Optimización y Paralelización de Software de Monte Carlo para el Estudio de Nano-Dispositivos Semiconductores

Sara M. Rubio Largo [1], Álvaro Rubio Largo [2], Miguel A. Vega Rodríguez [2] y Jesús Enrique Velázquez Pérez [1]

Resumen— **En este trabajo se presenta una optimización de un software científico utilizado en el estudio de nano-dispositivos semiconductores basado en un simulador de tipo Monte Carlo. Se ha realizado un proceso de optimización del código fuente original escrito en Fortran haciendo uso de librerías OpenMP. En primer lugar, se optimizó el código unidimensional (1D), con el fin de abordar posteriormente el caso más complejo de un simulador bidimensional (2D). Además, se ha estudiado el comportamiento de ambos simuladores sobre una arquitectura Intel Xeon X5550 (con dos procesadores quad-core), variando el número de núcleos (1, 2, 4 y 8 núcleos); obteniendo eficiencias superiores al 60% en todos los casos. Por último, se ha realizado una comparativa de nuestros códigos 1D y 2D con otras publicaciones de la misma naturaleza encontradas en la literatura, concluyendo que nuestras métodos obtienen resultados muy prometedores.**

Palabras clave— **Nano-Dispositivos, Física Aplicada, Monte Carlo, OpenMP.**

I. Introducción

La industria microelectrónica ha transformado el mundo en las últimas décadas desde la introducción por Intel del primer microprocesador integrado hace exactamente 40 años [1]. A lo largo de las últimas cuatro décadas la industria de microelectrónica ha debido afrontar problemas serios para continuar su avance hacia menores dimensiones de los componentes: corrientes de puerta más elevadas por reducción del espesor del óxido, degradación de la movilidad de portadores en el canal, incrementos del coste de fabricación en cada transición, ..., etc.

Uno de los problemas más tempranos a los que tuvo que enfrentarse la industria fue el desarrollo de modelos para el diseño CAD (Computer-Aided Design) de componentes y circuitos. Si bien en los nodos de los años 70 (con tamaños superiores a la micra) el uso de modelos simples semi-analíticos de tipo SPICE (Software Process Improvement and Capability Determination) eran satisfactorios y suficientes para las herramientas CAD, estos modelos son claramente insuficientes para describir la complejidad del transporte de carga en dispositivos submicrométricos (nodos de los años 90) y nanométricos (actualidad).

No existen modelos analíticos que reproduzcan y prevean los comportamientos de transistores de dimensiones submicrométricas, aunque en el desarrollo de circuitos complejos se sigue usando SPICE, ya desde mediados de la década de los 80 se hace uso extensivo de herramientas de simulación de tipo TCAD en la industria para el desarrollo a nivel de componente y circuitos con un número pequeño de transistores. Las herramientas TCAD constituyen un vasto campo de trabajo en software EDA (Electronic Design Automation) que controla todas las fases de fabricación de circuitos integrados (layout, máscaras, simulaciones - eléctrica, lógica, funcional, ..., etc.). En las herramientas TCAD podemos distinguir entre las que proporcionan simulaciones eléctricas de dispositivos que resuelven las ecuaciones de Poisson (PE) y de transporte de Boltzmann (BTE) de manera acoplada usando el método de los momentos para aproximar la solución de la BTE cuya solución no aproximada es prácticamente imposible, y las que, en vez de resolver aproximaciones de la BTE, simulan localmente el transporte de carga (movimiento de electrones y huecos) y acoplan esta simulación a la PE.

