV_DEV ( N ) )
  A_DEV = A
  B_DEV = B

  STATUS = CULA_DEVICE_SGESV ( N, NRHS, A_DEV, N, IPIV_DEV, B_DEV, N )

  B = B_DEV
  DEALLOCATE ( A, B, ANS )
  DEALLOCATE (A_DEV, B_DEV, IPIV_DEV )

END SUBROUTINE DO_CULA_DEVICE_TEST
END MODULE SUBROUTINES

************************************************************************
```

A continuación explicaremos en más detalle cada parte del Programa 2.

Sección 1

En esta sección del programa se inicializan las variables y arreglos que se utilizarán. A continuación explicaremos para que se utilizará cada una de las variables y arreglos que en esta sección del programa.

N	Número de ecuaciones del sistema.
NRHS	Número de columnas de la matriz B.
A	Matriz de ecuaciones de (N, N).
B	Matriz de dimensión $(N, NRHS)$, que será llenada con el lado derecho del problema.
STATUS	Almacena la información de si CULA se inicializó correctamente.
INFO	Almacena la información de si la rutina $SGESV$ tuvo éxito.
I	Contador.
J	Contador.

Sección 2

En esta parte del programa se realiza el llenado de la matriz A para que sea invertible y la matriz B sea la identidad.

Sección 3

En esta sección se llama a la rutina encargada de inicializar CULA, así como a la rutina encargada de revisar que la inialización se llevó acabo correctamente y la encargada de realizar los preparativos para poder utilizar $SGESV$.

CULA_STATUS: Esta rutina inicializa CULA.

CULA_CHECK_STATUS: Es la rutina encargada de revisar que la inicialización de CULA se realizó correctamente.

DO_CULA_DEVICE_TEST: Llamado a la rutina encargada de realizar todos los preparativos pertinentes para utilizar SGESV y la llamada a la misma; así como regresar la solución de la inversión.

Sección 4

Inicio de la rutina encargada de realizar los preparativos para utilizar SGESV, así como declaración y inicialización de variables y arreglos que se utilizarán.

Sección 5

En esta sección se reserva memoria en la GPU para la matriz A, B e $IPIV$, se transfieren las matrices A y B localizadas en la memoria de la computadora a la memoria de la GPU, se manda llamar a la rutina $CULA\_DEVICE\_SGESV$ la cual es la encargada de encontrar la inversa, se envía la matriz solución a la memoria de la computadora y por último se libera la memoria utilizada. Para más información relacionada a la rutina $CULA\_DEVICE\_SGESV$ revisar la siguiente Ref. (Sánchez López, 2016, 87–90).

III.4.3. Comparación de tiempo de ejecución de programas secuenciales y paralelos en la inversión de una matriz

Para hacer la comparación de tiempo de ejecución entre programas secuenciales y paralelos en la inversión de una matriz se utilizó: LAPACK, ScaLAPACK y CULA. En la tabla I se muestran los tiempos de cómputo en segundos para diferentes tamaños de la matriz con las formas programación secuencial: Gauss Jordan (G.J.) y LU; así como para el caso paralelo: SCALAPACK ($4 \times 8 = 32$), SCALAPACK ($5 \times 5 = 25$) y CULA usando LU.

Tabla I. Tiempos de cómputo para la inversión de una matriz.

Entrada	**Secuencial (G.J)**	**Secuencial (LU)**	**Paralelo ScaLAPACK (LU)** $(4 \times 8 = 32)$	**Paralelo ScaLAPACK (LU)** $(5 \times 5 = 25)$	**Paralelo CULA (LU)**
1024×1024	7.76 s	0.26 s	0.43 s	0.33 s	0.028 s
1600×1600	26.97 s	0.84 s	0.61 s	0.52 s	0.058 s
2240×2240	76.28 s	1.92 s	1.30 s	0.77 s	0.093 s
3200×3200	300.05 s	6.37 s	1.99 s	1.14 s	0.208 s
4160×4160	728.96 s	14.85 s	2.91 s	2.1 s	0.358 s
5120×5120	1467.91 s	28.81 s	4.29 s	2.91 s	0.588 s
6400×6400	3371.54 s	64.74 s	6.62 s	4.44 s	1.02 s
7040×7040	4586.96 s	80.13 s	7.52 s	5.57 s	1.15 s
8000×8000	6231.66 s	116.66 s	10.40 s	7.40 s	1.68 s
9280×9280	9425.09 s	165.96 s	15.57 s	12.54 s	2.13 s
10240×10240	11347.98 s	227.28 s	20.00 s	15.76 s	2.63 s
11200×11200	15678.81 s	310.64 s	25.40 s	19.02 s	3.37 s
12480×12480	21677.66 s	442.84 s	31.02 s	23.70 s	4.62 s
13760×13760	30445.70 s	557.66 s	41.64 s	30.14 s	5.67 s

En las versiones de ScaLAPACK se usaron bloques de 64×64 ($MB = NB = 64$). Como se esperaba, en la tabla I podemos notar que los tiempos de cómputo llevados acabo por la versión secuencial son mucho más grandes que las versiones en paralelo. Además, en las versiones de ScaLAPACK un mayor número de procesadores ($4 \times 8 = 32$)

no nos garantiza que los tiempos de cómputo sean menores. Para reducir el tiempo de cómputo se tiene que hacer un mallado de la matriz de forma cuadrada ($5 \times 5 = 25$) para que los procesadores tengan una carga similar de trabajo.

De igual manera vamos a hacer una comparación con todas las versiones respecto a la versión secuencial tomando en consideración la matriz más grande de 13760×13760. Primero, con la versión secuencial de LAPACK se obtuvo una rapidez de casi 55 veces con respecto a la versión secuencial con el algoritmo de G.J. Ahora comparando los casos paralelos del algoritmo LU con la versión secuencial de este mismo. La versión de ScaLAPACK con 32 (mallado no cuadrado) y con 25 (mallado cuadrado) se obtuvo una rapidez de 13 y 18.5 veces más rápido y finalmente para el caso paralelo con CULA se obtuvo una rapidez de más de 98 veces. Esto nos muestra que para invertir una matriz, la forma de programación más apropiada para reducir el tiempo de cómputo es usando librerías de CULA. Estos resultados también se muestran en la Fig. 5.

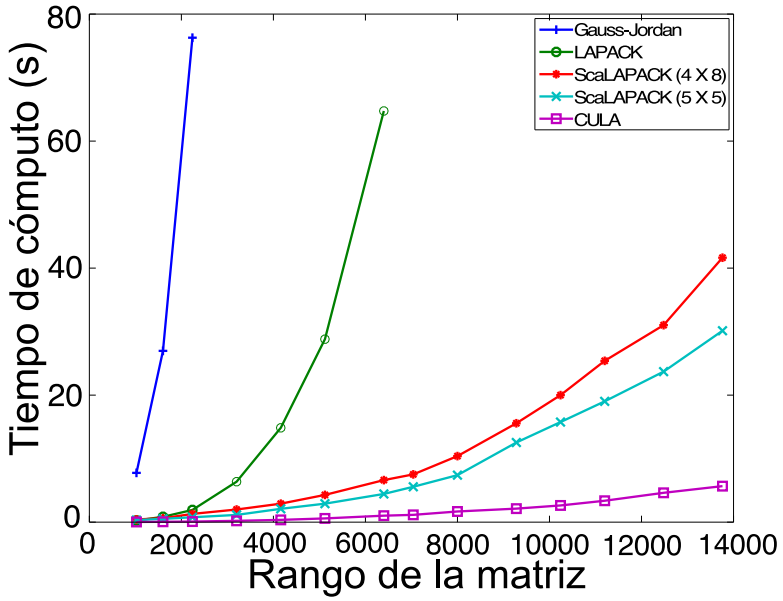

Figura 5. Tiempo de cómputo en segundos para la matriz inversa.

Capítulo IV

MÉTODO DE LA ECUACIÓN INTEGRAL

En este capítulo se describe una técnica rigurosa para modelar la interacción de la luz con una guía de onda de cristal fotónico (PCW) finita. La técnica se conoce como el Método de la Ecuación Integral (IEM). En particular, lo aplicaremos para estudiar la propagación de luz a través de una PCW calculando la reflectancia o transmitancia como función de la frecuencia.

El método numérico para calcular la respuesta óptica de una PCW está basado en la solución numérica de la ecuación de Helmholtz usando ecuaciones integrales (Mendoza-Suárez *et al.*, 2006; Pérez *et al.*, 2009; Mendoza-Suárez and Pérez-Aguilar, 2016). Este método parte del segundo teorema integral de Green permitiendo obtener un par de ecuaciones integrales acopladas que involucran, como incógnitas el modo del campo y su derivada normal evaluados en las fronteras o superficies involucradas. La discretización del sistema resulta en una ecuación matricial inhomogénea cuya solución determina las funciones fuente, con las que se puede calcular la reflectancia y transmitancia del sistema de estudio. Siendo esto, uno de los objetivos de este trabajo de investigación en el menor tiempo posible de cómputo numérico.

Este método implica ecuaciones independientes del tiempo, por lo que la evolución en el tiempo de los sistemas no es una preocupación, y funciona a lo largo de los contornos de las fronteras involucradas en la geometría que se manejan. Esto presenta algunas ventajas en comparación con otros métodos, ya que sólo toma en cuenta un número finito de puntos de muestreo a lo largo de los contornos del sistema, permitiendo una menor cantidad de recursos computacionales.

IV.1. Descripción del método de la ecuación integral

A continuación se presenta un planteamiento del IEM para calcular la reflectancia y transmitancia de una PCW bidimensional finita formada por dos paredes internas planas que encierran un arreglo de inclusiones cilíndricas circulares (Mendoza-Suárez and Pérez-Aguilar, 2016).

La geometría del sistema que se consideró para la teoría se muestra en la Fig. 6. Un sistema de q cuerpos invariantes a lo largo de z, es iluminado por una onda plana. El plano de incidencia es el $x-y$. La región 0 se caracteriza por un índice real de refracción $n_0 = \sqrt{\varepsilon_0(\omega)}$ y las regiones de la 1 a q se definen por las curvas Γ_j, y se caracterizan por el correspodiente índice de refracción n_j o alternativamente por la constante dieléctrica $\epsilon_j(\omega)$.

De la ecuación de onda que involucra campos armónicos en el tiempo, se obtiene la ecuación de Helmholtz

$$\nabla^2\psi(\mathbf{r}) + k^2\psi(\mathbf{r}) = 0, \tag{5}$$

donde $\psi(\mathbf{r})$ representa el campo electromagnético y k el vector de onda de propagación.

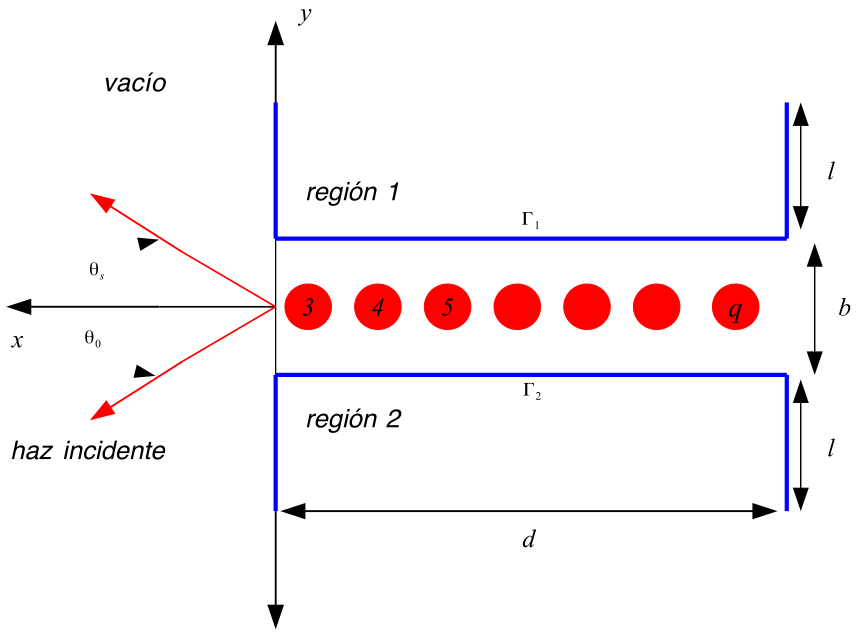

Figura 6. Descripción esquemática de una PCW, de ancho b y longitud d con inclusiones circulares, iluminado por una onda plana en la región del vacío. Las regiones 1 y 2 constituyen las placas paralelas conductoras, mientras que las regiones 3, 4 y así sucesivamente constituyen las inclusiones circulares conductoras.

Así podemos asociar una función de Green para el j-ésimo medio,

$$\nabla^2 G_j(\mathbf{r},\mathbf{r}') + k_j^2 G_j(\mathbf{r},\mathbf{r}') = 4\pi\delta(\mathbf{r}-\mathbf{r}'), \tag{6}$$

donde $G_j(\mathbf{r},\mathbf{r}')$ representa el propagador del campo debido a una fuente luz puntual que emite a la frecuencia ω en la posición $\mathbf{r}'$, correspondiente a cada medio. La $\delta(\mathbf{r}-\mathbf{r}')$ es la delta de Dirac. Una solución de la Ec. (6) está dada por

