

– Prix Bull-Fourier 2012 –

Le code TB_Sim

Yann-Michel Niquet
SP2M/L_Sim, INAC, CEA Grenoble

François Triozon
LETI-MINATEC, CEA Grenoble

Christophe Delerue
IEMN/ISEN, UMR CNRS 8520, Lille

Nous présentons un code de simulation pour les nanosciences et les nanotechnologies, TB_Sim¹, adapté au calcul hautes performances (HPC). Nous introduisons le contexte du développement de ce code, les modèles physiques qu'il implémente et ses fonctionnalités au paragraphe I. Nous décrivons brièvement l'ensemble des résultats scientifiques obtenus avec ce code au paragraphe II, puis détaillons un exemple, en insistant sur les optimisations réalisées pour le HPC. Enfin, nous exposons au paragraphe III les développements effectués cette année sur la partie la plus avancée du code, le module GPU « fonctions de Green » pour la modélisation des composants micro- et nano-électroniques. Nous montrons comment les GPUs permettent de gagner un facteur ~ 25 sur le temps de rendu, moyennant une implémentation et une gestion des données appropriée, et décrivons les perspectives d'utilisation de ce code sur les calculateurs hybrides du CCRT et du TGCC.

I. Le code TB_Sim.

I.1. Contexte.

Les nanosciences et les nanotechnologies désignent l'ensemble des savoirs et procédés relatifs aux objets naturels ou artificiels dont les dimensions caractéristiques sont de l'ordre de quelques nanomètres ($1 \text{ nm} = 10^{-9} \text{ m} = 0.001 \mu\text{m}$). Elles suscitent un grand intérêt depuis deux décennies pour leurs applications potentielles dans la micro-électronique, l'énergie (photovoltaïque...) ou la médecine. En micro-électronique par exemple, la dimension caractéristique des transistors sur un processeur est passée de 90 à 22 nm en 10 ans, et les industriels préparent actuellement le « nœud » 16 nm à venir d'ici 2015. A cette échelle, le « canal » du transistor dans lequel circulent les électrons ressemblera à un « nanofil » de silicium dont les dimensions seront de l'ordre de 10 nm (Figure 1). Réduire ainsi la taille des transistors permet d'en intégrer toujours plus sur une même puce, et d'accroître ainsi ses fonctionnalités et ses performances (apparition des processeurs et cartes graphiques multi-cœurs, des mémoires RAM et flash de grandes capacités, etc...). Cette évolution s'est accompagnée d'une complexification des dispositifs, avec l'introduction de nouveaux matériaux (oxydes d'Hafnium HfO_2 , siliciures, etc...). Cette tendance va se poursuivre, avec des composants de plus en plus originaux (à « nanotubes de carbone », « isolants topologiques », ...) qui pourraient à terme remplacer le transistor silicium et qui sont actuellement explorés dans les laboratoires.

Dans le domaine de l'énergie, les nanosciences ont notamment des applications dans le photovoltaïque, où elles permettent d'augmenter le rendement des cellules solaires.

La modélisation et la simulation ont un rôle important à jouer dans le développement des nanosciences et des nanotechnologies. D'une part, la caractérisation complète des nanostructures est souvent impossible, si bien qu'il peut être difficile de démêler différents effets physiques. L'interprétation des expériences repose alors sur des conjectures invérifiables, ou sur des modèles macroscopiques qui ne sont plus valables à l'échelle nanométrique. Une simulation réaliste peut permettre de quantifier chaque effet et d'affiner l'interprétation d'expériences ambiguës. Le code TB_Sim a souvent contribué de façon décisive à la compréhension d'expériences menées sur des

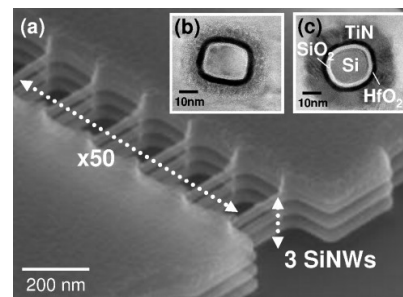


Fig. 1 : Un transistor à nanofils (« SiNWs »). Une électrode (appelée « grille ») déposée autour des fils permet de contrôler le courant qui les traverse (Image CEA/LETI).

¹ http://inac.cea.fr/L_Sim/TB_Sim/index.html

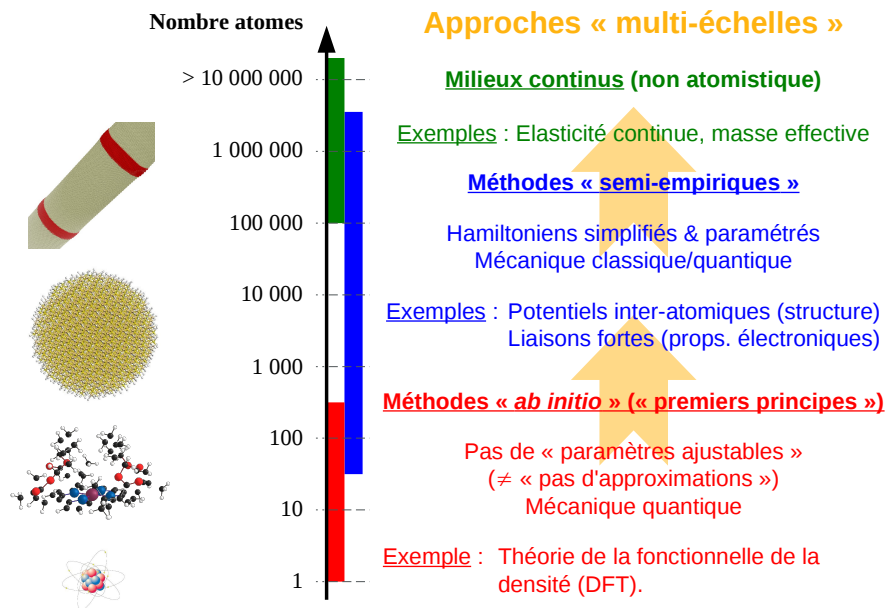


Fig. 2 : Hiérarchie des méthodes numériques en nanosciences.

nanostructures complexes, comme en témoignent les nombreuses publications réalisées avec des groupes expérimentaux [1-9]. Mieux encore, la simulation peut mettre en évidence des phénomènes nouveaux avant même que ceux-ci n'aient été observés, et contribuer ainsi au renouvellement des nanosciences. Nous présenterons au paragraphe II quelques exemples de simulations *prédictives* qui ont permis d'anticiper une physique originale. Enfin, la simulation des nanotechnologies doit permettre de cribler les options et d'optimiser les dispositifs, donc d'orienter des campagnes de caractérisation expérimentale souvent coûteuses.

Relever ces défis nécessite le développement de codes de simulation adaptés, et suffisamment prédictifs pour pouvoir être emmenés sur des terrains inexplorés. La vocation de tels codes n'est pas nécessairement de faire de la modélisation « ordinaire », mais de résoudre les problèmes les plus avancés, ce qui justifie le recours à des moyens de calcul exceptionnels tels que ceux des centres de calculs nationaux et européens (CCRT, TGCC...).

Le code TB_Sim a été développé dans ce but et avec cet esprit. Avant de le présenter en détail, nous allons faire un bref panorama des méthodes de calcul pour les nanosciences, en insistant sur la nécessité d'une approche intégrée ou « multi-échelle » capable de combiner les forces de chacune d'entre elles.

1.2. Les méthodes de calcul pour les nanosciences.

En nanosciences, la physique classique cède souvent la place à la « mécanique quantique » qui seule décrit un certain nombre de phénomènes spécifiques à l'échelle du nanomètre. Il existe deux grandes familles de méthodes quantiques pour le calcul des propriétés structurales et électroniques des nanostructures (Fig. 2) :

– Les méthodes « premiers principes » ou méthodes « *ab initio* » : Ces méthodes cherchent à résoudre le problème original – un ensemble d'atomes en interaction dans une nanostructure – sans faire d'hypothèse préalable sur la nature de ces interactions (mais pas sans faire

d'approximations). La théorie de la fonctionnelle de la densité (« DFT ») est la plus connue de ces méthodes. La DFT permet de prédire les propriétés structurales (arrangement des atomes) des molécules et nanostructures avec précision. Elle est néanmoins très coûteuse numériquement, et ne permet pas de traiter des systèmes de plus de ~ 1000 atomes même avec des moyens de calculs type HPC. En outre, la DFT prédit mal les « excitations » de ces systèmes (réponse à des champs électromagnétiques, etc...), ce qui limite son intérêt pour le calcul des propriétés optiques ou des courants électriques par exemple. Les méthodes « post-DFT », qui corrigent ces carences, sont encore plus coûteuses et ne peuvent donc pas être appliquées à des nanostructures complexes. Les méthodes *ab initio* restent néanmoins la référence au niveau « macromoléculaire ».