El grupo liderado por J.E. Velázquez-Pérez en la USAL ha desarrollado herramientas de tipo Monte Carlo para el estudio de dispositivos en 1 y 2 dimensiones espaciales [2] [3]. Aunque las herramientas Monte Carlo tienen numerosas ventajas desde el punto de vista físico [4] [5], exigen un elevado esfuerzo computacional. A modo de ejemplo, mientras el cálculo de un punto de operación en DC de un transistor puede exigir alrededor de 1 minuto usando un modelo hidrodinámico, suele exigir alrededor de 1 hora usando el método de Monte Carlo. A cambio, las simulaciones Monte Carlo, a diferencia de las que usan resoluciones numéricas de la BTE, proporcionan información de las fluctuaciones en el dispositivo, es decir, permiten el cálculo de la densidad espectral de ruido. Esto es de gran importancia en dispositivos que operan a frecuencias por encima de 1GHz. El cálculo de la densidad espectral de ruido para un punto de operación estacionario exige tiempos de cálculo superiores a las 10 horas en 2D. Esto explica el interés del presente trabajo: la reducción del tiempo de cálculo en simulaciones Monte Carlo utilizando técnicas de paralelización.

El resto del artículo se organiza como sigue. Una definición formal del método de Monte Carlo se presenta en la Sección II. En la Sección III se describe las principales modificaciones realizadas en la par-

[1] Dep. Física Aplicada, Sección de Física, Facultad de Ciencias, Universidad de Salamanca, Plaza de la Merced s/n 37008 Salamanca, e-mail: {saramaria766, js,}@usal.es

[2] Dep. Tecnología de Computadores y Comunicaciones, Universidad de Extremadura, Escuela Politécnica, Campus Universitario s/n, 10003 Cáceres, e-mail: {arl, mavega}@unex.es

alelización del software de Monte Carlo para los simuladores de 1D y 2D. En la Sección IV se presentan todos los experimentos y comparativas entre el simulador propuesto y otros métodos de la misma naturaleza publicados en la literatura. Por último, en la Sección V aparecen las conclusiones y las líneas de trabajo futuro.

II. El Método de Monte Carlo

El método Monte Carlo está asociado con algoritmos que tratan de simular sistemas donde hay demasiados grados de libertad. Algunos ejemplos pueden ser el estudio de fluidos, el estudio de propiedades físicas de un material (semiconductores, por ejemplo), sólidos fuertemente acoplados, resolver ecuaciones diferenciales complejas, integrales definidas con condiciones de contorno complejas, etc.

En función del problema a resolver, el método Monte Carlo puede aplicarse a conveniencia, pero por lo general, suelen seguir un patrón particular:

- Definir el dominio de valores de las variables de entrada aleatorias.
- Generar muestreos de la variable aleatoria según una cierta distribución de probabilidad.
- Realizar cálculos deterministas usando esos valores de las variables aleatorias.
- Agregar los resultados de los cálculos individuales al resultado final.

El método de Monte Carlo sirve para implementar la simulación aleatoria de los fenómenos no conocidos a priori (colisiones, ...) y, en cierta medida, es similar a un método experimental pues no existe un modelo matemático con ecuaciones de tipo determinista. También se entiende que el número simulado de partículas debe ser alto, pero esto no aumenta la precisión en el sistema (para aumentar la precisión un decimal sería necesario multiplicar por 100 el tiempo de cálculo) en gran manera y los resultados macroscópicos deben entenderse con un cierto margen de error (como el error en cualquier medida real) que permite, por ejemplo, extraer el ruido electrónico en un dispositivo.

III. Paralelización del Simulador de Nano-Dispositivos Semiconductores

Este trabajo, se encuadra en una línea de investigación de desarrollo de nuevos modelos físicos para la investigación de los fenómenos de transporte de carga y calentamiento en nano-transistores. En esta línea existe un software desarrollado en Fortran que se ejecuta en modo secuencial. Dados los elevados tiempos de cálculo y la disponibilidad de procesadores multi-núcleo nos proponemos optimizar y paralelizar el código con objeto de conseguir un simulador competitivo en términos de tiempo de ejecución.

En Algoritmo 1 se detalla de manera muy básica los pasos del programa original escrito en Fortran.