$$G_j(\mathbf{r},\mathbf{r}') = i\pi H_0^{(1)}(k_j|\mathbf{r}-\mathbf{r}'|), \tag{7}$$

siendo $H_0^{(1)}(x)$ la función de Hankel de primera especie y orden cero. Empleando el segundo teorema integral de Green (Jackson, 1999) aplicado a las funciones $\psi(\mathbf{r})$ y $G_j(\mathbf{r},\mathbf{r}')$ en cada región correspondiente al j-ésimo medio, obtenemos

$$\theta(\mathbf{r})\psi^{(j)}(\mathbf{r}) = \frac{1}{4\pi}\int_{\Gamma_j}\left[\frac{\partial G_j(\mathbf{r},\mathbf{r}')}{\partial n_j}\psi^{(j)}(\mathbf{r}) - G_j(\mathbf{r},\mathbf{r}')\frac{\partial\psi^{(j)}}{\partial n}(\mathbf{r})\right]ds_j, \tag{8}$$

donde $\theta(\mathbf{r})$ es la función de Heaveside,

$$\theta(\mathbf{r}) = \begin{cases} 1 \ \text{ si } \ \mathbf{r} \in S \\ \\ 0 \ \text{ si } \ \mathbf{r} \notin S, \end{cases} \tag{9}$$

siendo S una superficie cerrada. Los vectores $\mathbf{r}$ representa el punto de observación (donde se mide el campo EM) y $\mathbf{r}'$ es la variable de integración o la posición de las fuentes. En la Ec. (8) la superficie S_j está limitada por el contorno cerrado Γ_j correspondiente y la derivada normal $\partial/\partial n' \equiv \hat{\mathbf{n}}\cdot\nabla\hat{\mathbf{n}}$ va hacia afuera del contorno Γ_j.

De la Ec. (8), a cada uno de los términos los etiquetamos como

$$\frac{1}{4\pi}\oint_{\Gamma_j} G(\mathbf{r},\mathbf{r}')\frac{\partial\psi(\mathbf{r})}{\partial n'}ds' \cong \sum_{n=1}^{N_q} L^{(j)}_{mn(q)}\Phi^{(j)}_{n(q)}, \tag{10}$$

$$\frac{1}{4\pi}\oint_{\Gamma_j} \Psi(\mathbf{r})\frac{\partial G(\mathbf{r},\mathbf{r}')}{\partial n'}ds' \cong \sum_{n=1}^{N_q} N^{(j)}_{mn(q)}\Psi^{(j)}_{n(q)}, \tag{11}$$

donde $\Psi^{(j)}_{n(q)} = \psi_j(\mathbf{r}')|_{\mathbf{r}'=\mathbf{r}_{n(q)}}$ es el campo y $\Phi^{(j)}_{n(q)} = \frac{\partial\psi_j(\mathbf{r}')}{\partial n'_j}|_{\mathbf{r}'=\mathbf{r}_{n(q)}}$ su derivada normal, m representa el m-ésimo punto [coordenadas del observador $\mathbf{r} = (x_m, y_m)$] a lo largo del contorno Γ_j de cada región ($q = 1, 2, 3, ...$). Los elementos de matriz están dados por (Mendoza-Suárez *et al.*, 2006)

$$L^{(j)}_{mn} = [1-\delta_{mn}]\frac{i\Delta s}{4}H^1_o(k_j R_{mn}) + \left[\frac{i\Delta s}{4}H^1_o\left(k\frac{\Delta s}{2e}\right)\right]\delta_{mn}, \tag{12}$$

$$N^{(j)}_{mn} = [1-\delta_{mn}]\frac{i\Delta s}{4}k_j\hat{\mathbf{n}}_n\cdot\frac{\mathbf{R}_{mn}}{R_{mn}}H^{(1)}_1(k_j R_{mn}) + \left[\frac{1}{2} + \frac{\Delta s}{4\pi}\hat{\mathbf{n}}_n\cdot\hat{\mathbf{t}}'_n\right]\delta_{mn}, \tag{13}$$

siendo

$$\begin{aligned} \hat{\mathbf{n}}_n\cdot\mathbf{R}_{mn} &= -y'(s)(x_m - x_n) + x'(s)(y_m - y_n), \\ \hat{\mathbf{n}}_n\cdot\hat{\mathbf{t}}'_n &= x'(s)y''(s) - y'(s)x''(s), \\ R_{mn} &= \sqrt{(x_m - x_n)^2 + (y_m - y_n)^2}, \end{aligned} \tag{14}$$

y $H^{(1)}_1(x)$ es la función de Hankel de primera especie y primer orden.

El campo y potencia incidente

Una vez que obtenemos las fuentes, $\psi_n^{(1)}$ y $\phi_n^{(1)}$, ahora podemos calcular el campo en cualquier punto dentro de las regiones que constituyen el sistema mediante las mismas ecuaciones integrales. Como ejemplo ilustrativo, mostraremos el caso de una interfaz que separa dos medios (ver Fig. 7).

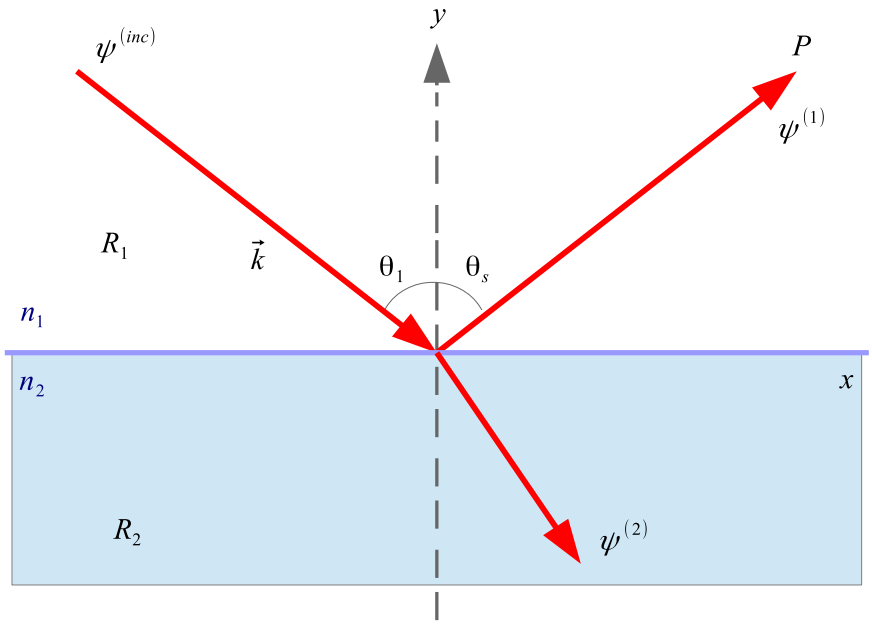

Figura 7. Campo esparcido en la región 1 (ψ^1) o en la región 2 (ψ^2).

Si $\mathbf{r}_m \in R_1$, con la ecuación

$$\psi_m^{(1)} = \sum_{n=1}^{N} \mathcal{L}_{mn}^{(1)} \phi_n^{(1)} - \sum_{n=1}^{N} \mathcal{N}_{mn}^{(1)} \psi_n^{(1)} - \psi_m^{inc(1)} \tag{15}$$

o si $\mathbf{r}_m \in R_2$, con

$$\psi_m^{(2)} = \sum_{n=1}^{N} \mathcal{L}_{mn}^{(2)} \phi_n^{(2)} - \sum_{n=1}^{N} \mathcal{N}_{mn}^{(2)} \psi_n^{(2)}, \tag{16}$$

podemos obtener el campo en ambas regiones.

De la Fig. 7, tenemos

$$k_{1x} = k_1 \sin\theta_1 = \frac{2\pi}{\lambda} \sin\theta_1, \tag{17}$$

$$k_{1y} = k_1 \cos\theta_1 = \frac{2\pi}{\lambda}\cos\theta_1, \tag{18}$$

entonces, el campo incidente es

$$\psi_{inc}(\mathbf{r}) = \psi_0 e^{i\frac{2\pi}{\lambda}n_1[x\sin\theta_1 - y\cos\theta_1]} \tag{19}$$

o bien,

$$\psi_{inc}(\mathbf{r}) = \psi_0 e^{i\mathbf{k}\cdot\mathbf{r}}, \tag{20}$$

donde ψ_0 es una constante con unidades adecuadas, $\mathbf{k}$ es el vector de onda y $\mathbf{r}$ la posición de cada punto sobre el cual la onda incide.

Para el campo lejano utilizaremos las expresiones

$$\mathcal{L}_{mn} \simeq \frac{i\Delta s}{4} H_0^{(1)}(kR_{mn}) \tag{21}$$

y

$$\mathcal{N}_{mn} \simeq \frac{ik\Delta s}{4}\widehat{\mathbf{n}}_n \cdot \mathbf{R}_{mn} H_1^{(1)}(kR_{mn}), \tag{22}$$

que están definidas en las Ecs. (12) y (13), respectivamente. Aquí $\mathbf{R}_{mn} = \mathbf{r}_m - \mathbf{r}_n$ donde $\mathbf{r}_m = r_m \sin\theta_s \hat{\imath} + r_m \cos\theta_s \hat{\jmath}$ y $\mathbf{r}_n = x_n\hat{\imath} + y_n\hat{\jmath}$.

Tomando en cuenta que el observador $\mathbf{r}_m$ se encuentra muy lejos del sistema, $\mathbf{r}_m \gg \mathbf{r}_n$ y $\mathbf{r}_m \gg \lambda$. Así, necesitamos una aproximación asintótica de la función de Hankel $H_0^{(1)}$ para argumentos grandes,

$$H_0^{(1)}(x) \approx \sqrt{\frac{2}{\pi x}} e^{-i\frac{\pi}{4}} e^{ix}, \tag{23}$$

siendo $x = kR_{mn}$ y

$$\begin{aligned} R_{mn} &= \sqrt{(\mathbf{r}_m - \mathbf{r}_n)\cdot(\mathbf{r}_m - \mathbf{r}_n)} = \sqrt{\mathbf{r}_m^2 - 2\mathbf{r}_m\cdot\mathbf{r}_n + \mathbf{r}_n^2} \\ &= \mathbf{r}_m\sqrt{1 - \tfrac{2\mathbf{r}_m\cdot\mathbf{r}_n}{\mathbf{r}_m^2} + \tfrac{\mathbf{r}_n^2}{\mathbf{r}_m^2}} \approx \mathbf{r}_m\sqrt{1 - \tfrac{2\mathbf{r}_m\cdot\mathbf{r}_n}{\mathbf{r}_m^2}} \\ &\approx \mathbf{r}_m\left(1 - \tfrac{\mathbf{r}_m\cdot\mathbf{r}_n}{\mathbf{r}_m^2}\right) = \mathbf{r}_m - \tfrac{\mathbf{r}_m\cdot\mathbf{r}_n}{\mathbf{r}_m}. \end{aligned} \tag{24}$$

Por otro lado

$$\sqrt{R_{mn}} \approx \sqrt{\mathbf{r}_m \left(1 - \frac{\mathbf{r}_m \cdot \mathbf{r}_n}{\mathbf{r}_m^2}\right)} = \sqrt{\mathbf{r}_m}\sqrt{1 - \frac{\mathbf{r}_m \cdot \mathbf{r}_n}{\mathbf{r}_m^2}} \approx \sqrt{\mathbf{r}_m}. \tag{25}$$

Como $\mathbf{r}_m \cdot \mathbf{r}_n = r_m x_n \sin\theta_s + r_m y_n \cos\theta_s$ donde θ_s es el ángulo de esparcimiento de la luz. Así que, sustituyendo la Ec. (23) con las aproximaciones (Ecs. (24) y (25)) en la Ec. (21) tenemos que

$$\begin{aligned} \mathcal{L}_{mn} &\approx \left(\frac{i\Delta s}{4}\sqrt{\frac{2}{\pi k}}e^{-i\frac{\pi}{4}}\right)\frac{e^{ikr_m}}{\sqrt{\mathbf{r}_m}}e^{-\frac{ik}{r_m}(r_m x_n \sin\theta_s + r_m y_n \cos\theta_s)} \\ &\approx \left(\frac{i\Delta s}{4}\sqrt{\frac{2}{\pi k}}e^{-i\frac{\pi}{4}}\right)\frac{e^{ikr_m}}{\sqrt{\mathbf{r}_m}}e^{-ik(x_n \sin\theta_s + y_n \cos\theta_s)}. \end{aligned} \tag{26}$$

Si llamamos a

$$q = \frac{i\Delta s}{4}\sqrt{\frac{2}{\pi k}}e^{-i\frac{\pi}{4}}, \tag{27}$$

entonces,

$$\mathcal{L}_{mn} \approx q\frac{e^{ikr_m}}{\sqrt{\mathbf{r}_m}}e^{-ik(x_n \sin\theta_s + y_n \cos\theta_s)}. \tag{28}$$

Similarmente, para la función $H_1^{(1)}(x)$ para argumentos grandes,

$$H_1^{(1)}(x) \approx -i\sqrt{\frac{2}{\pi x}}e^{-i\frac{\pi}{4}}e^{ix} = -iH_0^{(1)}, \tag{29}$$

tenemos, que

$$\widehat{\mathbf{R}}_{mn} = \frac{\mathbf{R}_{mn}}{R_{mn}} \approx \frac{x_m}{r_m}\hat{\imath} + \frac{y_m}{r_m}\hat{\jmath}, \tag{30}$$

donde $x_m = r_m \cos\theta_s$ y $y_m = r_m \sin\theta_s$, por lo que $\widehat{\mathbf{R}}_{mn} \approx \cos\theta_s\hat{\imath} + \sin\theta_s\hat{\jmath}$. Sustituyendo la Ecs. (29) y (30) en la Ec. (22) tenemos

$$\mathcal{N}_{mn} \approx q\frac{e^{ikr_m}}{\sqrt{r_m}}e^{-ik(x_n \sin\theta_s + y_n \cos\theta_s)}(-ik\widehat{\mathbf{n}}_n \cdot \mathbf{R}_{mn}). \tag{31}$$

Por lo tanto, el campo reflejado (Ec. (15)),

$$\psi_m^{(1)} = \sum_{n=1}^{N}\mathcal{L}_{mn}^{(1)}\phi_n^{(1)} - \sum_{n=1}^{N}\mathcal{N}_{mn}^{(1)}\psi_n^{(1)}, \tag{32}$$

se puede reescribir como

$$\psi_m^{(1)} = q\frac{e^{ik_1 r_m}}{\sqrt{r_m}} \sum_{n=1}^{N} \left[\phi_n^{(1)} + ik_1(n_x \cos\theta_s + n_y \sin\theta_s)\psi_n^{(1)}\right] e^{-ik_1(x_n \sin\theta_s + y_n \cos\theta_s)}. \quad (33)$$

En esta expresión se omitió el término del campo incidente.

Similarmente para a región de transmisión donde $\mathbf{r}_m \in R_2$, tenemos

$$\psi_m^{(2)} = q\frac{e^{ik_2 r_m}}{\sqrt{r_m}} \sum_{n=1}^{N} \left[\phi_n^{(2)} + ik_2(n_x \cos\theta_s + n_y \sin\theta_s)\psi_n^{(2)}\right] e^{-ik_2(x_n \sin\theta_s + y_n \cos\theta_s)}. \quad (34)$$

Ahora vamos a calcular la potencia total incidente. Para ello, tenemos que la onda incidente que se propaga depende de la orientación del campo electromagnético. Primeramente consideraremos el caso de la polarización TE, cuyos campos eléctrico y magnético son

$$E(\mathbf{r}) = \widehat{\mathbf{k}} E_z(x, y) \quad (35)$$

y

$$H(\mathbf{r}) = \hat{\imath} H_x(x, y) + \hat{\jmath} H_y(x, y). \quad (36)$$