– Les méthodes semi-empiriques : Ces méthodes utilisent des modèles effectifs plus simples mais capables de reproduire la physique la plus importante du système étudié. L'approximation de la masse effective est l'une des méthodes semi-empiriques les plus utilisées pour décrire les propriétés électroniques des semiconducteurs. Elle substitue au réseau atomique un milieu continu caractérisé par la masse « effective » des électrons, différente de leur masse dans le vide. Les équations de la mécanique quantique peuvent alors être résolues avec des méthodes de différences/éléments finis, et le coût du calcul est, en première approximation, indépendant de la taille du système (puisque'il n'y a plus de relation aux atomes). Toutefois, la masse effective n'est pas toujours suffisamment précise dans des nanostructures de moins de 10 nm, et décrit mal les détails de nature atomique tels que des impuretés ou des molécules greffées sur une surface. La méthode des liaisons fortes permet de jeter un pont entre le monde *ab initio* et celui des milieux continus. Elle consiste à développer les équations de la mécanique quantique dans une base constituée par les « orbitales » de chaque atome (les solutions pour des atomes isolés). Cette base est idéale pour décrire la liaison chimique. Les interactions entre orbitales sont préalablement tabulées sur des calculs *ab initio* (souvent post-DFT) effectués sur des systèmes (simples) de référence, puis transférées aux nanostructures. Cette étape de paramétrisation permet de réduire considérablement le coût du calcul et de modéliser des nanostructures comprenant plusieurs millions d'atomes avec une résolution atomique. Elle nécessite bien sûr une bonne compréhension des limites de l'exercice (comme pour toutes les méthodes, même *ab initio*). La méthode des liaisons fortes constitue sans doute le meilleur compromis précision/efficacité dans toute la gamme de dimensions 2-10 nm.

Un code de simulation pour les nanosciences et nanotechnologies doit pouvoir tirer parti du meilleur des méthodes ci-dessus et les combiner si besoin. C'est l'un des objectifs du code TB_Sim.

1.3. Le code TB_Sim.

TB_Sim est à l'origine un code de « liaisons fortes » dont le développement a été initié au CEA par Y. M. Niquet en 2004 afin de relever les défis présentés par la modélisation des nanotechnologies. Il permet de calculer les propriétés électroniques, optiques (photovoltaïque, ...) et les propriétés de transport de charges (courants électriques) de nanostructures de matériaux aussi variés que les semiconducteurs usuels (Si, GaAs, ...), les nanotubes de carbone ou les rubans de « graphène ». L'objectif poursuivi était (et reste) de développer un outil de simulation adapté aux calculateurs parallèles haute performance, capable de résoudre les problèmes les plus avancés. Un soin particulier a donc été apporté à sa parallélisation (OpenMP, MPI, ...), puis à l'utilisation d'accélérateurs de type GPUs sur les architectures hybrides. Depuis 2004, le code s'est enrichi de nombreuses fonctionnalités, avec en particulier l'introduction des méthodes de type masse effective/**k.p**, et d'une interface avec le code *ab initio* SIESTA², qui permettent à TB_Sim de couvrir

² <http://www.icmab.es/dmmis/leem/siesta/>

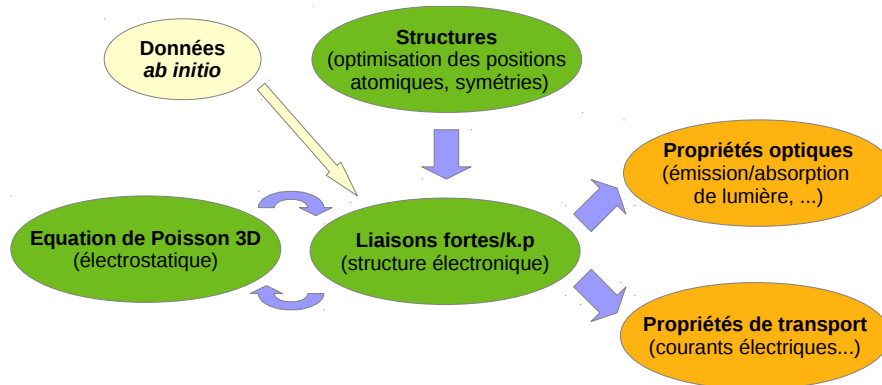


Fig. 3 : Principales fonctionnalités du code TB_Sim.

toutes les échelles du nanomètre au micromètre. Les principales fonctionnalités du code sont (cf. Fig. 3) :

- Propriétés électroniques avec la méthode des liaisons fortes (modèles atomistiques) ou en masse effective/**k.p** (milieux continus en différences/éléments finis).
- Interface avec le code *ab initio* SIESTA.
- Relaxation structurale (optimisation des positions atomiques) avec des champs de forces classiques.
- Champs électriques et magnétiques : Résolution de l'équation de Poisson en différences finies & solutions semi-analytiques pour les géométries simples.
- Réponse linéaire (fonctions de réponses « RPA »).
- Calcul des propriétés optiques : « excitons » en interaction de configurations, temps de vie radiatif, etc...
- Calcul des propriétés de transport de charges : Equation de Boltzmann (méthode semi-classique), méthode de Kubo quantique et fonctions de Green.

Le code intègre de nombreux solveurs « standard » (gradients conjugués, GMRES,... pour les systèmes linéaires, Jacobi-Davidson pour les problèmes aux valeurs propres, etc...). Nombre de ces solveurs ont été réécrits pour TB_Sim afin de pouvoir y introduire des optimisations spécifiques (traitement « à la volée » des symétries spatiales et temporelles par exemple).

TB_Sim propose plusieurs niveaux de parallélisation (OpenMP, MPI avec distribution des tâches *a priori* ou sur un mode « maître/esclave » en fonction des problèmes traités).

A ce jour, TB_Sim se présente essentiellement comme une ensemble de « bibliothèques » (modules Fortran et C/C++) traitant chacune une tâche spécifique (modèles de liaisons fortes, maillages différences/éléments finis, solution des systèmes linéaires ou des équations aux valeurs propres, etc...) partageant des structures de données communes leur permettant d'échanger des informations tout en limitant au maximum les dépendances. Les « codes » (en général assez courts) basés sur TB_Sim appellent ces bibliothèques pour effectuer les opérations dont ils ont besoin. Le pre- et le post-processing sont généralement confiés au langage Python, et un effort a été engagé pour pouvoir appeler les principales fonctionnalités de TB_Sim directement depuis des scripts Python afin d'en faciliter l'utilisation (« binding »).

Depuis 2004, TB_Sim est développé sous une licence propriétaire CEA. Le code est

néanmoins ouvert à un certain nombre de partenaires français et européens (IEMN à Lille, UCL en Belgique, ICN à Barcelone...). Une grande partie du code (à l'exception peut-être des fonctionnalités les plus avancées) sera libérée à la fin de l'année 2012 sous licence GNU GPL. Nous travaillons actuellement à la documentation et à la mise en place des infrastructures (sites web, ...) nécessaires à l'animation d'une « communauté » autour de TB_Sim. L'objectif est de permettre un large accès à ce code, et de fédérer de nouveaux développeurs, afin d'accélérer l'introduction de nouvelles fonctionnalités et d'outils tels que des interfaces graphiques.

– Fiche d'identité du code –

Nom : TB_Sim.

Langages utilisés : Fortran 95/2003, C/C++, CUDA, Python.

Taille du code : Modules & librairies ~ 175 000 lignes.

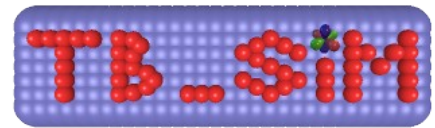
Codes ~ 60 000 lignes.

Librairies extérieures : Intel MKL, cuBLAS, MAGMA.

Parallélisation : OpenMP, MPI, CUDA.

Taille de job typique : de 1 à 1024 cœurs.

Production scientifique : 40 papiers et 12 actes de conférence (dont 6 Nano Letters, 5 Physical Review Letters, 12 Physical Review B, 5 IEEE journals, 2 Applied Physics Letters, ...)



II. Résultats scientifiques.

II.1. Rétrospective.

TB_Sim a permis de répondre à nombre de questions d'intérêt expérimental, sur des sujets très variés :

- Propriétés structurales et optiques des nanostructures de semiconducteurs [2-11], en lien direct avec les expériences menées sur ces objets. TB_Sim a par exemple permis d'expliquer le décalage vers le rouge de la lumière émise par des fils de nitrure de gallium (GaN), dû à l'apparition de champs électriques internes [2]. Les simulations se sont révélées suffisamment fines pour suggérer que les fils n'avaient pas l'orientation cristalline attendue, ce qui fut confirmé expérimentalement.

- Propriétés de transport des nanofils [12-19] : TB_Sim a permis de mieux comprendre les forces et les faiblesses des technologies nanofils pour la micro-électronique (voir détails ci-dessous). Nous avons proposé (et quantifié) des solutions pour améliorer les performances électriques des nanofils.

- Propriétés de transport des nanotubes et du graphène [20-24] : Nous avons calculé les propriétés de transport de ces matériaux à base de carbone qui suscitent un énorme engouement aujourd'hui pour leurs applications potentielles en micro-électronique (composants haute fréquence, électrodes transparentes, etc...). Un soin particulier a été apporté à la description des défauts et du greffage moléculaire dans ces structures.