Echando un vistazo al código que se nos ofrece (Algoritmo 1), puede parecer trivial parelizar el simulador insertando un par de directivas de OpenMP

Algoritmo 1 Simulador Nano-Dispositivos Semiconductores

1. Inicio del Programa
2. Inicialización de variables globales
3. **desde** DT=1 hasta MAX_DT **hacer**
4. Inicializar y actualizar variables de iteración
5. Almacenar los electrones que están dentro del dispositivo
6. **desde** e=1 hasta Numero total de Partículas **hacer**
7. Actualizar variables de la partícula
8. Calcular las corrientes
9. Mover partícula
10. Actualizar estado de variables globales
11. **fin desde**
12. Alcanzar la neutralidad de carga en las mallas
13. Resolver la ecuación de POISSON
14. Guardar resultados
15. Actualizar variables globales
16. **fin desde**
17. Fin del Programa

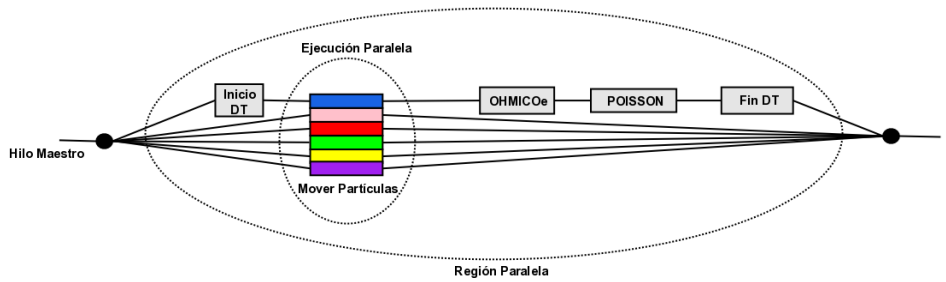
las cuales indiquen que las iteraciones del bucle principal (lineas 3-16) del programa se dividan entre el número total de núcleos disponibles, y añadir alguna otra directiva que indique si hay alguna variable que no debe ser compartida para evitar dependencias de datos.

Sin embargo, esta idea no es correcta ya que puede llegar a producir resultados erróneos. Para comprender por qué hay que considerar el código no sólo como un simple conjunto de instrucciones escritas en Fortran, sino que hay que entender el significado físico de cada una de ellas, y la parte del mundo real que tratan de simular.

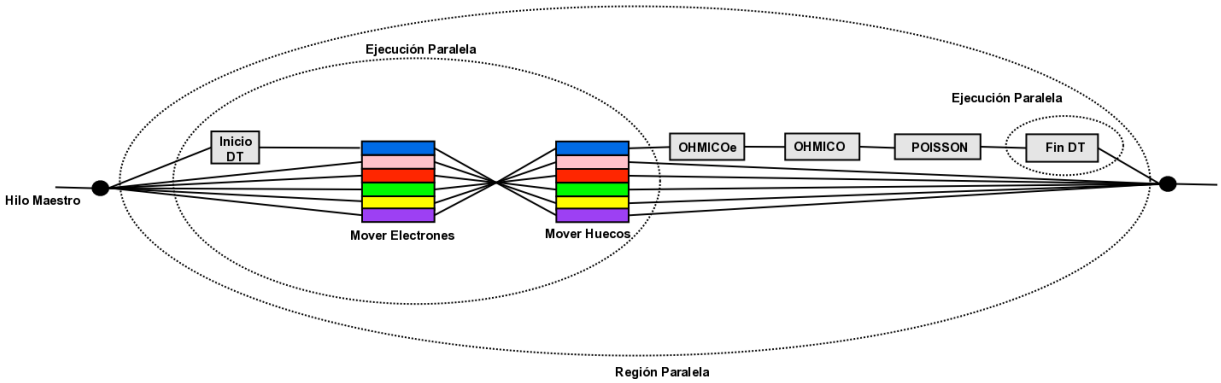
Desde una perspectiva física muy simplificada, estamos intentando estudiar las propiedades de un dispositivo que está constituido por un material, el dispositivo tiene unos límites en los que se deben imponer unas condiciones de contorno matemáticas que den cuenta de lo que ocurre en el dispositivo real en su entorno (por ejemplo, los electrodos son polarizados a ciertos valores de tensión eléctrica). Las propiedades eléctricas del dispositivo se ven alteradas por medio del movimiento que llevan a cabo las partículas que lo integran en cada instante de tiempo simulado bajo la acción del campo eléctrico en cada punto que a su vez depende de la geometría del dispositivo, del material o materiales que lo constituyen, de las condiciones de contorno y de las propias partículas simuladas. Las características eléctricas del dispositivo sufrirán una evolución a lo largo del tiempo que el código simula.