Para calcular el coeficiente de reflexión diferencial, el cual representa la fracción de energía incidente sobre una superficie que es esparcida por unidad de ángulo, se necesita calcular el flujo incidente total y el flujo esparcido total. Para esto, se emplea el vector de Poynting $\mathbf{S}$, que proporciona la dirección y magnitud del flujo de energía por unidad de tiempo. Empleando la notación compleja se tiene que

$$\mathbf{S} = \mathbf{E} \times \mathbf{H}^*. \quad (37)$$

El promedio temporal del vector de Poynting de una onda representa la potencia media que la onda transporta por unidad de área transversal a la propagación y se conoce como densidad de potencia o irradiancia,

$$\begin{aligned} P &= \frac{1}{2}\mathrm{Re}(\mathbf{E} \times \mathbf{H}^*) = \tfrac{1}{2}\mathrm{Re}\left[\widehat{\mathbf{k}} E_z \times (\hat{\imath} H_x^* + \hat{\jmath} H_y^*)\right] \\ &= \frac{1}{2}\mathrm{Re}(\hat{\jmath} E_z H_x^* - \hat{\imath} E_z H_y^*). \end{aligned} \quad (38)$$

Como nada más nos interesa el flujo de energía a tráves de la interfaz, por lo que solamente contribuirá un sólo término

$$P = \frac{1}{2} Re(E_z H_x^*). \tag{39}$$

Como $\frac{\partial E_z}{\partial y} = i\omega\mu H_x$, entonces

$$P = \frac{1}{2} Re\left\{ E_z \left(\frac{1}{i\omega\mu_1} \frac{\partial E_z}{\partial y} \right)^* \right\}. \tag{40}$$

Si el campo incidente en la superficie es

$$E_z^{inc}(x, y) = e^{ik_1(x_n \sin\theta_1 + y_n \cos\theta_1)}, \tag{41}$$

entonces,

$$\frac{\partial E_z^{inc}}{\partial y} = -ik_1 \cos\theta_1 E_z^{inc}. \tag{42}$$

Así que

$$\begin{aligned} P^{inc} &= \frac{1}{2} Re\left\{ E_z^{inc} \left(\frac{-ik_1 \cos\theta_1}{i\omega\mu_1} E_z^{inc} \right)^* \right\} \\ &= -\frac{k_1 \cos\theta_1}{2\omega\mu_1} \mid E_z^{inc} \mid^2 . \end{aligned} \tag{43}$$

Para obtener la potencia incidente total es necesario integrar sobre un área. El haz está confinado a lo largo del plano $x - y$ con límites de integración desde $-L/2$ hasta $L/2$. Entonces, la potencia total incidente en un área de $L_x L_z$ de la interfaz será

$$\begin{aligned} P_{total}^{inc} &= \int_{-\frac{L_x}{2}}^{\frac{L_x}{2}} \int_{-\frac{L_y}{2}}^{\frac{L_y}{2}} P^{inc} dx dz = -\frac{k_1 \cos\theta_1}{2\omega\mu_1} \left| E_z^{inc} \right|^2 L_x L_z \\ &= -\frac{n_1 \frac{\omega}{c} \cos\theta_1}{2\omega\mu_1} \mid E_z^{inc} \mid^2 L_x L_z \\ &= -\frac{1}{\mu_1 c} \frac{n_1}{2} \cos\theta_1 \mid E_z^{inc} \mid^2 L_x L_z. \end{aligned} \tag{44}$$

Si $\mid E_z^{inc} \mid^2 = 1$ entonces,

$$\begin{aligned} P_{total}^{inc} &= -\frac{1}{\mu_1 c} \frac{n_1}{2} \cos\theta_1 L_x L_z \\ &= -\eta \frac{n_1}{2} \cos\theta_1 L_x L_z, \end{aligned} \tag{45}$$

donde η es la admitancia óptica en el vacío dada por $\eta = \sqrt{\varepsilon_1/\mu_1}$.

Para el caso de polarización TM se obtiene la misma expresión.

El campo y potencia esparcida

Si el campo esparcido tiene la forma

$$\psi_j(\mathbf{r}) = \int_{\Gamma_j} \left[L_{mn}^{(j)} \phi_n^{(j)} - N_{mn}^{(j)} \psi_n^{(j)} \right] ds', \tag{46}$$

y empleando las funciones de Hankel en el límite asintótico bajo la suposición de que $\mid \mathbf{r} \mid \gg \mid \mathbf{R} \mid$, entonces la Ec. (33) se puede reescribir como:

$$\begin{aligned} \psi_m^{(1)}(\mathbf{r}) &= \left[\tfrac{i}{4} \sqrt{\frac{2}{\pi k_1}} e^{-i\frac{\pi}{4}} \frac{e^{ik_1 r_m}}{\sqrt{r_m}} \right] \Delta s \times \\ & \quad \textstyle\sum_{n=1}^{N} \left[ik_1 (y_n' \sin\theta_s - x_n' \cos\theta_s) \psi_n^{(1)} - \phi_n^{(1)} \right] e^{-ik_1 (x_n \sin\theta_1 + y_n \cos\theta_1)}. \end{aligned} \tag{47}$$

Por lo tanto,

$$\begin{aligned} \left| \psi_m^{(1)}(\mathbf{r}) \right|^2 &= \frac{1}{8\pi k_1 r_m} \times \\ & \quad \left| \Delta s \textstyle\sum_{n=1}^{N} \left[ik_1 (y_n' \sin\theta_s - x_n' \cos\theta_s) \psi_n^{(1)} - \phi_n^{(1)} \right] e^{-ik_1 (x_n \sin\theta_1 + y_n \cos\theta_1)} \right|^2. \end{aligned} \tag{48}$$

Definiendo la sección eficaz de esparcimiento de un material como

$$\sigma_R(\theta_s) = \Delta s \sum_{n=1}^{N} \left[ik_1 (y_n' \sin\theta_s - x_n' \cos\theta_s) \psi_n^{(1)} - \phi_n^{(1)} \right] e^{-ik_1 (x_n \sin\theta_1 + y_n \cos\theta_1)}, \tag{49}$$

se puede observar que la Ec. (47) es función de dos variables independientes, $\mathbf{r}$ y θ_s, por lo que podemos decir que

$$\mathbf{E}_s(\mathbf{r}, \theta_s) = \boldsymbol{I}_s(\mathbf{r}) \sigma_s(\theta_s). \tag{50}$$

Utilizando un procedimiento similar al caso de la potencia incidente, tenemos que la potencia esparcida en reflexión está dada por

$$P_R = \int (\mathbf{E}_s \times \mathbf{H}_s) \cdot \widehat{\mathbf{n}}_2 da, \tag{51}$$

donde, es posible escribir

$$\begin{aligned} \mathbf{E}_s \times \mathbf{H}_s &= \frac{1}{\mu_1 c} \mid \mathbf{E}_s \mid^2 \\ &= \frac{1}{\mu_1 c} \mid \boldsymbol{I}_s \mid^2 \mid \sigma_s \mid^2 . \end{aligned} \tag{52}$$

Así la potencia esparcida en reflexión se puede expresar como

$$\begin{aligned} P_R &= \frac{\eta n_1}{2}\int_s \left|\psi_m^{(1)}\right|^2 da = \frac{\eta n_1}{2}\frac{1}{8\pi k_1}\int_0^{L_z}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{1}{\mathbf{r}_m}\left|\sigma_R(\theta_s)\right|^2 dz r_m d\theta_s \\ &\approx \frac{\eta n_1}{2}\frac{L_z}{8\pi k_1}\sum_{i=1}^{N_{\theta_s}}\left|\sigma_R(\theta_s)\right|^2 \Delta\theta. \end{aligned} \tag{53}$$

Dividiendo la Ec. (54) entre la Ec. (46) obtenemos la reflectancia diferencial

$$dR(\theta_s) \equiv \frac{P_R}{P^{inc}} = \frac{\frac{\eta n_1}{2}\frac{1}{8\pi k_1}\left|\sigma_R(\theta_s)\right|^2 d\theta_s}{\frac{\eta n_1}{2}L_x L_z\cos\theta_1} = \frac{1}{8\pi k_1 L_x\cos\theta_1}\left|\sigma_R(\theta_s)\right|^2 d\theta_s, \tag{54}$$

que integrando en el intervalo de $[-\pi/2, \pi/2]$ obtenemos la reflectancia,

$$R(\theta_s) = \int_{-\pi/2}^{\pi/2} dR(\theta_s) = \int_{-\pi/2}^{\pi/2}\frac{\left|\sigma_R(\theta_s)\right|^2}{8\pi k_1 L_x\cos\theta_1} d\theta_s. \tag{55}$$

Por otro lado, la potencia esparcida en transmisión está dada por

$$P_T = \frac{\eta_2}{2}\frac{L_z}{8\pi k_2}\int_{-\pi/2}^{\pi/2}\left|\sigma_T(\theta_s)\right|^2 d\theta_s, \tag{56}$$

que análogamente nos lleva a la diferencial de transmitancia $dT(\theta_s)$,

$$dT(\theta_s) \equiv \frac{P_T}{P^{inc}} = \frac{\frac{\eta n_2}{2}\frac{L_z}{8\pi k_2}\left|\sigma_T(\theta_s)\right|^2 d\theta_s}{\frac{\eta n_1}{2}L_x L_z\cos\theta_1} = \frac{n_2}{8 n_1\pi k_2 L_x\cos\theta_1}\left|\sigma_T(\theta_s)\right|^2 d\theta_s, \tag{57}$$

y ahora integrando sobre el intervalo de $[\pi/2, -\pi/2]$, obtenemos la transmitancia como

$$T(\theta_s) = \int_{-\pi/2}^{\pi/2} dT(\theta_s) = \frac{n_2}{8 n_1\pi k_2 L_x\cos\theta_1}\int_{-\pi/2}^{\pi/2}\left|\sigma_T(\theta_s)\right|^2 d\theta_s. \tag{58}$$

La conservación de la energía está dada por $R + T = 1$.

Verificación del método numérico

Por simplicidad en este trabajo únicamente se presenta el caso de la polarización TE; es decir $\psi^{(0)} = 0$. En la Fig. 8(a) se ilustra el perfil de una PCW de longitud d = 10 periodos y ancho $b = \pi$ con inclusiones circulares de radio r = 0.177. Las Figs. 8(b),

(c) y (d) muestran la reflectacia, la transmitancia y la conservación de la energía de una PCW como función de la frecuencia reducida ω_r, respectivamente.

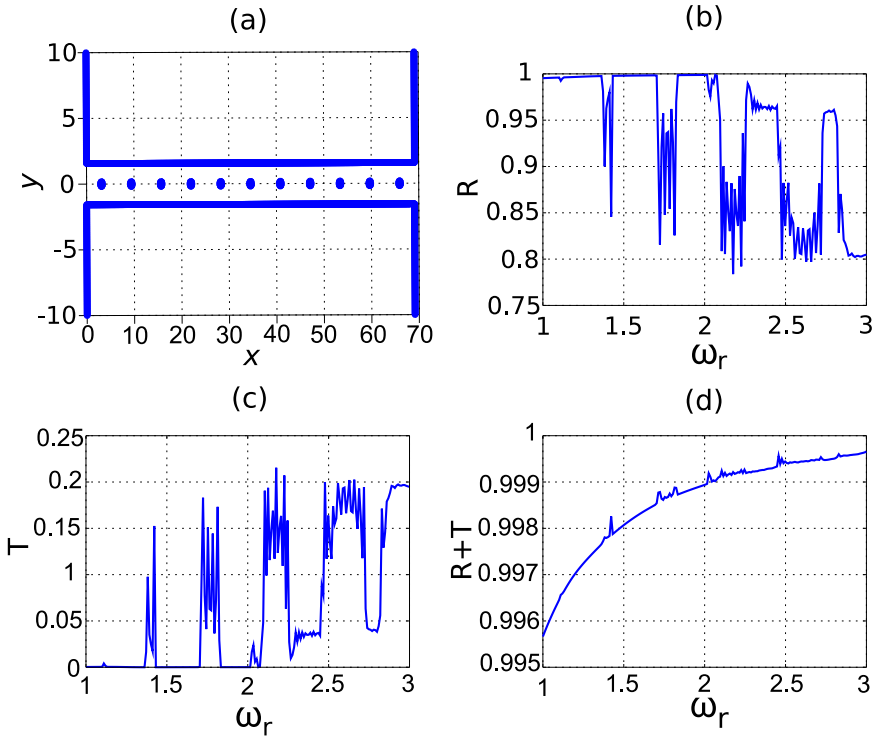

Figura 8. (a) Perfil de la guía de ondas de longitud $d = 10$ periodos y de ancho $b = \pi$ con inclusiones circulares de radio $r = 0.177$. (b) Reflectancia , (c) Transmitancia y (d) Balance de energía de una PCW como función de la frecuencia reducida ω_r.

En la Fig. 8(d) se ilustra el balance de la energía mostrando un error numérico al 1%. Esto nos indica que el método integral que hemos empleado es muy confiable.

En el siguiente capítulo aplicaremos el IEM a una PCW finita para calcular su reflectancia o transmitancia como función de la frecuencia de la onda electromagnética incidente.

Capítulo V

RESPUESTA ÓPTICA DE UNA PCW

En este capítulo se presenta los resultados numéricos obtenidos de la respuesta óptica de una guía de ondas de cristal fotónico (PCW) bidimensional finita aplicando el método de la ecuación integral, descrito en el capítulo anterior, en sus distintas formas de programación en paralelo.

V.1. Algoritmos de programación en paralelo

En esta sección mostraremos las diferentes formas de programación en paralelo del método integral, usando tanto los procesadores como la tarjeta gráfica.

V.1.1. Algoritmo de MPI

En esta sección se exhibe la implementación de un programa en FORTRAN que utiliza la librería de MPI para implementar el Método Integral en forma paralela y poder estudiar la propagación de la luz a través de una PCW. En el programa 3 se muestra el algoritmo que se utilizó para dicho fin. Para conocer el funcionamiento del programa 3 revisar la Ref. (Pérez Hernández, 2015, 74–87). Por tal motivo, no se dará una

explicación del algoritmo.

****

Programa 3. Método Integral para una PCW usando MPI

****