TB_Sim a donc fait la démonstration de sa pertinence et de sa capacité à résoudre des problèmes complexes. La combinaison de méthodes atomistiques (liaisons fortes) et de méthodes de milieux continus lui permet de traiter des systèmes couvrant plusieurs échelles de longueurs différentes (comme par exemple dans la Ref. [2]). Plus important encore, TB_Sim a permis de mettre en évidence par la simulation des phénomènes nouveaux, qui ont ensuite été observés expérimentalement :

– En 2006 nous avons prédit que les impuretés (atomes en substitution) auraient un comportement électrique très différent dans les nanofils de semiconducteurs et dans les matériaux massifs [25-26]. Ces prédictions ont fait l'objet de vifs débats, mais ont été confirmées expérimentalement par IBM Zürich³. Les comparaisons entre TB_Sim et codes *ab initio* ont en outre permis de faire des avancées méthodologiques importantes dans la modélisation des défauts chargés [27].

– L'été dernier, nous avons prédit que le modèle « standard » universellement accepté pour le potentiel de confinement dans les « hétérostructures » de semiconducteurs (empilements de matériaux différents) n'était pas inconditionnellement valable dans les systèmes unidimensionnels tels que les nanofils [28]. Nous avons proposé de nouveaux dispositifs pour le photovoltaïque exploitant cette physique originale.

A titre d'illustration, nous discutons ci-dessous quelques travaux récents sur les propriétés de transport des nanofils de silicium, en justifiant notamment le besoin de simulations atomistiques, et en décrivant le travail d'optimisation effectué sur les codes.

II.2. Exemple : Propriétés de transport des fils de silicium contraints.

Comme nous l'avons expliqué dans l'introduction, les transistors évoluent d'une architecture planaire vers une architecture « 3D » de type « FinFET » ou « nanofils » avec des dimensions latérales de l'ordre de la dizaine de nanomètres. Il convient donc de s'intéresser aux propriétés de transport électronique de ces objets, afin d'appréhender les forces et faiblesses de ces nouveaux dispositifs.

Les propriétés de transport d'un nanofil peuvent être caractérisées par la « mobilité » μ , qui relie la vitesse moyenne des électrons $\langle v \rangle$ au champ électrique E appliqué le long du fil : $\langle v \rangle = \mu E$. Plus la mobilité est élevée, plus les électrons vont vite et plus le courant est important.

La mobilité dans un nanofil est limitée par les « défauts » éventuels de ce nanofil (impuretés, ...) et par les vibrations des atomes (« phonons ») qui dévient les électrons de leur trajectoire. Ce dernier mécanisme est très efficace dans les nanofils où le confinement des électrons et des vibrations renforce leurs interactions. La modélisation du couplage électrons/phonons dans des fils de 10 nm et moins nécessite une approche atomistique seule capable de rendre compte de la structure électronique et vibrationnelle très compliquée de ces objets.

Nous avons récemment calculé la mobilité dans les nanofils de silicium en résolvant « l'équation de Boltzmann » [13]. Nous avons démontré que la mobilité diminue effectivement avec le diamètre des nanofils dans des proportions inquiétantes (Fig. 4). Les résultats expérimentaux confirment cette tendance, mais sont encore très dispersés car il est souvent difficile de « faire la part » entre différents effets à cette échelle. Ce calcul a donc le mérite d'apporter des éléments quantitatifs et d'élucider les mécanismes responsables de la dégradation de la mobilité dans les nanofils.

Fort de cette analyse, nous avons pu proposer des solutions pour limiter cette dégradation, voir inverser la tendance. Nous avons notamment démontré que la mobilité pouvait être multipliée par un facteur 2 à 5 en « tirant » de façon modérée sur les nanofils [12]. Il existe des artifices technologiques permettant d'appliquer de telles contraintes à un nanofil. Des contacts ont été pris

³ M. T. Björk, H. Schmid, J. Knoch, H. Riel et W. Riess, Nature Nanotechnology **4**, 103 (2009).

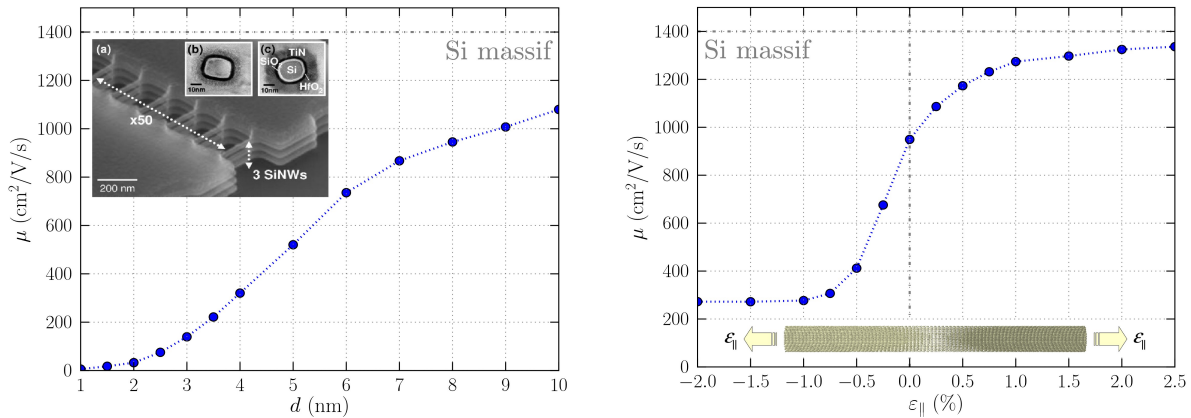


Fig. 4 : A gauche, mobilité dans un nanofil de silicium cylindrique en fonction de son diamètre d . La ligne horizontale $\mu = 1400 \text{ cm}^2/\text{V}\cdot\text{s}$ est la mobilité dans le silicium massif. **A droite**, mobilité dans un nanofil de silicium de 8 nm de diamètre étiré le long de son axe. $\epsilon_{||}$ est l'allongement (en %) du fil. On retrouve des mobilités comparables à celles du matériau massif pour des allongements de l'ordre de 1%.

avec le CEA/LETI et ST Microelectronics à ce sujet. Ces calculs illustrent parfaitement l'intérêt de la simulation numérique pour les nanotechnologies, en particulier lorsqu'elle est capable d'anticiper comme ici sur la caractérisation expérimentale.

Ce travail est le fruit de nombreux développements méthodologiques et numériques :

- Nous avons construit en 2009 des modèles atomistiques pour le silicium capables de rendre compte de ses propriétés électroniques pour des déformations (et donc des vibrations) arbitraires [29].
- Nous avons développé un premier code en 2010, avec lequel nous avons dégagé des tendances pour des petits fils de diamètres $d < 5 \text{ nm}$ [15].
- Profitant de l'expérience ainsi acquise, nous avons développé un nouveau code en 2011 avec une vraie perspective HPC. Ce code nous a permis d'étudier des fils de 10 nm de diamètre sur CCRT/Titane, malgré la complexité ($\propto d^6$) très défavorable du problème [13].

Les optimisations introduites dans ce nouveau code sont les suivantes (cf. Fig. 5) :

- Sur les algorithmes utilisés, en particulier pour l'interpolation des modes électroniques et vibrationnels dans l'espace réciproque.
- Sur la parallélisation MPI du code en mode « maître/esclave » : Le processus maître distribue sur requête les différents modes de vibration (typiquement quelques millions) à étudier aux processus esclaves, et collecte les résultats. Ce modèle de parallélisation permet d'assurer des performances quasi-optimales sur plusieurs centaines de cœurs (le calcul étant inhomogène, un mode de vibration prenant de quelques secondes à quelques minutes à traiter). Le code a été exploité sur 32 à 768 cœurs.
- Sur l'optimisation SSE du code qui calcule le couplage entre électrons et phonons. Nous avons en particulier réécrit avec le jeu d'instructions SSE3 du C des produits matrice réelle/vecteur complexe⁴, qui représentent 95% du temps de calcul et qui ne sont pas correctement

⁴ L'opérateur de couplage entre électrons et phonons peut être représenté par une matrice dont la dimension est $N_{\text{orb}}N_{\text{at}}$, où N_{at} est le nombre d'atomes et $N_{\text{orb}} = 10$ le nombre de degrés de liberté (« orbitales ») par atome. Seuls certains sous-blocs 10×10 de cette matrice sont non nuls. Les opérations sur ces matrices 10×10 représentent 95% du temps

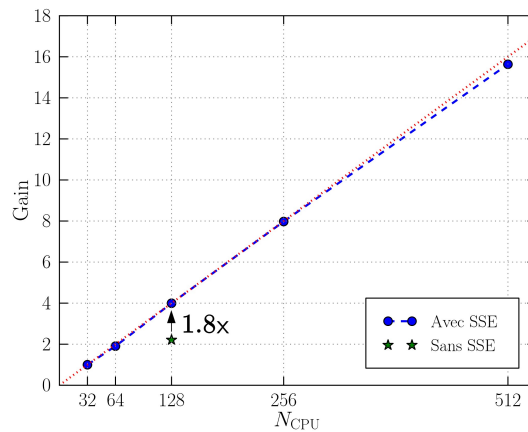


Fig. 5 : Performances du code de calcul du couplage électrons-phonons en fonction du nombre N_{CPU} de cœurs (pour un fil de silicium de 4 nm de diamètre). Les points bleus représentent le gain sur le temps de rendu (par rapport à la référence : $T_{wall} = 3h59mn$ @ $N_{CPU} = 32$). La ligne pointillée rouge représente la loi d'échelle idéale. Le code est parallélisé sur un mode « maître/esclave » (le processus maître distribue sur requête les tâches aux processus esclaves et collecte les résultats). Remarquez la très bonne linéarité du code. Les optimisations SSE3 permettent de gagner un facteur ~ 2 sur le temps de calcul. Mesures effectuées sur CCRT/Titane.

optimisés par les compilateurs. Cette opération nous a permis de gagner un facteur presque 2 sur le temps de rendu, et de réaliser les études de la figure 4 dans un délai raisonnable (coût total des Refs. [12] & [13] : $\sim 200\,000$ heures sur CCRT/Titane).