El fundamento teórico de los programas unidimensional y bidimensional son idénticos: se está simulando la evolución de un dispositivo semiconductor. Se simulan pequeños intervalos de tiempo, DT, en el cual las partículas se mueven en el dispositivo para posteriormente resolver la ecuación de Poisson. La principal diferencia entre ambos programas (unidimensional y bidimensional), es que mientras que en unidimensional sólo existe un tipo de partícula (partículas electrón), en el caso bidimensional se tratan dos: las partículas electrón y las partículas hueco.

A efectos del simulador, un hueco es entendido, al igual que un electrón, como portador de carga en un semiconductor ya que ambas partículas tienen



(a) Unidimensional (1D)



(b) Bidimensional (2D)

Fig. 1. Esquema del Simulador de Nano-Dispositivos Semiconductores Unidimensional y Bidimensional

la misma carga, pero de signo contrario. Sin embargo, en la realidad, un hueco no es un partícula real, sino que representa la ausencia de electrón en una banda de valencia del material, aunque en el trato con este tipo de materiales se habla de los huecos como si lo fueran realmente, podrían considerarse quasipartículas.

Hay que señalar que existe una dificultad añadida al código bidimensional, y es que todas las variables están en dos dimensiones. En el simulador unidimensional, se consideraba un dispositivo lineal (X), mientras que en el bidimensional se trabaja con un dispositivo en el plano (XY).

En la Figura 1 se muestran las versiones paralelas de ambos programas: Unidimensional (Figura 1(a)) y Bidimensional (Figura 1(b)).

En la paralelización de ambos programas, dado un instante de simulación DT, se lleva a cabo en paralelo la tarea con más carga computacional que es el procesamiento de las partículas (dividiendo entre los núcelos del sistema el procesamiento del movimiento de partículas); dejando el cálculo de la nueva distribución de carga y la resolución de la ecuación de Poisson de manera secuencial. El cálculo de las variables globales al inicio y al final de cada instante de tiempo ha sido únicamente paralelizado en la versión bidimensional, por su alto coste computacional en esta versión.

IV. Resultados Experimentales

En esta sección se presenta una serie de experimentos con el fin de demostrar la eficiencia obtenida en la paralelización del simulador.

Los resultados que se muestran son resultados medios basados en 30 ejecuciones realizadas variando el número de núcleos (1, 2, 4 y 8). De este modo evitaremos desviaciones en los tiempos de ejecución producidas por el carácter aleatorio del código. Las características de la máquina en la que se han realizado los experimentos se muestra en la Tabla I.