```
PROGRAM PCW_MPI
IMPLICIT NONE
INCLUDE 'MPIF.H'

INTEGER, PARAMETER :: MASTER = 0
INTEGER, PARAMETER :: SIZEARRAY = 192
INTEGER :: NUMTASKS, TASKID, LEN, IERR, NPROCES, OFFSET
INTEGER :: PARTNER, MESSAGE, STATUS (MPI_STATUS_SIZE)

CHARACTER ( MPI_MAX_PROCESSOR_NAME ) :: HOSTNAME

INTEGER :: N, INICIO, EPP, EXTRA, J, COLS, CHUNKSIZE, ERROR
INTEGER NPT, NANG, NPA, NPB, NFF, OP, NW, NPCILI, NPER

REAL PI, TINID, DTHET, DELTA, NWA, NWB, G, LAMBD, B, PHASE, PERIOD,
REAL F, R, P

PARAMETER ( PI=3.14159265358979323846264338327 95 )
PARAMETER ( B = 1.0*PI, PHASE = 0.5*PI, PERIOD = 2.0*PI)
PARAMETER ( LAMBD = 1.0, NFF = 181, NWA = 28.0, NPER = 11, NWB = 500)
PARAMETER ( DELTA = 1.0/15.0, NANG = 1, NPA = NINT ( NWA / DELTA ) )
PARAMETER ( R = SQRT ( F * PERIOD * B / PI), F = 0.005 )
PARAMETER ( P = 2.0*PI*R, NPCILI = INT ( P/DELTA ) )
PARAMETER ( NPB = NINT ( NWB/DELTA), NPT = 2*(2*NPA+NPB) + NPER*NPCILI)

INTEGER K0, I, ID, POL, FDO, IWA (NPT), IER

REAL RAD, FK0, HSLITW, SLITW, HLARGE, LARGE, DW, D, A1, A2
REAL THETA0 ( NANG ), NORFAC (NANG ), SF (NPT ), SG(NPT ), WNMIN, WNMAX
REAL DSF ( NPT ), DSG ( NPT ), DDSF ( NPT ), DDSG ( NPT )
REAL CONSUP, CONSDOWN, ENERBAL, REFLPOW, TRANSPOW

COMPLEX EI ( NPT, NANG ), ME ( NPT, NPT ), AK ( NFF, NANG )

REAL, DIMENSION ( SIZEARRAY ) :: WNUM1, REFLACT, TRANS, ENERG, FRECUENCY
& SALIDA1, SALIDA2, SALIDA3, SALIDA4
REAL, DIMENSION ( : ), ALLOCATABLE :: WNUM
```

```
CALL MPI_INIT( IERR )
CALL MPI_COMM_SIZE( MPI_COMM_WORLD, NUMTASKS, IERR )

NPROCES = NUMTASKS -1

CALL MPI_COMM_RANK( MPI_COMM_WORLD, TASKID, IERR)
CALL MPI_GET_PROCESSOR_NAME( HOSTNAME, LEN, IERR)

CHUNKSIZE = SIZEARRAY/NUMTASKS

ALLOCATE( WNUM ( CHUNKSIZE ), STAT = ERROR)

IF ( CHUNKSIZE*NUMTASKS .NE. SIZEARRAY ) THEN
  CALL MPI_Abort ( MPI_COMM_WORLD, IERR )
  STOP
END IF

WNMIN = 1.0
WNMAX = 3.0
NW = 192
OP = 4
POL = 2
FDO = 2

PRINT*, 'SIZE_ARRAY:', NPT

RAD = PO/180
D = NWB
SLITW = B
HSLITW = 0.5*B
HLARGE = NWA + HSLITW
LARGE = 2.0*HLARGE

A1=0.3*B
A2=A1
WNUM = WNMIN
G = 2.0*NWA/5.0
DTHET = 1.0
TINID = 0.0

DO KO = 1, NANG
  FKO = FLOAT ( KO )
  THETAO ( KO ) = TINID + FLOAT ( KO-1 )*DTHET
```

```
  THETA0 ( K0 ) = RAD*THETA0 ( K0 )
END DO

DW = 1.0/FLOAT ( NW - 1)

IF ( TASKID == MASTER ) THEN
  DO I = 1, NW
    WNUM1 ( I ) = WNMIN + (WNMAX - WNMIN )*DW*FLOAT ( I-1 )
  END DO
END IF

CALL MPI_SCATTER( WNUM1, CHUNKSIZE, MPI_REAL, WNUM, CHUNKSIZE,
&               MPI_REAL, 0, MPI_COMM_WORLD, IERR )

DO ID = 1, CHUNKSIZE

  CALL NORTE ( NORFAC, WNUM ( ID ), G, THETA0, NANG, FDO, LARGE, SLITW )
  CALL PERFILES ( SF, SG, DSF, DSG, DDSF, DDSG, NWA, NWB, NPT, NPA, NPB,
&               HSLITW, HLARGE, DELTA, A1, A2, PERIOD, PHASE, PI, B, R,
&               NPCILI, NPER )
  CALL EINUMNR ( EI, NPT, NANG, SF, SG, THETA0, WNUM ( ID ), G, D, FDO )
  CALL MATM ( ME, NPT, SF, SG, DSF, DSG, DDSF, DDSG, DELTA, WNUM ( ID ),
&               POL )
  CALL CGESV ( NPT, NANG, ME, NPT, IWA, EI, NPT, IER)
  CALL AKAN ( AK, NFF, NANG, LARGE, THETA0, WNUM ( ID ), G, D, FDO )
  CALL ENERGY ( NPT, NANG, FDO, POL, G, THETA0, WNUM ( ID ), AK, SLITW,
&               LARGE, SF, SG, DSF, DSG, DELTA, EI, NORFAC, CONSUP,
&               CONSDOWN, ENERBAL, DTHET, REFLPOW, TRANSPOW, NFF )

  WRITE ( *, * ) WNUM ( ID ), REFLPOW, TRANSPOW, ENERBAL, TASKID

  FRECYEBCY ( ID ) = WNUM1 ( ID )
  REFLECT ( ID ) = REFLPOW
  TRANS ( ID ) = TRANSPOW
  ENERG ( ID ) = ENERBAL

END DO

CALL MPI_GATHER ( WNUM, CHUNKSIZE, MPI_REAL, SALIDA1, CHUNKSIZE,
&               MPI_REAL, 0, MPI_COMM_WORLD, IERR )
CALL MPI_GATHER(REFLECT, CHUNKSIZE, MPI_REAL, SALIDA2, CHUNKSIZE,
&               MPI_REAL, 0, MPI_COMM_WORLD, IERR )
ALL MPI_GATHER(TRANS, CHUNKSIZE, MPI_REAL, SALIDA3, CHUNKSIZE,
&               MPI_REAL, 0, MPI_COMM_WORLD, IERR )
```