III. Le code de fonctions de Green hors équilibre sur GPU.

Nous allons maintenant détailler les travaux effectués cette année sur les modules les plus avancés du code, qui implémentent la méthode des fonctions de Green hors équilibre sur GPU.

III.1. La méthode des fonctions de Green hors équilibre.

La méthode des fonctions de Green permet de calculer (en mécanique quantique) les propriétés de transport de systèmes « ouverts » (dans lesquels les électrons peuvent rentrer et sortir) mis hors-équilibre (en appliquant une différence de potentiel à leurs extrémités par exemple).

Un système quantique peut être complètement caractérisé par son Hamiltonien H , représenté par une matrice qui couple entre eux tous les degrés de liberté du système (par exemple les orbitales atomiques en liaisons fortes ou les différents nœuds du maillage de différences/éléments finis en masse effective).

La fonction de Green d'un Hamiltonien H est son inverse $G(z) = [zI - H]^{-1}$, où z est un nombre complexe et I l'identité. La fonction de Green contient, comme l'Hamiltonien, toute l'information relative aux propriétés électroniques du système. Elle est toutefois mieux adaptée que ce dernier à la description des systèmes ouverts.

de calcul, et doivent donc être soigneusement optimisées. Nous avons en particulier complètement déroulé ces opérations et nous les avons réécrites avec le jeu d'instructions SSE3 du C.

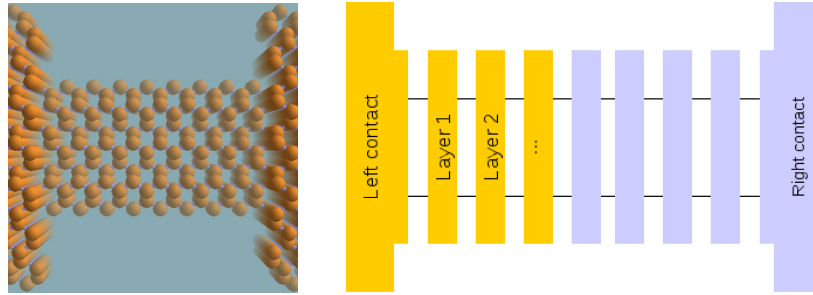


Fig. 6 : La méthode des fonctions de Green récursives. Le système (à gauche, un nanofil relié à deux contacts) est découpé en tranches successives qui ne sont couplées qu'à leurs plus proches voisines. La fonction de Green est calculée tranche par tranche.

Le formalisme des fonctions de Green est très polyvalent : Il reproduit naturellement tous les effets quantiques observables dans les dispositifs ultimes (quantification des états électroniques, effet tunnel, etc...) et peut décrire le couplage aux degrés de libertés externes au système électronique, tels que les vibrations du réseau atomique.⁵

Ces qualités ont, bien sûr, un prix. Dans un fil de rayon R et de longueur L par exemple, le nombre de degrés de liberté électroniques (orbitales atomiques ou points de maillage), $N \propto R^2 L$, est proportionnel au volume du fil. Le coût de l'inversion « brutale » d'un Hamiltonien de dimension N avec LAPACK (via une décomposition LU) croît comme N^3 , soit comme $R^6 L^3$. La taille de la matrice G est par ailleurs proportionnelle à $N^2 \propto R^4 L^2$. La complexité du problème est donc très défavorable.

Toutefois, nous pouvons profiter du fait que :

- La matrice Hamiltonien, exprimée dans une base de l'espace réel telle que des orbitales atomiques ou des éléments finis est « creuse » : Seuls des orbitales ou nœuds du maillage voisins sont couplés entre eux. Une fraction seulement des éléments de H sont donc non nuls.
- Nous n'avons en pratique besoin que de certains éléments de la fonction de Green : En particulier sa diagonale, qui permet de calculer la densité d'électrons dans le système, et les éléments couplant chaque tranche aux deux extrémités du fil, qui permettent de calculer le courant.

De nombreuses méthodes numériques ont été développées ces 25 dernières années pour inverser partiellement des matrices creuses. Plusieurs d'entre elles sont implémentées dans TB_Sim [14, 16, 28] : Méthode des fonctions de Green « récursives », « knitting », ... Nous nous focaliserons sur la méthode des fonctions de Green récursives, qui fait référence aujourd'hui dans le domaine du transport quantique et qui nous a semblé la mieux adaptée à une implémentation sur GPU.

III.2. L'algorithme des fonctions de Green récursives.

Bien que la méthode puisse être appliquée à d'autres géométries, nous nous limiterons, par souci de simplicité, au cas particulier d'un fil (représenté schématiquement sur la Fig. 6). Nous décrivons ci-dessous les principales étapes de l'algorithme, dans sa version la plus simple (sans couplage aux phonons).

⁵ La méthode des fonctions de Green diffère de la méthode « équation de Boltzmann » utilisée pour calculer les mobilités discutées au paragraphe II.2. Elle est beaucoup plus générale et précise, notamment en régime hors-équilibre (fortes polarisations).

Ce fil peut être découpé en n tranches successives (ici des plans atomiques) qui ne sont couplées qu'entre premières voisines. L'Hamiltonien H a donc la forme suivante :

$$H = \begin{pmatrix} H_{11} & t_{12} & 0 & & \\ t_{21} & H_{22} & t_{23} & 0 & \\ 0 & t_{32} & H_{33} & \ddots & 0 \\ & 0 & \ddots & \ddots & \\ & & 0 & & H_{nn} \end{pmatrix} \quad G(z) = \begin{pmatrix} G_{11} & & & & G_{1n} \\ G_{21} & G_{22} & & & G_{2n} \\ G_{31} & & G_{33} & & G_{3n} \\ \vdots & & & \ddots & \vdots \\ G_{n1} & & & & G_{nn} \end{pmatrix}$$

Fig. 7 : Forme de l'Hamiltonien H et de la fonction de Green $G(z)$. Chaque ensemble de lignes et de colonnes, délimité par une ligne pointillée, correspond à une tranche du système. Seuls les blocs gris de H sont non nuls. Tous les blocs de $G(z)$ sont *a priori* non nuls, mais seuls les blocs gris sont nécessaires au calcul des propriétés de transport du système.

Nous appellerons $H_{i,i}$ les blocs diagonaux de H , et $t_{i,i+1} = t_{i+1,i}^\dagger$ les blocs non-diagonaux qui couplent entre elles les tranches. A noter que les blocs diagonaux $H_{1,1}$ et $H_{n,n}$ de la première et de la dernière tranche contiennent une correction de « self-énergie » qui décrit le couplage de ces tranches aux contacts (conditions aux limites ouvertes vers le milieu extérieur). Nous y reviendrons brièvement au paragraphe III.4.

Nous voulons maintenant calculer i) tous les blocs diagonaux de la fonction de Green $G(z)$, et ii) tous les blocs couplant chaque tranche aux deux extrémités du fil. Nous allons procéder de façon récursive, tranche par tranche.

Nous commençons par calculer la fonction de Green $g_{1,1}^{(1)} = [zI - H_{1,1}]^{-1}$ du système constitué par la seule tranche N°1. Ensuite, nous calculons les blocs de la fonction de Green $g^{(2)}$ du système constitué par les tranches 1 et 2 réunies. Un peu d'algèbre linéaire montre que :

$$\begin{aligned} g_{2,2}^{(2)} &= [zI - H_{2,2} - t_{2,1} g_{1,1}^{(1)} t_{1,2}]^{-1} \\ g_{2,1}^{(2)} &= g_{2,2}^{(2)} t_{2,1} g_{1,1}^{(1)} \end{aligned}$$

On procède ainsi tranche après tranche. Les relations de récurrence pour le système constitué par les i premières tranches s'écrivent :

$$\begin{aligned} g_{i,i}^{(i)} &= [zI - H_{i,i} - t_{i,i-1} g_{i-1,i-1}^{(i-1)} t_{i-1,i}]^{-1} \\ g_{i,1}^{(i)} &= g_{i,i}^{(i)} t_{i,i-1} g_{i-1,1}^{(i-1)} \end{aligned}$$

Une fois les n tranches ajoutées et le système reconstruit dans son ensemble, nous obtenons les blocs $G_{n,n} = g_{n,n}^{(n)}$ et $G_{n,1} = g_{n,1}^{(n)}$ de la fonction de Green. A ce stade (que nous appellerons passe « avant ») nous n'avons calculé que deux blocs de $G(z)$.