TABLA I
Características de la Máquina

Procesador	Intel Xeon X5550
Número de Núcleos	4
Número de Hilos	8
Velocidad de procesador	2.66 GHz
Velocidad Máxima Turbo	3.06 GHz
Intel Smart Caché	8 MB
Intel Turbo Boost Technology	Sí
Intel Hyper-Threading Technology	Sí
Número de procesadores	2

Por un lado, la configuración utilizada en el simulador Unidimensional (1D) es la siguiente:

- Número de intervalos de tiempo que se simularán (MAX_DT): 80000
- Número inicial de partículas en el dispositivo: 300*10,40*1,300*10,0
- Número de electrones reales que representa cada partícula electrón en la simulación: 100*1.0 E14
- Número máximo de partículas electrón en el dispositivo: 10000

Por otro lado, la configuración utilizada en el simulador Bidimensional (2D) es la siguiente:

- Número de intervalos de tiempo que se simularán (MAX_DT): 200000
- Número inicial de partículas en el dispositivo: 300*10,40*1,300*10,0

- Número de electrones reales que representa cada partícula electrón en la simulación: 100*1.0 E14
- Número máximo de partículas electrón en el dispositivo: 120000
- Número máximo de partículas hueco en el dispositivo: 120000

En la Tabla II se muestran los tiempos medios obtenidos en 30 ejecuciones secuenciales independientes de los simuladores: Unidimensional (1D) y Bidimensional (2D). Como se puede observar, el tiempo de ejecución total está dividido en dos: *región paralela* y *región secuencial*. El motivo de esta división es poder calcular el factor de aceleración máximo teórico (S_{max}) utilizando la ley de Amdhal. A la vista de los resultados mostrados en la Tabla II, se puede ver que el simulador 1D emplea 474,05s en realizar 80000 instantes de tiempo (iteraciones), mientras que la versión 2D emplea 29176,7s en realizar 200000.

TABLA II
TIEMPO MEDIO DE EJECUCIÓN PARA 1D Y 2D (SECUENCIAL)

Tiempo_{1D}	474,05s	$\text{T}_{Region\ Paralela\ 1D}$	469,59s (99,06%)
		$\text{T}_{Region\ Secuencial\ 1D}$	4,46s (0,94%)
Tiempo_{2D}	29176,7s	$\text{T}_{Region\ Paralela\ 2D}$	28448,8s (97,51%)
		$\text{T}_{Region\ Secuencial\ 2D}$	727,9s (2,49%)

En la Tabla III, se muestran los distintos factores de aceleración (máximo y medio), y la eficiencia obtenida por cada simulador (1D y 2D) utilizando diferente número de núcleos. Tal y como podemos observar, el comportamiento es muy similar tanto con el código unidimensional como con el código bidimensional, sin embargo, el código unidimensional obtiene, en general, mejores resultados de eficiencia y factor de aceleración. En la Figura 2 se muestran de manera gráfica el factor de aceleración medio obtenido con 1D y 2D.

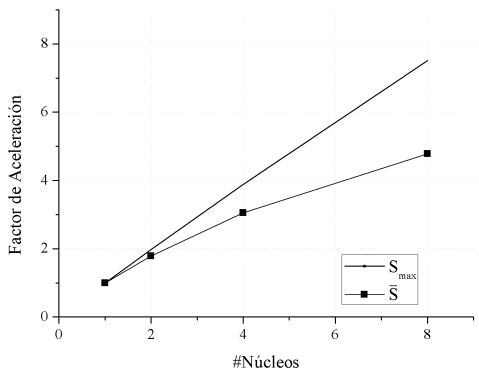
TABLA III
RESULTADOS PARA LOS SIMULADORES 1D Y 2D

Simulador 1D			
#Núcleos	S_{max}	$\overline{S}$	$\overline{E}$
1	1	1	100%
2	1,98	1,79	90,40%
4	3,89	3,05	78,41%
8	7,51	4,78	63,65%