```
CALLMPI_GATHER(ENERG, CHUNKSIZE, MPI_REAL, SALIDA4, CHUNKSIZE,
&               MPI_REAL, 0, MPI_COMM_WORLD, IERR )}

IF ( TASKID == MASTER ) THEN
  DO I = 1, SIZEARRAY
     WRITE ( *, * ) SALIDA1 ( I ), SALIDA2 ( I ), SALIDA3 ( I ),SALIDA4 ( I )
  END DO
END IF

CALL MPI_FINALIZE ( IERR )
END
```

****

V.1.2. Algoritmo SCALAPACK

En esta sección se muestra la implementación de un programa en FORTRAN que utiliza la biblioteca de SCALAPACK aplicado al mismo Método de la Ecuación Integral en forma paralela usando los procesadores de la CPU. En el programa 4 se muestra el algoritmo que se implementó.

****

Programa 4. Método Integral para una PCW usando SCALAPACK

****

```
PROGRAM PCW_SCALAPACK
IMPLICIT NONE

-----------------------------------------------------------------------
```

SECCIÓN 1.- Inicialización de variables, arreglos y declaración de rutinas externas

```
-----------------------------------------------------------------------

INTEGER       NPT, NANG, NPA, NPB, NFF, OP, NW, NPCILI, NPER
INTEGER       DLEN_, IA, JA, IB, JB, M, N, MB, NB, RSRC, CSRC, MXLLDA,
$             MXLLDB, NRHS, NBRHS, MXLOCR, MXLOCC, MXRHSC, RE, CO

REAL          PI, TINID, DTHET, DELTA, NWA, NWB, G, LAMBD, B, PHASE,
$             PERIOD, F, R, P
```

```
PARAMETER     ( PI = 3.14159265358979323846264338327795 )
PARAMETER     ( B = 1.0*PI, PHASE = 0.5*PI, PERIOD = 2.0*PI )
PARAMETER     ( LAMBD = 1.0, NFF = 181, NWA 28.0, NPER = 11, NWB = 600 )
PARAMETER     ( DELTA = 1.0/15.0, NANG = 1, NPA = NINT (NWA/DELTA )
PARAMETER     ( F = 0.005, R = SQRT ( F*PERIOD*B/PI ), P = 2.0*PI*R )
PARAMETER     ( NPCILI = INT( P/DELTA ), NPB = NINT ( NWB/DELTA ),
PARAMETER     ( NPT = 2*( 2*NPA+NPB )+NPER*NPCILI )

PARAMETER     ( DLEN_ = 9, IA = 1, JA = 1, IB = 1, JB = 1,RE = 5, CO=5,
$             M = npt, N = M, MB = 64, NB = 64, RSRC = 0, CSRC = 0,
$             MXLLDA = ( N/RE )+( MB-1), MXLLDB = ( N/RE )+( MB-1 ),
$             NRHS= 1, NBRHS = 1, MXLOCR = ( N/re )+( MB-1 ),
$             MXLOCC = ( N/CO )+( NB-1 ), MXRHSC = 1)

INTEGER       KO, I, ID, POL, FDO, IWA ( NPT ), EIR

REAL          RAD, WNUM, FKO, HSLITW, SLITW, HLARGEM DW, D, TRANSPOW
REAL          WNMIN, WNMAX, CONSUP, CONSDOWN, ENERBAL, REFLPOW, A1, A2
REAL          THETAO ( NANG ), NORFAC ( NANG ), SF ( NPT ), SG ( NPT )
REAL          DSF ( NPT ), DSG ( NPT ), DDSF ( NPT ), DDSG ( NPT )

COMPLEX       EI ( NPT, NANG ),  AK ( NFF, NANG )

INTEGER       ICTXT, INFO, MYCOL, MYROW, NPCOL, NPROW
INTEGER       DESCA ( DLEN_ ), DESCB( DLEN_ ), IPIV ( MXLOCR+NB ),
$             TC ( CO, N ), TR ( RE, N )

COMPLEX       A ( MXLLDA, MXLOCC ), BB( MXLLDB, MXRHSC )

EXTERNAL      BLACS_EXIT, BLACS_GRIDEXIT, BLACS_GRIDINFO, DESCINIT,
$             MATINIT, PCGESV, SL_INIT, DISTRIBUCIONCICLICA, ORDENA,
$             NORTE, PERFILES, EINUMNR, MATM, AKAN, ENERGY
DATA          NPROW / RE / , NPCOL / CO /

CALL          SL_INIT ( ICTXT, NPROW, NPCOL )
CALL          BLACS_GRIDINFO ( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
CALL          DISTRIBUCIONCICLICA ( TC, NPCOL, N, NB )
CALL          DISTRIBUCIONCICLICA ( TR, NPROW, N, NB )

WNMIN = 1.0
WNMAX = 3.0
NW = 200
OP = 4
```

```
POL = 2
FDO = 2
```

SECCIÓN 2.- Parámetros generales y parámetros de ángulo de incidencia

---

```
IF ( ( MYROW.EQ.0 ) .AND. ( MYCOL.EQ.0 ) ) THEN
    print*, 'RADIO = ', R, 'NPCILI = ', NPCILI, 'NPT', NPT
END IF

RAD = PI/180
D = NWB
SLITW = B
HSLITW = 0.5*B
HLARGE = NWA+HSLITW
LARGE = 2.0*HLARGE
A1 = 0.0*B
A2 = A1
WNUM = WNMIN
G = 2.0*NWA/5.0
DTHET = 1.0
TINID = 0.0

DO K0 = 1, NANG
    FK0 = FLOAT ( K0 )
    THETA0 ( K0 ) = TINID + FLOAT ( K0 -1 )*DTHET
    THETA0 ( K0 ) = RAD*THETA0 ( K0 )
END DO

DW = 1.0/FLOAT ( NW -1 )
```

SECCIÓN 3.- Ciclo de variación de frecuencias y llamado a las rutinas encargadas de dar vida a la simulación

---

```
DO ID = 1, NW
    WNUM = WNMIN+ ( WNMAX-WNMIN )*DW*FLOAT ( ID-1 )
    CALL  NORTE ( NORFAC, WNUM, G, THETA0, NANG, FDO, LARGE, SLITW )
```

```
    CALL  PERFILES ( SF, SG, DSF, DSG, DDSF, DDSG, NWA, NWB, NPT, NPA,
$         NPB, HSLITW, HLARGE, DELTA, A1, A2, PERIOD, PHASE, PI, B, R,
$         NPCILI, NPER )
    CALL  AKAN ( AK, NFF, NANG, LARGE, THETA0, WNUM, G, D, FDO )
    CALL  DESCINIT ( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA,
$         INFO )
    CALL  DESCINIT( DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT,
&         MXLLDB, INFO )
    CALL  MATINIT( A, DESCA, BB, DESCB, N, NPCOL, NPROW, TC, TR, NPT,
&         NANG, SF, SG, WNUM, G, D, DSF, DSG, DDSF, DDSG, DELTA )
    CALL  PCGESV( N, NRHS, A, IA, JA, DESCA, IPIV, BB, IB, JB,
&         DESCB, INFO )
```

SECCIÓN 4.- Parte encargada de organizar la solución

```
    IF ( ( MYROW.EQ.0 ) .AND. ( MYCOL.EQ.0 ) ) THEN
        DO I = 0, NPROW-1
            IF ( I .EQ. 0 )THEN
                CALL ORDENA( EI, BB, I, 0, N, NRHS, MXLLDB, MXRHSC, TR,
&                    NPROW)
            ELSE
                CALL CGERV2D ( ICTXT, MXLLDB, MXRHSC, BB, MXLLDB, I, 0 )
                CALL ORDENA ( EI, BB, I, 0, N, 1, MXLLDB, MXRHSC, TR,
&                    NPROW )
            END IF
        END DO
    END IF

    IF ( MYROW .NE. 0 .AND. MYCOL .EQ. 0 )THEN
         CALL CGESD2D ( ICTXT, MXLLDB, MXRHSC, BB, MXLLDB, 0, 0 )
    END IF
```

SECCIÓN 5.- Cálculo de la potencia media reflejada, transmitida y la conservación de la energía de la PCW

```
    IF ( ( MYROW.EQ.0 ) .AND. ( MYCOL.EQ.0 ) ) THEN
```

```
        CALL ENERGY ( NPT, NANG, FDO, POL, G, THETAO, WNUM, AK, SLITW,
&            LARGE, SF, SG, DSF, DSG, DELTA, EI, NORFAC, CONSUP, CONSDOWN,
&            ENERBAL, DTHET, REFLPOW, TRANSPOW, NFF)

        PRINT*, ID, 'FREQUENCY = ', WNUM,
        PRINT*, 'MEAN REFLECTED POWER =', REFLPOW
        PRINT*, 'MEAN TRANSMITTED POWER =', TRANSPOW
        PRINT*, 'MEAN SCATTERED POWER =', ENERBAL
    END IF
ENDDO

CALL BLACS_EXIT( 0 )
STOP
END PROGRAM
```

****

A continuación explicaremos en más detalle cada parte del Programa 4. Algunas partes serán omitidas en la explicación porque ya fueron explicadas a detalle en el Capítulo 3.

Sección 1

En esta sección del programa se inicializan las variables, arreglos y se declaran las rutinas externas que se utilizarán en dicho programa. A continuación explicaremos la función de las variables de mayor relevancia para el funcionamiento del programa.

Variables	Definición
LAMBD	Longitud de onda en micras.
NWA	Semi-longitud de la guía de ondas (unidades de longitud de onda).
NWB	Espesor de la guía de ondas (unidades de longitud de onda).
NW	Número de puntos para el ancho de la rendija.
TINID	Ángulo inicial de incidencia (grado).
NANG	Número de ángulos de incidencia (grado).
G	Ancho del haz gaussiano.
HSLITW	Ancho medio de la rendija.

FDO Tipo de frente de onda.

Nota *1 - plana; 2 - gaussiana.*

ME Matriz de dimensión (NPT, NPT).

EI Matriz de dimensión $(NPT, NANG)$, la cual es la matriz inhomogénea del sistema algebraico que después del uso de la rutina $PCGESV$ contendrá la solución del sistema.

Sección 2

En esta sección se encuentran los parámetros generales que serán utilizados en el programa. También se encuentra el ciclo que hace el llenado del arreglo *THETA*, el cual contiene los ángulos a los cuales se hará incidir la onda plana a la PCW. En nuestro programa sólo utilizaremos el ángulo de incidencia normal (0^o).

Sección 3

En esta sección se tiene el ciclo principal, el cual realiza las variaciones de la frecuencia en intervalos *DW*. A continuación daremos una pequeña explicación de cual es la función de cada rutina.

NORTE.- Esta rutina se encarga de calcular el factor de normalización, el cual será almacenado en el arreglo *NORFAC*.

PERFILES.- Esta rutina genera el perfil geométrico de la PCW calculando sus coordenadas paramétricas; así como su primera y segunda derivada, las cuales serán almacenadas en los arreglos *SF, SG, DSF, DSG, DDSF, DDSG*.

AKAN.- Esta rutina calcula el espectro del campo incidente de forma analítica, el cual es almacenado en el arreglo *AK*.

DESCINIT.- Esta rutina ya fue explicada en la sección de Inversión de Matriz.

MATINIT.- Esta rutina llena las submatrices ME, EI y las distribuye a cada proceso.
PCGESV.- Esta rutina es la encargada de calcular la solución del sistema de ecuaciones lineales complejas.

Sección 4

En esta sección se ordena la solución dada por la rutina $PCGESV$, ya que ésta la da en desorden y más adelante requeriremos que la solución esté ordenada.

Sección 5

En esta sección se llama a la rutina $ENERGY$ la cual calcula la potencia media reflejada y transmitida, verificando que se cumpla la conservación de la energía al ser iluminada la PCW. Las otras rutinas son las encargadas de terminar el programa las cuales ya fueron explicada en el capítulo 3.

V.1.3. Algoritmo CUDA

En esta sección se muestra la implementación de un programa en CUDA C que utiliza la biblioteca de CULA aplicado al mismo Método Integral en forma paralela. Como parte de la paralelización en el algoritmo es construir la matriz usando las funciones Hankel de primera y segunda especie, no se pudo utilizar CUDA FORTRAN ya que el compilador de PGIFORTRAN no las contiene. Por esta razón se tuvo que utilizar el lenguaje C. En el programa 5 se muestra el algoritmo que se implementó para calcular la respuesta óptica de la PCW.

****

Programa 5. Método Integral para una PCW usando CULA

****

SECCIÓN 1.- Declaración de librerías que se utilizarán en el programa.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "subrutinas.cu"
#include <cuda.h>
#include <cula_lapack.h>

int main ( ) {
```