Nous pouvons calculer tous les blocs manquants en effectuant une passe « arrière » depuis l'avant-dernière jusqu'à la première tranche. Les relations de récurrence de cette passe arrière

s'écrivent :

$$\begin{aligned} G_{i,i} &= g_{i,i}^{(i)} + g_{i,i}^{(i)} t_{i,i+1} G_{i+1,i+1} t_{i+1,i} g_{i,i}^{(i)} \\ G_{i,1} &= g_{i,1}^{(i)} + g_{i,i}^{(i)} t_{i,i+1} G_{i+1,1} \\ G_{i,n} &= g_{i,i}^{(i)} t_{i,i+1} G_{i+1,n} \end{aligned}$$

Notez comment la passe arrière réutilise les blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ calculés lors de la passe avant. Ceux-ci doivent donc avoir été stockés en mémoire ou sur disque.

Nous allons maintenant discuter la complexité de l'algorithme ci-dessus. Le nombre de degrés de libertés M dans chaque tranche est proportionnel à la surface du fil, donc à R^2 . Chaque étape (une tranche) de la passe avant ou arrière comprend une inversion et/ou des multiplications de matrices de taille $M \times M$, dont le coût est proportionnel à $M^3 \propto R^6$. Enfin, le nombre de tranches n à traiter est proportionnel à la longueur L du fil. Le coût total du calcul est donc proportionnel à $R^6 L$, soit deux ordres de grandeur inférieur à celui d'une inversion directe ($R^6 L^3$). En outre, si les blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ sont stockés sur disque, le code n'a besoin, dans sa version optimisée, de stocker que 4 matrices complexes de taille $M \times M$ en mémoire.

Dans le cas (pratique) où les électrons sont couplés aux phonons, l'algorithme est sensiblement plus compliqué (il faut en particulier résoudre un système d'équations pour deux fonctions de Green couplées), mais partage les mêmes bases.

III.3. Implémentation sur CPU.

Même simplifié, l'algorithme ci-dessus met bien en évidence les opérations caractéristiques réalisées dans un code de fonctions de Green récursives. Nous avons implémenté cet algorithme dans TB_Sim, d'une façon que nous estimons optimale. Nous profitons des accélérations suivantes :

- Les opérations sur les matrices « denses » (95% du temps de calcul) sont confiées aux bibliothèques BLAS MKL (produits matriciels) et LAPACK MKL (inversion) d'Intel qui sont très bien optimisées. Il va de soi que les différentes opérations sont entrelacées de façon à réutiliser au mieux les termes communs aux relations de récurrence ci-dessus, afin de minimiser le nombre total de produits matriciels.

- En outre, les blocs $t_{i,i+1}$, de l'Hamiltonien H sont eux-mêmes creux. La multiplication d'une matrice $t_{i,i+1}$ par une matrice dense (opération dont la complexité est $\propto M^2$ au lieu de $\propto M^3$) est faite « à la volée » par des fonctions spécifiques qui traitent directement les éléments de matrice non nuls de H . Ces produits matrice dense/matrice creuse peuvent représenter jusqu'à la moitié des produits matriciels effectués, aussi est-il important de les optimiser comme tels.

- Si la matrice Hamiltonien H est réelle symétrique, la fonction de Green $G(z)$ est complexe symétrique. Les blocs $g_{i,i}^{(i)}$ et $G_{i,i}$ ci-dessus sont donc eux aussi complexes symétriques. Il n'y a pas, cependant, de fonction BLAS permettant de calculer efficacement un produit $C = AB$ symétrique. Ces produits sont donc traités par des « extensions » de la bibliothèque BLAS codées par nos soins. A l'heure actuelle, le triangle inférieur d'un produit symétrique est calculé par blocs 32 à 128 lignes/colonnes par les fonctions de multiplication standard de la bibliothèque BLAS MKL, puis transposé si besoin. Nous pouvons ainsi tirer parti des performances de la bibliothèque MKL tout en gagnant un facteur $\sim 1.8 \times$ (au lieu de 2 idéalement) sur un produit $C = AB$ symétrique.

- Certains réseaux atomiques (ou de différences finies) peuvent être décomposés en

deux sous-réseaux A et B tels que les atomes de A soient couplés aux atomes de B, mais pas les atomes de A entre eux ni les atomes de B entre eux. Cette propriété topologique permet de réduire le coût du calcul d'un facteur ~ 2 dans certains systèmes.

L'historique (blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$) de la passe avant est stocké en mémoire si le système est suffisamment petit, sur disque sinon. Le stockage sur disque entraîne une perte de performances d'environ 5% sur CPU. Comme nous le verrons plus loin, les entrées/sorties deviennent néanmoins un véritable goulot d'étranglement sur GPU et nous avons dû mettre en place des solutions spécifiques que nous décrirons au paragraphe III.4. Bien entendu, ces solutions ont également profité au code CPU (qui partage ses fonctions d'entrées/sorties avec le code GPU), et ont permis *in fine* de réduire le coût des entrées/sorties à seulement $< 1\%$ du temps de calcul.

Les codes de fonctions de Green sont parallélisés de la façon suivante :

- A bas niveau, les opérations BLAS et LAPACK confiées à la librairie MKL, ainsi que les opérations relatives aux matrices creuses peuvent être parallélisées OpenMP.
- A haut niveau, le calcul du courant à travers un dispositif nécessite d'intégrer la fonction de Green de $G(z)$ sur un chemin dans le plan complexe comprenant plusieurs centaines de points z (typiquement 128 à 1024). Les points de ce chemin sont distribués *a priori* aux différents processus d'un communicateur MPI. Les performances de cette parallélisation MPI sont presque idéales dès lors que le nombre de processus MPI divise le nombre de points du chemin, ce qui est toujours le cas en pratique (moyennant si nécessaire un léger suréchantillonnage du chemin). Des communications point à point ont lieu avant et pendant le calcul des fonctions de Green (échange de données pouvant coupler différentes valeurs de z), suivies d'une réduction après ce calcul (collecte des résultats pour intégration).
- A plus haut niveau encore le calcul de la « caractéristique courant-tension » d'un dispositif comprend plusieurs « points de polarisation » (tensions appliquées aux différents contacts) qui peuvent eux-aussi être distribués à des communicateurs MPI. Cette parallélisation est triviale (car les points de polarisation sont indépendants) mais permet en principe de réduire considérablement le temps de rendu !

La parallélisation OpenMP est surtout utilisée pour calculer de gros systèmes dont les besoins excèdent la mémoire moyenne par cœur (par ex. 3Go au CCRT/Titane) sans avoir à dépeupler les nœuds. Nous exploitons peu la parallélisation sur les points de polarisation car les temps d'attente en queue pour des requêtes > 1024 cœurs ne le justifient pas forcément. Ce code est donc couramment utilisé sur 128 à 512 cœurs.

III.4. Implémentation sur GPU et performances.

Nous avons entrepris début 2011 de porter nos codes de fonctions de Green sur architectures hybrides afin de profiter des performances des GPUs pour réduire nos temps de rendu et/ou simuler des systèmes plus complexes.

Notre choix s'est porté sur les cartes NVIDIA FERMI (architecture retenue au CCRT et au TGCC), le langage CUDA, et sur les librairies NVIDIA (cuBLAS, ...) qui sont bien plus avancées que celles de ses concurrents. Nous discutons ci-dessous le choix des librairies, notre implémentation sur GPU et le problème posé par les entrées/sorties disque.

III.4.a. Choix des librairies.

Sur CPU, 95% du temps de calcul est consommé par les librairies BLAS et LAPACK (algèbre linéaire). Nous avons donc besoin d'une implémentation très performante de ces librairies sur GPU.

La librairie cuBLAS⁶ (BLAS sur GPU NVIDIA) fournit les produits entre matrices complexes. Nous avons complété cette librairie avec nos propres « noyaux » pour les produits matrices réelles/matrices complexes et pour les produits $C = AB$ symétriques (cf. paragraphe III.3).

La librairie MAGMA⁷ implémente les principales opérations LAPACK. A l'époque où nous avons commencé (février 2011), MAGMA fournissait la décomposition LU préalable à l'inversion d'une matrice, mais pas l'inversion elle-même, que nous avons du coder. Depuis, MAGMA fournit également l'inversion, mais avec des performances ~30% inférieures à notre implémentation. Les résultats donnés ci-dessous ont été calculés avec notre propre code pour l'inversion.

III.4.b. Implémentation(s).

Une fois les librairies cuBLAS et MAGMA complétées pour nos besoins, nous avons développé des interfaces de haut niveau (« wrappers ») pour toutes les opérations BLAS et LAPACK. Ces wrappers (qui ont le même prototype que les fonctions originales) décident, en fonction de la taille du problème, s'il est opportun d'effectuer l'opération sur GPU, et, le cas échéant, transfèrent les données depuis la mémoire centrale vers le GPU et vice-versa. L'utilisation des GPUs est donc totalement transparente pour l'utilisateur, qui doit simplement remplacer dans son code tous les appels BLAS et LAPACK (ZGEMM, ZGETRI, etc...) par les wrappers correspondants (myZGEMM, myZGETRI, etc...). Toutefois, une telle implémentation ne saurait être optimale (comme nous allons le démontrer), car il faut payer le coût des transferts entre mémoire centrale et GPU à chaque appel, aucune donnée n'étant laissée sur le GPU pour enchaîner les opérations. Nous appellerons ainsi cette implémentation « code GPU/données CPU » ou plus simplement « code GPU/CPU ».