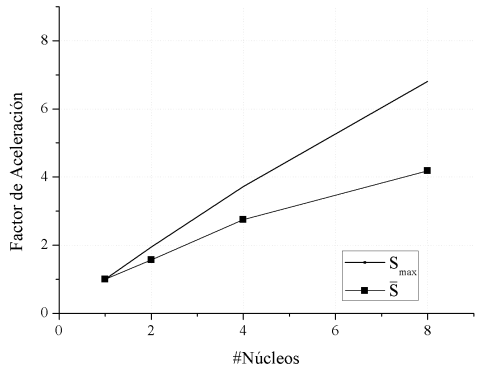
Simulador 2D			
#Núcleos	S_{max}	$\overline{S}$	$\overline{E}$
1	1	1	100%
2	1,95	1,57	80,51%
4	3,72	2,75	73,92%
8	6,81	4,18	61,38%

S_{max}: F. de Aceleración máximo teórico
$\overline{S}$: F. de Aceleración medio (30 ejec.)
$\overline{E}$: Eficiencia media (30 ejec.)

Puesto que hemos trabajado con la API de OpenMP, propia de sistemas con memoria compartida, existe riesgo de obtener resultados erróneos. El principal riesgo consiste en la sobreescritura de información porque varios hilos estén trabajando simultáneamente en la misma zona de memoria, o que la declaración de variables no permita visualizar datos necesarios para algún hilo en alguna zona del código en la que debiera verlos.



(a) Simulador 1D



(b) Simulador 2D

Fig. 2. Factor de Aceleración Máximo (S_{max}) y Medio ($\overline{S}$) para los Simuladores 1D y 2D

Para verificar que nuestra propuesta es correcta, hemos desarrollado un software que trata de analizar el ruido electrónico producido por la aletoriedad propia de Monte Carlo, es decir, el análisis de las fluctuaciones de magnitudes macroscópicas (tensión, corriente, ...) que puedan dar información de fenómenos físicos de segundo orden.

Este análisis se lleva a cabo por medio de las gráficas generadas (ver Figura 3). Puesto que las gráficas de ruido generadas por las versiones secuencial y paralela de código son prácticamente idénticas, podemos concluir que la optimización se ha llevado a cabo con éxito.

Por último, vamos a comparar los resultados obtenidos por nuestros simuladores paralelos 1D y 2D, con aquellos publicados en la literatura. Los resultados de esta comparación son meramente orientativos puesto que no se trata de la optimización del mismo código.

En [6], los autores llevan a cabo el análisis de una implementación paralela de un simulador de Monte Carlo tridimensional en el que se se emplean estruc-

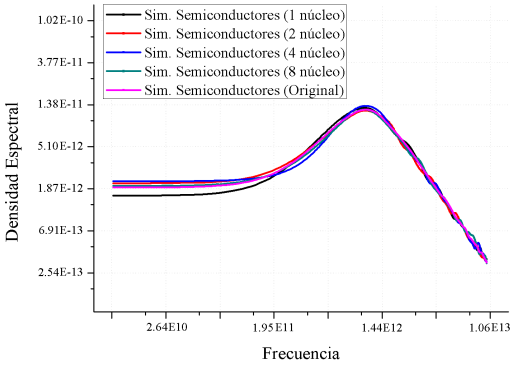


Fig. 3. Análisis del ruido electrónico producido por la aleatoriedad propia de Monte Carlo

turas de datos complejas y asignación dinámica de memoria. El simulador, mediante el método Monte Carlo, resuelve, al igual que este trabajo, la ecuación de Boltzmann para flujos de gases en el régimen de transición. La herramienta de optimización empleada es también OpenMP. En la Figura 4, se muestra una comparativa en términos de factor de aceleración obtenidas por nuestros simuladores semiconductores (1D y 2D), y el simulador de fluidos (3D) propuesto en [6].

Las conclusiones que pueden extraerse a la vista de esta comparativa son las siguientes:

- Diferencia en el hardware de las máquinas. Los autores del trabajo citado no disponen de las teconologías Intel Turbo Boost e Hyper-Threading de las que disponen nuestras máquinas.
- Complejidad del código. Al aumentar las dimensiones del simulador se va perdiendo rendimiento incluso para nuestro propio código. Al trabajar con datos pesados las eficiencias no escalan bien.