SECCIÓN 2.- Declaración y inicialización de variables.

```
    int npt, nang, npa, npb, nw, npcili, nper, nff, k0, id, fdo, pol;
    float pi, tinid, dthet, delta, nwa, nwb, g, b, phase, period;
    float f, r, p;

    pi = 3.141593;
    b = 1.0 * pi;
    phase = 0.5 * pi;
    period = 2.0 * pi;
    nf f= 181;
    nwa = 28.0;
    nper = 11;
    nwb = nper * period;
    delta = 1.0 / 15.0;
    nang = 1;
    npa = ( int ) nwa / delta;
    f = .005;
    r = sqrt ( f * period * b / pi );
    p = 2.0 * pi * r;
    npcili = ( int )( p / delta );
    npb = ( int )( nwb / delta );
    npt = 2 * ( 2 * npa + npb ) + nper * npcili;

    float rad, wnum, hslitw, slitw, hlarge, large, dw, d, A1, A2;
    float wnmin, wnmax, consup, consdown;
```

```
    float theta0 [ nang ], norfac [ nang ], sf [ npt ], sg [ npt ];
    float dsf [ npt ], dsg[npt], ddsf[npt], ddsg[npt], variables[3];

    wnmin = 1.0;
    wnmax = 3.0;
    nw = 200;
    pol = 2;
    fdo = 2;

    culaFloatComplex* me_d;
    culaFloatComplex* ei_d;
    culaFloatComplex* ei_h;
    culaFloatComplex* ak;

    float* sf_d;
    float* sg_d;
    float* dsf_d;
    float* dsg_d;
    float* ddsf_d;
    float* ddsg_d;
    float* theta0_d;
    culaInt* IPIV;

    culaStatus status;

    ei_h = ( culaFloatComplex * )
        malloc ( npt * nang * sizeof ( culaFloatComplex ) );
    ak = ( culaFloatComplex * )
        malloc ( nff * nang * sizeof ( culaFloatComplex ) );
```

SECCIÓN 3.- Asignación de memoria de los arreglos que se encontrarán almacenados en la GPU.

```
    cudaMalloc ( ( void** ) &me_d, npt * npt *
        sizeof ( culaFloatComplex ) );
    cudaMalloc ( ( void** ) &ei_d,npt * nang *
        sizeof ( culaFloatComplex ) );
    cudaMalloc ( ( void** ) &theta0_d, nang * sizeof ( float ) );
    cudaMalloc ( ( void** ) &sf_d, npt * sizeof ( float ) );
```

```
    cudaMalloc ( ( void** ) &sg_d, npt * sizeof ( float ) );
    cudaMalloc ( ( void** ) &dsf_d, npt * sizeof ( float ) );
    cudaMalloc ( ( void** ) &dsg_d, npt * sizeof ( float ) );
    cudaMalloc ( ( void** ) &ddsf_d, npt * sizeof ( float ) );
    cudaMalloc ( ( void** ) &ddsg_d, npt * sizeof ( float ) );
    cudaMalloc ( ( void** ) &IPIV, npt * sizeof ( culaInt ) );
```

SECCIÓN 4.- Parámetros generales.

```
    printf ( "radio = %f  npcili = %d  npt = %d\n", r, npcili, npt );

    rad = pi / 180.0;
    d = nwb;
    slitw = b;
    hslitw = 0.5 * b;
    hlarge = nwa + hslitw;
    large = 2.0 * hlarge;
    A1 = 0.0 * b;
    A2 = 0.0 * b;
    wnum = wnmin;
    g = 2.0 * nwa / 5.0;
    dthet = 1.0;
    tinid = 0.0;
```

SECCIÓN 5.- Llenado del arreglo que almacena los ángulos a los cuales se hará incidir el haz.

```
    for ( k0 = 1; k0 <= nang; k0++ ) {
        theta0 [ k0-1 ] = tinid + ( k0-1.0 ) * dthet;
        theta0 [ k0-1 ] = rad * theta0 [ k0-1 ];
    }
```

SECCIÓN 6.- Inicialización de la tarjeta gráfica, ciclo principal encargado de realizar las variaciones de frecuencia, llamado a las rutinas y copia de arreglos de la memoria de la máquina a la memoria de la GPU.

```
    dw = 1.0 / ( nw-1.0 );
    status = culaInitialize ( );
    checkStatus ( status );

    for ( id = 1; id <= nw; id++ ) {
        wnum = wnmin + ( wnmax-wnmin ) * dw * ( id-1.0 );

        norte ( norfac, wnum, g, theta0, nang, fdo, large, slitw );
        perfiles ( sf, sg, dsf, dsg, ddsf, ddsg, nwa, nwb, npt, npa,
            npb, hslitw, hlarge, delta, A1, A2, period, phase, pi, b,
            r, npcili, nper );

        cudaMemcpy ( sf_d, sf, npt * sizeof ( float ),
            cudaMemcpyHostToDevice );
        cudaMemcpy ( sg_d, sg, npt * sizeof ( float ),
            cudaMemcpyHostToDevice );
        cudaMemcpy ( dsf_d, dsf, npt * sizeof ( float ),
            cudaMemcpyHostToDevice );
        cudaMemcpy ( dsg_d, dsg, npt * sizeof ( float ),
            cudaMemcpyHostToDevice );
        cudaMemcpy ( ddsf_d, ddsf, npt * sizeof ( float ),
            cudaMemcpyHostToDevice );
        cudaMemcpy ( ddsg_d, ddsg, npt * sizeof ( float ),
            cudaMemcpyHostToDevice );
        cudaMemcpy ( theta0_d, theta0, nang * sizeof ( float ),
            cudaMemcpyHostToDevice );
```

SECCIÓN 7.- Definición del tamaño de la malla y de los bloques que se utilizarán, llamado a las rutinas encargadas de llenar la matriz de ecuaciones; así como la encargada de calcular la solución al sistema.

```
        dim3 blockSize ( 32, 32, 1 );
        dim3 gridSize ( floor ( npt/32 ) +32, floor ( npt/32 ) +32, 1 );

        matmYeinumnr_kernel <<<gridSize,blockSize>>> ( me_d, npt, sf_d,
            sg_d, dsf_d, dsg_d, ddsf_d, ddsg_d, delta, wnum, ei_d, nang,
            theta0_d, g, d );

        status = culaCgesv ( npt, nang, me_d, npt, IPIV, ei_d, npt );
        checkStatus ( status );

        cudaMemcpy (ei_h, ei_d, npt * nang * sizeof ( culaFloatComplex ),
            cudaMemcpyDeviceToHost );
```

SECCIÓN 8.- En esta sección se llama a la rutina encargada de calcular el espectro del campo incidente en su forma analítica; así como la encargada de calcular la potencia media reflejada y transmitida para verificar la conservación de la energía al ser iluminada la PCW y por último, pero no menos importante la liberación de la memoria de la GPU.

```
        akan ( ak, nff, nang, large, theta0, wnum, g, d, fdo );
        energy ( npt, nang, fdo, pol, g, theta0, wnum, ak, slitw, large,
                sf, sg, dsf, dsg, delta, ei_h, norfac, consup, consdown,
                dthet, nff, variables );

        printf ( "id: %d      Frequency = %f\n", id, wnum );
        printf ( "MEAN REFLECTED POWER = %.10f\nMEAN TRANSMITTED
            POWER = %.7e\nMEAN SCATTERED POWER = %.10f\n",
            variables [ 0 ], variables [ 1 ], variables [ 2 ] );
    }

    printf ( "Shutting down CULA\n\n" );
    culaShutdown ( );
    cudaFree ( me_d );
    cudaFree ( sf_d );
    cudaFree ( sg_d );
    cudaFree ( dsf_d );
    cudaFree ( dsg_d );
```

```
    cudaFree ( ddsf_d );
    cudaFree ( ddsg_d );
    cudaFree ( ei_d );
    cudaFree ( theta0_d );
    return 0;
}

**************************************************************
```

En general este programa es muy similar a los programas antes descritos; por lo tanto nada más nos enfocaremos en las diferencias principales, las cuales se encuentran en la sección 7. Principalmente se define el tamaño de la malla y de los bloques que utilizará la GPU, por medio de la rutina *matmYeinumnr_kerner*, la cual es la encargada de llenar las matrices ME y EI en forma paralela. Después se encuentra el llamado a la rutina *culaCgesv*, la cual es la encargada de encontrar la solución al sistema de ecuaciones. Por último se encuentra la rutina encargada de copiar la solución del sistema de la memoria de la GPU a la memoria de la CPU.

Como se puede observar, en CULA no se tiene que indicar cómo se realizará la repartición del trabajo en la GPU para encontrar la solución al sistema, como lo realizamos con ScaLAPACK. Esto es una gran ventaja de la programación en paralelo en la GPU porque el algoritmo es más sencillo de desarrollarse.

V.2. Resultados

A continuación se muestran los resultados obtenidos para diferentes versiones de los algoritmos tanto en su forma secuencial como en paralelo del IEM para estudiar la respuesta óptica de una PCW. En la tabla II se muestran los resultados del tiempo de cómputo en segundos para una matriz cuadrada variando el número de periodos

de la PCW y utilizando cuatro distintas formas de programación. La primera es la secuencial, que es la forma tradicional de programación. Para la forma secuencial se tienen dos columnas distintas: una es usando las librerías de LAPACK y la otra es con las librerías de MKL (Intel). Las otras tres formas de programación usan el paradigma de paralelización usando distintas librerías y distintos grados de paralelización. Para MPI se usa la comunicación no bloqueante y la colectiva, que para estos casos la parte que se paraleliza es el ciclo principal que se encarga de calcular las 200 frecuencias usando el método integral. Otra de las formas de paralelización que se utilizó, fue el uso de la librería de ScaLAPACK. Para este caso se paralelizó la inversión como la construcción de la matriz. Por último tenemos la paralelización mediante la tarjeta gráfica (GPU). En este caso se tienen 2 columnas: en una de ellas la construcción de la matriz se realiza de forma secuencial, pero la inversión de la matriz se realiza mediante el paradigma de paralelización con CUDA FORTRAN, y en la última columna, tanto la construcción de la matriz como la inversión se realizan con CUDA C.

Como se esperaba, en la tabla II podemos notar que los tiempos de cómputo llevados acabo por las versiones en paralelo son mucho más pequeños que las versiones en secuencial. Esto se debe a que se hace un uso más eficiente de los recursos de la máquina.

De igual manera vamos a hacer una comparación de todas las versiones respecto a la versión secuencial LAPACK tomando en consideración la matriz de 13856×13856. Primero, con la versión secuencial usando la librería MKL de Intel se obtuvo una rapidez de 26 veces con respecto a la versión secuencial de LAPACK. Ahora haciendo la comparación con las versiones paralelas que usan la CPU, podemos observar que la versión de ScaLAPACK obtuvo una rapidez de 32 veces; así como las versiones de MPI con comunicación No bloqueante y Colectiva obtuvieron una rapidez de 46 y 53 veces

Tabla II. Tiempo de cómputo para calcular la respuesta óptica de una PCW finita variando el número de periodos usando algoritmos del método integral en las formas secuencial y paralela.

Entrada	**Num. Periodos**	**Secuencial LAPACK (s)**	**Secuencial MKL (INTEL) (s)**	**Scalapack (Inv. + Cons.)(s)**	**Paralelo FORTRAN (Inv.)(GPU) (s)**	**MPI (No bloqueante) (s)**	**MPI Colectiva (s)**	**Paralelo C (Inv. + Cons.) (GPU)(s)**
2156 × 2156	10	766.13	129.66	79.82	133.48	23.16	20.19	35.24
3356 × 3356	50	2766.41	289.32	183.09	286.46	67.33	61.66	61.13
4856 × 4856	100	8022.10	614.51	419.27	566.41	185.29	165.13	111.78
6356 × 6356	150	17951.68	1066.49	776.72	1004.24	421.77	349.30	185.93
7856 × 7856	200	33971.93	1797.86	1278.97	1509.80	741.02	645.75	287.61
10856 × 10856	300	85653.91	3859.40	2930.97	2767.35	1903.44	1674.40	621.44
13856 × 13856	400	181620.76	6929.26	5603.89	4869.25	3930.45	3389.61	1533.31
16856 × 16856	500	327520.96	10860.07	9491.46	7055.29	10763.34	9836.16	2884.68

más rápido, respectivamente. Finalmente para las versiones que usan la programación en paralelo en la GPU, tanto la que nada más invierte la matriz, como la que construye la matriz en la GPU y la invierte se obtuvo una rapidez de 37 y 118 veces más rápido, respectivamente. Esto nos muestra que la mejor forma de estudiar la respuesta óptica de una PCW es utilizando la GPU. Por otro lado, si nada más se quiere usar la CPU, la librería de MPI es la mejor forma. Sin embargo, tanto usando la GPU como los procesadores con MPI tienen una pequeña desventaja respecto a las versiones secuenciales y la paralela usando ScaLAPACK, la cual es el tamaño de la memoria requerida.