Nous comparons les performances du code GPU/CPU et celle du code CPU dans les tables 1 et 2. La machine utilisée est un Intel Xeon quadricœurs X5550 cadencé à 2.67 GHz, avec un GPU NVIDIA Tesla C2070 (technologie Fermi), et les librairies MKL 11.1, CUDA 4.1 et MAGMA 1.1. Le système test est un fil de silicium carré de côté $a = 4$ nm et de longueur $L = 12.3$ nm (64 tranches). La taille de chaque bloc $H_{i,i}$ de l'Hamiltonien de liaisons fortes du fil est $M = 1604$. Le fil reste suffisamment petit pour que l'historique des blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ puisse être stocké en mémoire (8 Go RAM). Nous reviendrons plus tard sur le problème du stockage sur disque.

Sont reportés dans la table 1 les temps de calcul des « self-énergies », des passes avant et arrière (pour les $G_{i,i}$ et les $G_{i,1}/G_{i,n}$ séparément), et le temps « total » (somme des précédents) pour une valeur de z donnée. Ces self-énergies sont des corrections aux blocs $H_{1,1}$ et $H_{n,n}$ de l'Hamiltonien qui décrivent le couplage du fil aux contacts extérieurs. Elles se calculent également avec un algorithme récursif assez similaire dans sa structure à celui du paragraphe III.2. Le coût du calcul des self-énergies est proportionnel à a^6 (comme le coût $\propto a^6 L$ du calcul de la fonction de Green), mais est indépendant de L .

⁶ <http://developer.nvidia.com/cublas>

⁷ <http://icl.cs.utk.edu/magma/index.html>

La part des principales opérations des passes avant et arrière est reporté dans la table 2 : Inversion des matrices, produits entre matrices denses, produits matrice dense/matrice creuse, et lecture/écriture des blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ (ici en mémoire).

Etant donné que le nombre de cœurs des machines hybrides est en général très supérieur au nombre de GPUs, il est important de comparer les performances des GPUs avec les performances d'un code CPU OpenMP (multithreads). C'est pourquoi les tables 1 et 2 donnent les performances du code CPU séquentiel (1 thread), du code CPU OpenMP (4 threads), et du code GPU/CPU.

$a = 4 \text{ nm}$ $L = 12.3 \text{ nm}$ (64 tranches)	CPU 1 thread	CPU 4 threads		GPU Données CPU		GPU Données GPU	
Self-énergies	1062,6	283,1	3,75 x	52,6	20,22 x	40,1	26,48 x
Passe avant	483,6	138,1	3,50 x	83,3	5,80 x	28,5	16,94 x
Passe arrière ($G_{i,i}$)	434,9	115,9	3,75 x	47,3	9,20 x	14,4	30,11 x
Passe arrière ($G_{i,1}$ & $G_{i,n}$)	810,1	220,5	3,67 x	49,3	16,42 x	27,9	29,05 x
Total	2791,2	757,6	3,68 x	232,5	12,01 x	111,0	25,14 x

Table 1 : Temps de calcul (secondes) des « self-énergies », de la passe avant et de la passe arrière ($G_{i,i}$, $G_{i,1}$ et $G_{i,n}$) sur CPU (1 & 4 threads) et sur GPU (avec stockage des données en mémoire CPU ou en mémoire GPU). Les accélérations sont mesurées par rapport au temps CPU 1 thread.

$a = 4 \text{ nm}$ $L = 12.3 \text{ nm}$ (64 tranches)	CPU 1 thread		CPU 4 threads		GPU Données CPU		GPU Données GPU	
Inversions	222,9	12,9 %	63,4	13,4 %	18,2	10,1 %	16,9	23,8 %
Multiplications denses	1402,5	81,1 %	363,4	76,6 %	56,8	31,6 %	41,3	58,2 %
Multiplications creuses	83,8	4,8 %	27,5	5,8 %	85,4	47,5 %	2,4	3,4 %
Blocs I/O	4,0	0,2 %	4,6	1,0 %	4,1	2,3 %	4,0	5,6 %
Sous-total	1713,2	99,1 %	458,9	96,8 %	164,4	91,5 %	64,6	91,1 %

Table 2 : Temps de calcul (secondes) et part des inversions, des multiplications entre matrices denses, des multiplications matrice dense/matrice creuse, et de la lecture/écriture des blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ dans les passes avant et arrière. La dernière ligne représente le temps et la part totale de ces quatre opérations dans le calcul. Les blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ sont stockés en RAM.

Le code CPU 4 threads est en moyenne $3,7\times$ plus rapide que le code CPU séquentiel. Le gain est assez homogène dans toutes les parties du code (self-énergies, passes avant et arrière) et sur toutes les opérations (inversions, produits denses et creux). Cela démontre la très bonne qualité de la parallélisation de la librairie MKL et des opérations sur les matrices creuses (codées par nos soins). Les inversions représentent environ 13% du temps de calcul sur CPU, les multiplications entre matrices denses presque 80%, et les multiplications matrice dense/matrice creuses 5%.

Le code GPU/CPU est en moyenne $12\times$ plus rapide que le code CPU séquentiel. Il est donc compétitif même vis à vis du code CPU 4 threads. Toutefois, ce gain reste loin des performances optimales que l'on peut attendre d'un GPU ($\sim 33\times$ sur un produit de matrices par exemple⁸). Il est

⁸ A titre de référence, sur la machine de test le GPU délivre 338.4 Gflops/s sur un produit de deux matrices complexes

surtout très inhomogène (seulement $5,8\times$ sur la passe avant mais jusqu'à $16,4\times$ sur la passe arrière). La table 2 démontre clairement que le temps de calcul est en fait dominé par la seule opération qui n'ait pas été implémentée sur GPU : les produits matrice dense/matrice creuse, qui représentent maintenant 47,5% de l'effort (au lieu de $\sim 5\%$ sur CPU). En outre, les transferts entre la mémoire centrale et le GPU consomment une fraction significative ($\sim 10\%$) du temps de calcul.

Paralléliser les multiplications matrice dense/matrice creuse sur 4 threads permettrait de porter le gain du code GPU/CPU à $15\text{--}16\times$. Nous avons toutefois entrepris, afin de tirer le meilleur parti des GPUs, d'écrire un code spécifique pour ces accélérateurs, en stockant les données sur le GPU afin de minimiser les transferts avec la mémoire centrale (code « GPU/GPU »). Nous avons dû coder à cet effet les noyaux manquants pour les opérations sur les matrices creuses⁹, ainsi qu'un ensemble de noyaux pour des opérations intermédiaires « simples », mais qui nous auraient obligé à rentrer/sortir des données du GPU si nous avions voulu continuer à les exécuter sur CPU. Au total, c'est une librairie d'environ 15 000 lignes de code en C et CUDA qui aura été développée pour porter complètement le code sur GPU.

Les performances de ce code GPU/GPU sont également données dans les tables 1 et 2. Les gains sont très significatifs (en moyenne $25\times$ sur le code CPU séquentiel et plus de $2\times$ sur le code GPU/CPU). Ces gains sont proches des valeurs optimales⁸ et assez homogènes dans toutes les parties du code (self-énergies, passes avant et arrière). L'inversion est l'opération la plus limitante sur GPU (ce qui explique la relative sous-performance de la passe avant). La part des opérations matrice dense/matrice creuse est retombée sous 4%. L'Hamiltonien est transféré une bonne fois pour toutes sur le GPU avant le début du calcul. Les seules données rentrées/sorties du GPU en cours de calcul sont les blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ qui doivent être stockés en mémoire. Noter que l'allocation et la copie de ces blocs représente quand même presque 6% du temps de calcul !

III.4.c. Le problème des entrées/sorties disque.

Dans ces conditions, le coût du stockage des blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ sur disque prend vite des proportions dramatiques... A titre d'exemple, nous donnons dans la table 3 la part prise par la lecture/écriture (synchrone) de ces blocs sur disque pour un fil 4 fois plus long (256 tranches) nécessitant le stockage de 18,8 Go de données. Les valeurs de référence pour un stockage en RAM (impossible !) ont été extrapolées à partir des données de la table 2. Les opérations de lecture/écriture sur disque représentent 50% du temps de calcul et dégradent les performances globales de 88%.

$a = 4 \text{ nm}$ $L = 49.2 \text{ nm}$ (256 tranches)	GPU/GPU RAM (extrapolé)		GPU/GPU Sync. I/O		GPU/GPU Async. I/O+4 Go RAM	
Blocs I/O	16,0	5,6 %	267,1	49,9 %	67,1	19,9 %
Total	285,2		535,9	-87,9 %	336,7	-18,1 %

Table 3 : Temps de calcul total (secondes) ; Temps et part de de la lecture/écriture des blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ dans les passes avant et arrière. Les blocs $g_{i,i}^{(i)}$ et $g_{i,1}^{(i)}$ sont stockés soit en RAM, soit sur disque avec I/O synchrones ou I/O asynchrones (voir texte principal). La dégradation des performances (en %) par rapport au stockage en RAM est également indiquée sur la ligne « Total ».

⁸ 1604×1604 , soit $33\times$ plus que le CPU (10.2 Gflops/s).

⁹ La librairie « cuSPARSE » de NVIDIA ne répond que très partiellement aux besoins du code.