Como conclusión de esta comparativa, podemos señalar que la tendencia de la Figura 4 indica que si siguiéramos aumentando el número de núcleos disponibles (con la misma arquitectura de la máquina) el speedup seguiría teniendo la misma

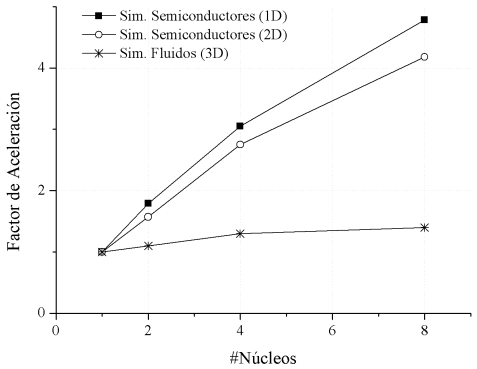


Fig. 4. Comparativa con otros autores

pendiente lineal positiva en los simuladores de semiconductores, mientras que se estancaría para el simulador de Fluidos.

V. Conclusiones y Trabajo Futuro

En este documento se ha propuesto la optimización de un producto software para el estudio de nanodispositivos semiconductores por medio de simuladores de Monte Carlo. Como ha podido comprobarse, tras aplicar un algoritmo de paralelización con la API de OpenMP se han conseguido unos rendimientos muy prometedores (superiores al siempre al 60%).

Como trabajo futuro, surge la idea inmediata de poder plasmar los resultados obtenidos en una versión tridimensional de este mismo algoritmo simulador de Monte Carlo para dispositivos semiconductores.

Otra línea abierta es la implementación de un algoritmo de paralelización para este mismo producto software que permita una mayor portabilidad del código: implementar nuestro algoritmo por medio de una hibridación OpenMP + MPI, dando la posibilidad de portar el programa a sistemas híbridos de memoria compartida y distribuida.

Agradecimientos

Álvaro Rubio-Largo es becario de investigación (PRE09010) del Gobierno de Extremadura (Consejería de Economía, Comercio e Innovación) y el Fondo Social Europeo (FSE). Por último, agradecer al Gobierno de Extremadura la ayuda GR10025 concedida al grupo de investigación TIC015.

Referencias

[1] Federico Faggin, Marcian E. Hoff Jr., Stanley Mazor, and Masatoshi Shima, "The History of the 4004," *IEEE Micro*, vol. 16, pp. 10–20, 1996.

[2] P. Dollfus, S. Galdin, P. Hesto, and J.E. Velazquez, "Monte carlo study of sub-0.1 mu;m si0.97c0.03/si modfet: electron transport and device performance," *Electron Devices, IEEE Transactions on*, vol. 47, no. 6, pp. 1247 –1250, jun 2000.

[3] M. J. Martín, D. Pardo, and J. E. Velázquez, "Analysis of voltage noise in forward-biased silicon bipolar homojunctions: Low- and high-injection regimes," *Applied Physics Letters*, vol. 71, no. 23, pp. 3382–3384, 1997.

[4] Philippe Dollfus, "Si/si1-xgex heterostructures: Electron transport and field-effect transistor operation using monte carlo simulation," *Journal of Applied Physics*, vol. 82, no. 8, pp. 3911 –3916, oct 1997.

[5] Florian Monsef, Philippe Dollfus, Sylvie Galdin-Retailleau, Hans-Joest Herzog, and Thomas Hackbarth, "Electron transport in si/sige modulation-doped heterostructures using monte carlo simulation," *Journal of Applied Physics*, vol. 95, no. 7, pp. 3587 –3593, apr 2004.

[6] Da Gao and Thomas E Schwartzentruber, "Optimizations and OpenMP implementation for the direct simulation Monte Carlo method," *Computers and Fluids*, vol. 42, no. 1, pp. 73–81, 2011.