Para este trabajo se utilizó una tarjeta TESLA k40 con 12 GB de memoria RAM, la cual nada más permite invertir matrices complejas de un rango de 26 mil en precisión simple. Para MPI la computadora cuenta con 64 GB de memoria RAM, pero cada procesador utiliza su propia matriz del tamaño de la entrada. Por ejemplo, para la matriz más grande que se encuentra en la tabla II esta ocupará un espacio de 2.2 GB de memoria RAM por procesador, como se utilizaron 32 procesadores esto da un total de 70.4 GB de memoria RAM que sobrepasa la capacidad de la máquina. Esto hace que se tenga la necesidad de usar la memoria de intercambio (SWAP) reduciendo el rendimiento como se ve en la tabla II. De igual manera se aprecia en la Fig. 9 en el último punto de MPI (no bloqueante y colectiva) una anormalidad en el tiempo de cómputo causado por el uso de la memoria de intercambio.

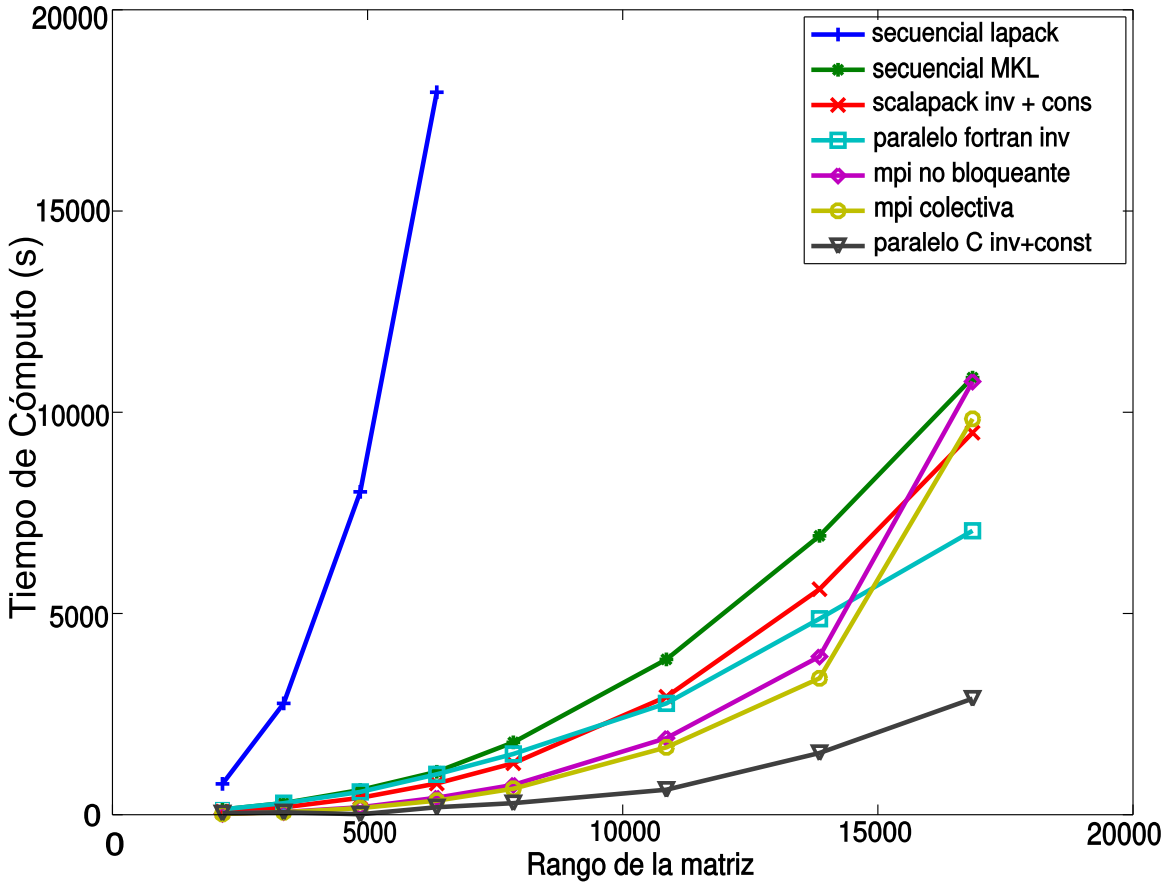

Figura 9. Tiempo de cómputo en segundos para el cálculo de la respuesta óptica en forma secuencial y paralela variando el número de períodos de la PCW.

Capítulo VI

CONCLUSIONES

En este capítulo mencionamos un breve resumen y en base a los resultados obtenidos enunciamos las conclusiones más importantes del trabajo.

Hemos dado un gran avance para mejorar el rendimiento que se puede alcanzar utilizando librerías especializadas en el cómputo paralelo. Para lograr esto hemos utilizado el modelo de programación en paralelo bajo distintos protocolos y librerías como son: MPI, ScaLAPACK y CUDA. Todo esto con el fin de obtener el mayor rendimiento posible aprovechando las ventajas que cada uno ofrece. Estas técnicas numéricas fueron aplicadas a un problema particular de álgebra lineal, como es la inversión de una matriz. Asimismo, esta herramienta fue utilizada como parte de una técnica numérica (conocida como el Método de la Ecuación Integral) para calcular la respuesta óptica de una guía de ondas de cristal fotónico bidimensional.

Primeramente mencionaremos las conclusiones que se tuvieron al analizar el tiempo de cómputo para calcular la inversión de una matriz general en precisión sencilla variando su rango. La implementación de los algoritmos numéricos en paralelo se desarrollaron con los protocolos de MPI y CUDA en el lenguaje de FORTRAN. Para el caso de MPI se utilizaron las librerías de ScaLAPACK y de CUDA FORTRAN las subrutinas

de CULA mediante el compilador de PGI FORTRAN. Los resultados para la matriz de rango mayor (13760×13760) muestran que para la versión de ScaLAPACK con 32 procesadores (mallado no cuadrado) y con 25 procesadores (mallado cuadrado) se obtuvo una rapidez de 13 y 18.5 veces más que las versiones secuenciales utilizando las técnicas LU y Gauss Jordan. También esto muestra que si se tiene un número mayor de procesadores ($4 \times 8 = 32$) no garantiza que los tiempos de cómputo sean menores. En cambio si hace una distribución de las tareas en una forma cuadrada ($5 \times 5 = 25$) haciendo que los procesadores tengan una carga similar de trabajo, se puede lograr un rendimiento mejor. Por otro lado, con las subrutinas de CULA se logró 98 veces más rápido que las versiones secuenciales. Todo esto nos indica que la mejor forma de calcular la inversa de una matriz es utilizando la programación en paralelo bajo el protocolo de CUDA con las librerías de CULA.

Como el objetivo principal de este trabajo de investigación fue implementar las herramientas numéricas previamente descritas en el método integral desarrollado, brevemente se mencionará en que consiste. Este método parte del segundo teorema integral de Green permitiendo obtener un par de ecuaciones integrales acopladas que involucran, como incógnitas el modo del campo y su derivada normal evaluados en las fronteras o contornos involucrados. La discretización del sistema resulta en una ecuación matricial inhomogénea cuya solución determina las funciones fuente, con las que se puede obtener la distribución del campo electromagnético. De esta manera calculamos la respuesta óptica de una PCW bidimensional de longitud finita.

En este trabajo se analizaron diferentes formas de programación en paralelo aplicado al método integral, usando tanto los procesadores como la tarjeta gráfica; así como en la forma secuencial para hacer un comparativo del tiempo de cómputo variando el número de periodos de la PCW. Para la forma secuencial se usaron las librerías de

LAPACK (externas) y MKL (internas) del compilador de Intel. Las otras formas de programación con el paradigma de paralelización fueron usando: las librerías de MPI con la comunicación No bloqueante y la Colectiva (que para estos casos la parte que se paraleliza es el ciclo principal que se encarga de calcular las 200 frecuencias), las librerías de ScaLAPACK (para la construcción e inversión de la matriz) y las librerías de CULA en la tarjeta gráfica (para el caso cuando la construcción de la matriz se realiza de forma secuencial y la inversión de la matriz mediante CUDA FORTRAN; así como el caso cuando la construcción e inversión de la matriz se realiza con CUDA C).

Las pruebas que se consideraron en este trabajo, fue tomando en cuenta la variación del número de periodos de 10 hasta 500 produciendo rangos de matrices de 2156×2156 hasta 16856×16856. En particular, para la matriz de rango 13856×13856, con la versión secuencial usando la librería MKL de Intel se obtuvo una rapidez de 26 veces más con respecto a la versión secuencial de LAPACK. En relación a las versiones paralelas con los procesadores de la CPU, usando ScaLAPACK se tuvo una rapidez de 32 veces; así como las versiones de MPI con comunicación No bloqueante y Colectiva fueron de 46 y 53 veces más rápido, respectivamente. Por otro lado, para las versiones que usan la programación en paralelo con la GPU, tanto la que nada más invierte la matriz, como la que construye e invierte la matriz se logró una rapidez de 37 y 118 veces más que la versión secuencial de LAPACK.

A pesar de que los resultados muestran que la forma más óptima para programar en paralelo es hacerlo en la GPU; se tiene que la tarjeta TESLA k40 con 12 GB en RAM presenta una desventaja, la cual es en el tamaño de la memoria que nada más permite invertir matrices complejas de rango 26000×26000 en precisión simple. En cambio, usando los 32 procesadores y 64 GB en RAM con ScaLAPACK distribuye la memoria de manera óptima. Bajo estas mismas condiciones, el protocolo de MPI (No

bloqueante o colectiva) también tiene un alto costo ya que utiliza la misma cantidad de memoria RAM en cada uno de los procesadores utilizados; es decir la cantidad de memoria total es el producto del número de procesadores y la memoria individual que utiliza cada procesador. En consecuencia, si queremos resolver un problema numérico hay que tomar en cuenta: la cantidad de memoria requerida en el cálculo, la memoria que se tiene disponible y el tiempo que le tomará el proceso de cálculo. Todo esto nos puede ayudar a tomar la decisión de cual algoritmo utilizar. Otra consideración a tomar en cuenta es que para programar algoritmos paralelos requiere tener más conocimientos de programación para poder lograr el mejor rendimiento.

Por lo tanto, la conclusión principal de este trabajo es que con estas técnicas de programación en paralelo bajo el protocolo de MPI y CUDA permiten modelar sistemas complejos en tiempos de cómputo relativamente muy cortos. Como trabajo futuro se pretende implementar el método integral con la combinación de estas técnicas de paralelización: OpenMP + MPI + CUDA, permitiendo estudiar las diferentes propiedades y características de una guía de ondas de cristal fotónico en diversas situaciones a gran escala.

Apéndice A

ALGORITMOS PARA INVERSIÓN DE MATRIZ CON SCALAPACK

En este apartado se muestran los códigos de las rutinas que fueron utilizadas para invertir una matriz utilizando ScaLAPACK.

A.1. Algoritmo "DISTRIBUCIONCICLICA"

Esta rutina calcula los índices de las columnas o de las filas para realizar la distribución cíclica.

```
**********************************************************************

SUBROUTINE DISTRIBUCIONCICLICA ( M, NP, N, NB )

  IMPLICIT        NONE
  INTEGER         NP,N,NB
  INTEGER         M(NP,N)
  INTEGER         C,CO,CONTADOR,I

  C = 1
  CONTADOR = 0
  DO WHILE ( C.LE.N )
    CO = 1
    DO WHILE ( CO.LE.NP )
      I = 1
```

```
      DO WHILE ( I.LE.NB )
        IF ( C.LE.N ) THEN
          M (CO, I + ( CONTADOR*NB ) ) = C
          C = C + 1
        ELSE
          I = NB + 1
          CO = NP + 1
        END IF
        I = I + 1
      END DO
      CO = CO + 1
    END DO
    CONTADOR = CONTADOR + 1
  END DO
  RETURN
END
```

****

A.2. Algoritmo "MATINIT"

Esta rutina es la encargada de hacer la distribución de las matrices A y B a cada proceso.