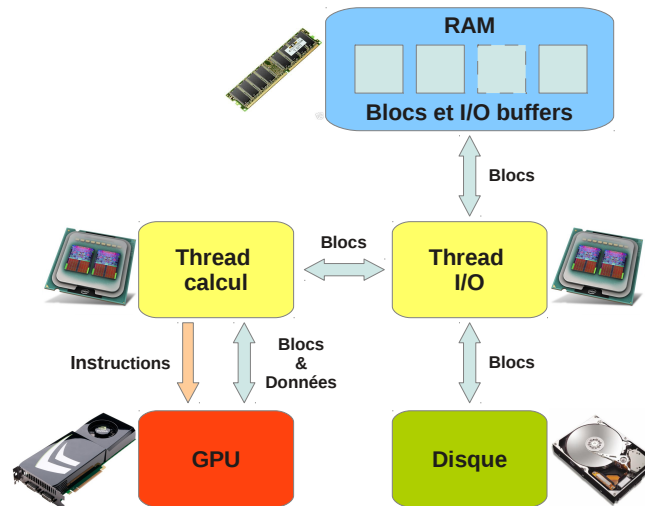


Fig. 8 : Organisation du code et des échanges de données.

Il nous a donc fallu mettre en place des solutions adaptées pour éliminer ce goulot d'étranglement. Nous sommes ainsi passés d'un modèle « synchrone » à un modèle « asynchrone » pour les entrées/sorties disque, confiées à un thread séparé. En écriture (passe avant), nous rendons la main au code fonctions de Green dès réception d'un bloc à stocker, qui est transféré sur disque en parallèle avec les opérations de calcul. Nous acceptons jusqu'à quatre requêtes¹⁰ dans la file d'attente, au-delà desquelles les opérations d'écriture redeviennent « bloquantes » (synchrone). Lors de la passe arrière, nous avançons la lecture des quatre premiers blocs autant que possible. Ensuite, dès qu'un bloc est réclamé par le code fonctions de Green, un autre est relu sur le disque (en parallèle toujours avec les opérations de calcul), afin que les données aient été rapatriées en mémoire lorsque le code en aura besoin. L'organisation du code et des échanges de données est représenté schématiquement sur la figure 8.

Le gestion asynchrone des entrées/sorties permet de réduire leur coût pourvu que la bande passante vers le disque ne soit pas saturée. Dans le cas contraire, les opérations d'écriture vont s'accumuler jusqu'à bloquer la file d'attente dans la passe avant, et les opérations de lecture ne seront jamais terminées à temps dans la passe arrière (dans les deux cas, les entrées/sorties redeviennent *in fine* synchrone). Pour limiter la pression mise sur le disque, nous avons permis d'entrelacer stockage en mémoire et stockage sur disque. Nous spécifions quelle quantité de RAM nous sommes prêts à consacrer au stockage des blocs, et le code répartit ceux-ci en mémoire et sur disque de façon à limiter au mieux la charge de ce dernier.

Dans le cas présent, en allouant 4 Go de RAM (sur 18,8 Go) au stockage des blocs, le coût des entrées/sorties est divisé par 4, et les performances ne sont plus dégradées que de 20% (au lieu de 88% en mode synchrone). La situation s'améliore en fait pour les fils plus gros, la quantité de données à stocker pour chaque tranche étant proportionnel à M^2 (a^4), alors que le coût des opérations de calcul est proportionnel à M^3 (a^6). Ce modèle d'entrées/sorties asynchrones entrelacées avec un stockage en RAM permet donc de couvrir les besoins du code quelle que soit la dimension de l'objet étudié avec seulement 3 à 4 Go de mémoire (centrale) par GPU, quantité disponible sur presque toutes les machines hybrides (et en particulier celles du CCRT et du TGCC).

¹⁰ Ce choix est dicté par la structure (fréquence et distribution) des entrées/sorties disque du code.

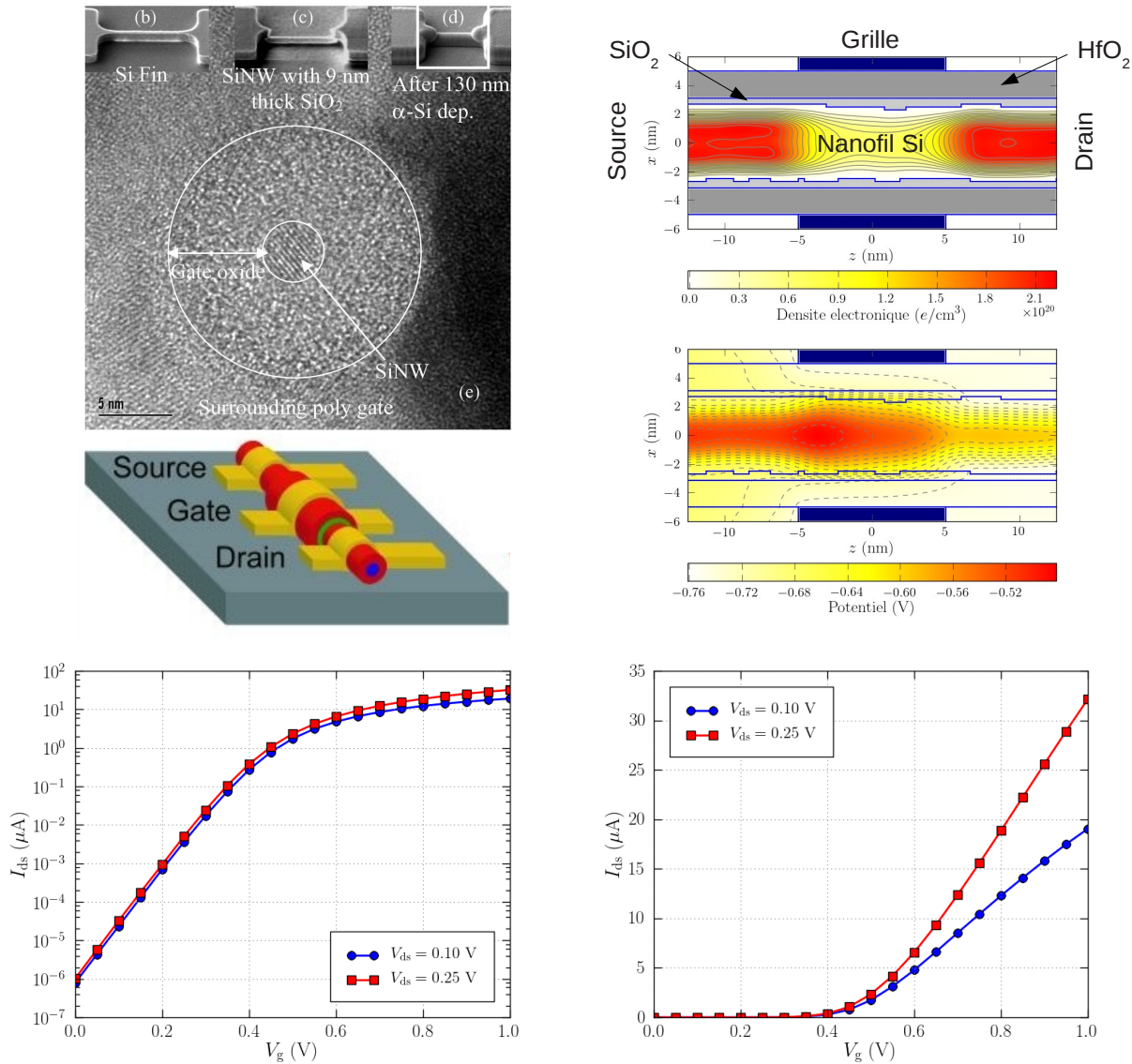


Fig. 9 : En haut à gauche, coupes d'un transistor à nanofil (Si NW) de silicium de 4 nm de diamètre. Le nanofil est encapsulé dans un oxide (« Gate oxide », SiO_2) qui l'isole de l'électrode de « grille » qui l'entoure (« Surrounding poly gate »). La tension V_g appliquée à cette électrode de grille permet de contrôler le courant qui circule dans le nanofil entre les électrodes de « source » et de « drain ». [N. Singh et al., IEEE Electron Device Letters **27**, 283 (2006)]. **En haut à droite**, densité d'électrons et potentiel calculé avec notre code de fonctions de Green GPU dans un transistor de géométrie comparable (diamètre du fil : 5 nm ; tension de grille $V_g = 0.75$ V et tension source-drain $V_{ds} = 0.1$ V). **En bas**, courant source-drain I_{ds} vs tension de grille V_g calculé dans ce transistor, pour deux tensions source-drain $V_{ds} = 0.1$ V et $V_{ds} = 0.25$ V (échelle logarithmique à gauche et linéaire à droite). Plus la tension de grille est élevée, plus le courant qui traverse le nanofil est important. Il est ainsi possible de faire fonctionner le transistor comme un interrupteur, en le faisant basculer d'un mode « bloqué » ($V_g \sim 0$ V) à un mode « passant » ($V_g > 0.5$ V). Calculs effectués sur TGCC/Curie sur 256 GPUs ($T_{wall} = 3h06mn$).

III.5. Applications et perspectives.