****

```
SUBROUTINE MATINIT ( AA, DESCA, B, DESCB, ATEM, BTEM, N, NPC, NPR, TC,
 $                   TR )

  INTEGER            DESCA( * ), DESCB( * )
  INTEGER            N,NPC,NPR
  REAL               AA( * ), B( * ), ATEM ( N, N ), BTEM ( N, N )
  INTEGER            CTXT_, LLD_, X, Y
  PARAMETER          ( CTXT_ = 2, LLD_ = 9 )
  INTEGER            ICTXT, MXLLDA, MXLLDB, MYCOL, MYROW, NPCOL, NPROW
  EXTERNAL           BLACS_GRIDINFO
  LOGICAL            BOL1,BOL2
  INTEGER            TC ( NPC, N ),TR ( NPR, N )
  ICTXT = DESCA( CTXT_ )

  CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
```

```
  MXLLDA = DESCA( LLD_ )
  MXLLDB = DESCB( LLD_ )
  BOL1 = .TRUE.
  X=1

  DO WHILE ( BOL1 )
    BOL2=.TRUE.
    Y=1
    DO WHILE ( BOL2 )
      AA ( Y + ( X - 1 ) * MXLLDA ) = ATEM ( TR ( MYROW + 1, Y ),
$          TC ( MYCOL+1, X ) )
      B ( y + ( X - 1 ) * MXLLDA ) = BTEM ( TR ( MYROW + 1, Y ),
$          TC ( MYCOL + 1, X ) )
      Y = Y + 1
      IF ( TR ( MYROW + 1, Y) .EQ.0 ) THEN
        BOL2 = .FALSE.
      END IF
    END DO
    X = X+1
    IF ( TC ( MYCOL + 1, X ) .EQ.0 ) THEN
      BOL1 = .FALSE.
    END IF
  END DO

RETURN
END

**********************************************************************
```

Apéndice B

ALGORITMOS PARA CALCULAR LA RESPUESTA ÓPTICA DE UNA GUÍA DE ONDAS DE CRISTAL FOTÓNICO UTILIZANDO SCALAPACK

En este apartado se muestran los códigos de las rutinas que fueron utilizadas para calcular la respuesta óptica de una PCW utilizando ScaLAPACK.

B.1. Algoritmo "MATINIT"

Esta rutina llena las submatrices ME, EI y las distribuye a cada proceso.

```
*****************************************************************

SUBROUTINE MATINIT ( AA, DESCA, B, DESCB, N, NPC, NPR, TC, TR, NPT,
$          NANG, SF, SG, WNUM, G, D, DSF, DSG, DDSF, DDSG, DELTA, THETA0 )

  IMPLICIT NONE

  INTEGER            DESCA ( * ), DESCB ( * )
  INTEGER            N, NPC, NPR, NPT, NANG
  REAL               WNUM, G, D
  REAL               SF ( NPT ), SG ( NPT ), THETA0 ( NANG ),DSF (NPT ),
$                    DSG ( NPT ), DDSF (NPT ), DELTA,DDSG (NPT)
  COMPLEX            AA ( * ), B ( * ), TEM
```

```
      INTEGER            CTXT_, LLD_, X, Y
      PARAMETER          ( CTXT_ = 2, LLD_ = 9 )
      INTEGER            ICTXT, MXLLDA, MXLLDB, MYCOL, MYROW, NPCOL, NPROW
      EXTERNAL           BLACS_GRIDINFO
      LOGICAL            BOL1, BOL2
      INTEGER            TC ( NPC, N ), TR( NPR, N )

      ICTXT = DESCA ( CTXT_ )
      CALL BLACS_GRIDINFO ( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

      MXLLDA = DESCA( LLD_ )
      MXLLDB = DESCB( LLD_ )
      BOL1 = .TRUE.
      X = 1

      DO WHILE ( BOL1 )
        BOL2 = .TRUE.
        Y = 1
        DO WHILE ( BOL2 )
          CALL MATM ( TEM, NPT, SF, SG, DSF, DSG, DDSF, DDSG, DELTA, WNUM,
     $         TR ( MYROW + 1, Y ), TC ( MYCOL + 1, X ) )
          AA ( Y + ( X - 1 ) * MXLLDA ) = TEM
          IF ( MYCOL.EQ.0 .AND.x.EQ.1 ) THEN
            CALL EINUMNR ( TEM, NPT, NANG, SF, SG, THETA0, WNUM, G, D,
     $           TR ( MYROW + 1,Y ), TC( MYCOL + 1, X ) )
            B ( Y + ( X - 1 ) * MXLLDA ) = TEM
          END IF
          Y = Y + 1
          IF ( TR ( MYROW + 1,Y ).EQ.0 ) THEN
            BOL2 = .FALSE.
          END IF
        END DO
        X = X + 1
        IF ( TC ( MYCOL + 1, X ).EQ.0 ) THEN
          BOL1 = .FALSE.
        END IF
      END DO

      RETURN
      END

*******************************************************************
```

B.2. Algoritmo "ORDENA"

Esta rutina es la encargada de ordenar la solución que produce cada procesador.

```
*****************************************************************

SUBROUTINE ORDENA (BTEM, B, MYROW, MYCOL, N, M, MXLLDB, MXRHSC, TR,
$           NPROW )

  IMPLICIT       NONE

  INTEGER        MYROW, MYCOL, N, M, MXLLDB, MXRHSC, NPROW, CON
  INTEGER        TR ( NPROW, N )
  COMPLEX        B ( MXLLDB, MXRHSC ), btem( N, M )
  LOGICAL        BOL

  CON = 1
  BOL = .TRUE.

  DO WHILE ( BOL )
    IF (TR(MYROW+1,CON).NE.0) THEN
      BTEM ( TR ( MYROW + 1, CON ), M ) = B ( CON, MXRHSC )
      CON = CON + 1
    ELSE
      BOL = .FALSE.
    END IF
  END DO

RETURN
END

*****************************************************************
```

Apéndice C

ALGORITMOS PARA CALCULAR LA RESPUESTA ÓPTICA DE UNA GUÍA DE ONDAS DE CRISTAL FOTÓNICO UTILIZANDO CUDA

En este apartado se muestran los códigos de las rutinas que fueron utilizadas para calcular la respuesta óptica de una PCW utilizando CUDA.

C.1. Algoritmo "matmYeinumnr"

Esta rutina es la encargada de llenar las matrices ME y EI en forma paralela usando la GPU.

```
**************************************************************

__global__ void matmYeinumnr_kernel ( culaFloatComplex* ME, int NPT,
float* SF, float* SG, float* DSF, float* DSG, float* DDSF, float* DDSG,
float DELTA, float WNUM, culaFloatComplex *EI, int NANG, float* THETA0,
float G, float D ) {

  float nume, arg1, gamma, aux, dq, q, alpha0, Aqkp, arg, pi, kp, x1,
  float x3, t;
  culaFloatComplex im1, c2, temporal ,sum, res, z;
  int i, j, tem, nq;
  pi = 3.141592654f;
```

```
im1.x = 0.0f;
im1.y = 1.0f;
nume = expf ( 1.0f );

c2.x = 0.25f*im1.x*DELTA;
c2.y = 0.25f*im1.y*DELTA;
i = threadIdx.x + blockIdx.x * blockDim.x;
j = threadIdx.y + blockIdx.y * blockDim.y;

if ( j < NPT ) {
  //math
  if ( i < NPT ) {
    if ( i == j ) {
      aux = WNUM*DELTA/( 2.0f * nume );
      gamma = sqrtf ( DSF[ i ] * DSF [ i ] + DSG [ i ] * DSG [ i ] );
      arg1 = aux * gamma;
      ME [ NPT * i + j ].x = c2.x * j0f ( arg1 ) - c2.y * y0f ( arg1 );
      ME [ NPT * i + j ].y = c2.x * y0f ( arg1 ) + c2.y * j0f ( arg1 );
    }
    else {
      arg1 = WNUM * sqrtf ( ( SF [ i ] - SF [ j ] ) *
             ( SF [ i ] - SF [ j ] ) + ( SG [ i ] - SG [ j ] ) *
             ( SG [ i ] - SG [ j ] ) );
      ME [ NPT * i + j ].x = c2.x * j0f ( arg1 ) - c2.y * y0f ( arg1 );
      ME [ NPT * i + j ].y = c2.x * y0f ( arg1 ) + c2.y * j0f ( arg1 );
    }
  }
  //einumnr
  else {
    if ( i - NPT < NANG ) {
      nq = 721;
      dq = 2.0f * WNUM / ( ( nq * 1.0f ) - 1.0f );
      kp = WNUM * sin ( THETA0 [ i - NPT ] );
      x1 = SF [ j ];
      x3 = SG [ j ] - D;
      sum.x = 0.0f;
      sum.y = 0.0f;
      for ( tem = 1; tem < nq - 3; tem++ ){
      q = - WNUM + ( tem * 1.0 ) * dq;
      alpha0 = sqrtf ( WNUM * WNUM - q * q );
      Aqkp = sqrtf ( pi ) * G * expf ( - ( G * ( q - kp ) / 2.0f ) *
             ( G * ( q - kp ) / 2.0f ) );
      arg = q * x1 - alpha0 * x3;
      z.x = ( im1.x ) * arg;
```

```
        z.y = ( im1.y ) * arg;
        t = expf ( z.x );
        sincosf ( z.y, &res.y, &res.x );
        res.x * = t;
        res.y * = t;
        sum.x = sum.x + Aqkp * res.x;
        sum.y = sum.y + Aqkp * res.y;
      }
      temporal.x = ( sum.x ) * dq / ( 2.0f * pi );
      temporal.y = ( sum.y ) * dq / ( 2.0f * pi );
      EI [ j + ( i - NPT ) * NPT ].x = temporal.x;
      EI [ j + ( i - NPT ) * NPT ].y = temporal.y;
    }
  }
}
```

****

Referencias

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. En *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), páginas 483–485, New York, NY, USA. ACM.

Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J., Dongarra, J. J., Du Croz, J., Hammarling, S., Greenbaum, A., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide (Third Ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. ISBN 0-89871-447-8.

Blackford, L. S., Choi, J., Cleary, A., D'Azeuedo, E., Demmel, J., Dhillon, I., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. (1997). *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. ISBN 0-89871-397-8.

Chapman, B., Jost, G., Van der Pas, R., and Kuck, D. J. (2008). *Using OpenMP : portable shared memory parallel programming*. MIT Press, Cambridge, Mass., London.

Cook, S. (2013). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, primero edición.

EM (2009). *CULA tools, CULA Reference Manual*. EM Photonics, Inc., East Main Street, primera edición edición. 61 pp.

Inan, U. and Marshall, R. (2011). *Numerical Electromagnetics: The FDTD Method*. Cambridge University Press.

Jackson, J. (1999). *Classical electrodynamics*. Alhambra.

John, S. (1987). Strong localization of photons in certain disordered dielectric superlattices. *Physical review letters*, **58**: 2486–2489.

Kaeli, D. R., Mistry, P., Schaa, D., and Zhang, D. P. (2015). *Heterogeneous Computing with OpenCL 2.0*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, primero edición.

Mendoza-Suárez, A. and Pérez-Aguilar, H. (2016). Numerical integral methods to study plasmonic modes in a photonic crystal waveguide with circular inclusions that involve a metamaterial. *Photonics*, **21**: 1–12.

Mendoza-Suárez, A., Villa-Villa, F., and Gaspar-Armenta, J. A. (2006). Numerical method based on the solution of integral equations for the calculation of the band structure and reflectance of one- and two-dimensional photonic crystals. *J. Opt. Soc. Am. B*, **23**(10): 2249–2256.

Pacheco, P. (2011). *An Introduction to Parallel Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pérez, H. I., Valencia, C. I., Méndez, E. R., and Sánchez-Gil, J. A. (2009). On the transmission of diffuse light through thick slits. *J. Opt. Soc. Am. B*, **26**(4): 909–918.

Pérez Hernández, E. (2015). *Estudio Numérico de la Propagación de la Luz en Guía de Ondas Periódicas y Onduladas Usando Programación en Paralelo*. Tesis de Licenciatura, Facultad de Ciencias Físico Matemáticas de la UMSNH.

Sánchez López, S. (2016). *Estudio Numérico de la Programación en Paralelo Usando el Protocolo Cuda Fortran*. Tesis de Licenciatura, Facultad de Ciencias Físico Matemáticas de la UMSNH.

Sözüer, H. S., Haus, J. W., and Inguva, R. (1992). Photonic bands: Convergence problems with the plane-wave method. *Phys. Rev. B*, **45**: 13962–13972.

Tennessee, U., California, U., Berkeley, Denver, U. C., and Ltd, N. (2012). Scalapack. [Web; accedido el 10-01-2017].

Yablonovitch, E. (1987). Inhibited spontaneous emission in solid-state physics and electronics. *Physical review letters*, **58**: 2059–2062.