Voici à titre d'exemple la caractéristique « courant-tension de grille » d'un transistor basé sur un nanofil de silicium de 5 nm de diamètre (Fig. 9), calculée avec le code fonctions de Green GPU

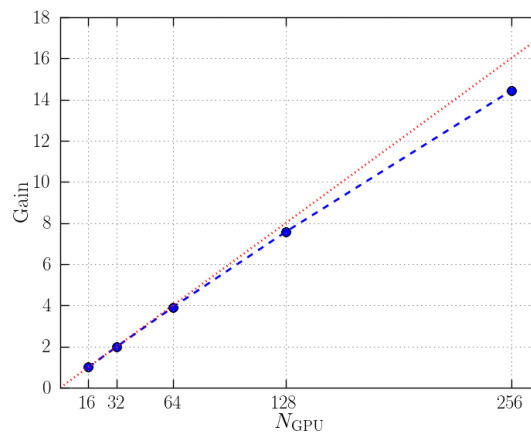


Fig. 10 : Performances du code de fonctions de Green en fonction du nombre N_{GPU} de GPUs. Les points bleus représentent le gain sur le temps de rendu (par rapport à la référence $N_{GPU} = 16$). La ligne pointillée rouge représente la loi d'échelle idéale. Mesures effectuées sur TGCC/Curie.¹²

présenté ci-dessus. La « grille » permet de moduler le potentiel dans le fil et de contrôler le courant qui circule entre les contacts de « source » et de « drain ». Plus la tension de grille est importante, plus le courant dans le fil est élevé. Il est ainsi possible de faire fonctionner le transistor comme un interrupteur, et de le faire basculer entre un état « bloqué » et un état « passant ». Ce type de dispositif est caractéristique des composants les plus avancés réalisés en laboratoire aujourd'hui. Les propriétés électroniques du nanofil ont été calculées avec un modèle de masse effective, en tenant compte du couplage électrons-phonons (version étendue de l'algorithme présenté au paragraphe III.2). La surface du nanofil est légèrement rugueuse, comme elle l'est en pratique.

Les calculs ont été réalisés sur la partition hybride de la machine Curie du TGCC (256 GPUs). Les GPUs calculent les fonctions de Green, tandis que les CPUs alimentent les GPUs, s'occupent des entrées/sorties disque (cf. paragraphe III.4.c), et résolvent un certain nombre de problèmes moins gourmands, tel que l'équation de Poisson (calcul du potentiel électrique dans le dispositif, parallélisé OpenMP sur les 4 cœurs associés à un GPU). Chaque GPU évalue les fonctions de Green pour une ou plusieurs valeurs¹¹ de z , et échange (via la librairie MPI) avec les autres GPUs les données nécessaires à l'avancement du calcul (par exemple, l'effet des phonons dépend de certains éléments de la fonction de Green calculée pour d'autres valeurs de z , donc par d'autres GPUs). Ces échanges sont initiés avec des requêtes non-bloquantes (asynchrones) lancées bien avant que les données ne soient effectivement réclamées par les GPUs afin (comme pour les entrées/sorties disque) de ne pas bloquer le calcul avec des communications.

Les performances du code en fonction du nombre de GPUs sont reportées sur la figure 10. Les mesures ont été effectuées sur TGCC/Curie. Le gain est presque linéaire de 16 à 256 GPUs, ce qui démontre la bonne parallélisation du code. La légère dégradation des performances à grand nombre de GPUs est due à la part croissante des opérations CPU (résolution de l'équation de Poisson parallélisée OpenMP) et à la part croissante des communications MPI.

Ce code va être utilisé dans les mois qui viennent pour modéliser les propriétés de dispositifs

¹¹ Le chemin d'intégration comprend 256 valeurs de z .

¹² Les performances ont été mesurées sur un sous ensemble des points de la courbe courant-tension de la figure 9.

nanofils & FinFETs fabriqués et caractérisés au CEA/LETI pour le nœud 16 nm. Notre objectif est d'affiner la compréhension de ces dispositifs, dans des gammes de dimensions où les outils de « TCAD¹³ » classiques deviennent insuffisants. Nous nous intéresserons en particulier aux interactions entre les différents mécanismes de diffusion qui limitent le courant dans ces dispositifs (phonons, impuretés, rugosité de surface, charges piégées dans les oxydes...). Nous espérons ainsi pouvoir répondre à un certain nombre de questions fondamentales non encore élucidées (sur la variabilité des dispositifs, les effets du « remote Coulomb scattering » dans les oxydes de grille, etc...). Nous nous attacherons aussi à évaluer l'effet d'un certain nombre de « boosters » technologiques utilisables pour augmenter les performances électriques de ces transistors, tels que les contraintes, l'introduction de Germanium ou même de matériaux III-V (InAs, ...). Seul un code comme TB_Sim, capable de modéliser ces dispositifs à l'échelle du matériau, peut répondre de façon quantitative à ces problèmes, guider le choix des options technologiques et alimenter les « modèles compacts » utilisés pour la conception de circuits à grande échelle. Nous espérons que cette collaboration étroite entre équipes de simulation et de caractérisation permettra d'optimiser rapidement ces dispositifs. Ces travaux seront menés dans le cadre d'un projet ANR (« Quasanova »), et n'ont été rendus possibles qu'avec le soutien du GENCI (projet gen6096) et de PRACE (projet 2010PA0885).

IV. Conclusions.

Nous avons présenté le code TB_Sim destiné à la modélisation des problèmes complexes en nanosciences et nanotechnologies. Nous avons montré sa capacité à répondre à des questions d'intérêt expérimental et à défricher des terrains inexplorés. Ce code fait usage de différents niveaux et techniques de parallélisation (OpenMP, MPI, GPUs) qui lui permettent de profiter pleinement et de façon efficace des machines de calcul haute performance. Un grand nombre des résultats obtenus avec TB_Sim n'auraient d'ailleurs pu l'être sans le concours des moyens de calcul du GENCI. TB_Sim illustre donc parfaitement la nécessité de supercalculateurs et de codes adaptés pour la modélisation avancée en physique.

13 « Technology Computer Aided Design », conception de dispositifs assistée par ordinateur.

Références :

- [1] : *Accumulation capacitance of narrow band gap metal-oxide-semiconductor capacitors.*
E. Lind, Y. M. Niquet, H. Mera et L.-E. Wernersson,
Appl. Phys. Lett. **96**, 233507 (2010).
- [2] : *Stark effect in GaN/AlN nanowire heterostructures: Influence of strain relaxation and surface states.*
D. Camacho et Y. M. Niquet,
Phys. Rev. B **81**, 195313 (2010).
- [3] : *The structural properties of GaN/AlN core-shell nanocolumn heterostructures.*
K. Hestroffer, R. Mata, D. Camacho, C. Lecrere, G. Tourbot, Y. M. Niquet, A. Cros, C. Bougerol, H. Renevier et B. Daudin,
Nanotechnology **21**, 415702 (2010).
- [4] : *Elastic strain relaxation in GaN/AlN nanowire superlattice.*
O. Landré, D. Camacho, C. Bougerol, Y. M. Niquet, V. Favre-Nicolin, G. Renaud, H. Renevier et B. Daudin,
Phys. Rev. B **81**, 153306 (2010).
- [5] : *Analysis of strain and stacking faults in single nanowires using Bragg coherent diffraction imaging.*
V. Favre-Nicolin, F. Mastropietro, J. Eymery, D. Camacho, Y. M. Niquet, B. M. Borg, M. E. Messing, L. E. Wernersson, R. E. Algra, E. P. A. M. Bakkers, T. H. Metzger, R. Harder et I. K. Robinson,
New Journal of Physics **12**, 035013 (2010).
- [6] : *Structural properties of GaN insertions in GaN/AlN nanocolumn heterostructures.*
C. Bougerol, R. Songmuang, D. Camacho, Y. M. Niquet, R. Mata, A. Cros et B. Daudin,
Nanotechnology **20**, 295706 (2009).
- [7] : *Scanning tunnelling spectroscopy of cleaved InAs/GaAs quantum dots at low temperatures.*
A. Urbieto, B. Grandidier, J. P. Nys, D. Deresmes, D. Stiévenard, A. Lemaître, G. Patriarche et Y. M. Niquet,
Phys. Rev. B **77**, 155313 (2008).
- [8] : *Quantum dots and tunnel barriers in InAs/InP nanowire heterostructures : Electronic and optical properties.*
Y. M. Niquet et D. Camacho Mojica,
Phys. Rev. B **77**, 115316 (2008).
- [9] : *Strain and shape of epitaxial InAs/InP nanowires measured by grazing incidence X-ray techniques.*
J. Eymery, F. Rieutord, V. Favre-Nicolin, O. Robach, Y. M. Niquet, L. Fröberg, T. Mårtensson et L. Samuelson,
Nano Letters **7**, 2596 (2007).
- [10] : *Effects of a shell on the electronic properties of nanowire superlattices.*
Y. M. Niquet,
Nano Letters **7**, 1105 (2007).
- [11] : *Electronic and optical properties of InAs/GaAs nanowire superlattices.*
Y. M. Niquet,
Phys. Rev. B **74**, 155304 (2006).
- [12] : *Effects of strains on the mobility in silicon nanowires.*
Y. M. Niquet, C. Delerue et C. Krzeminski,
Soumis à Nano Letters.
- [13] : *Fully atomistic simulations of phonon-limited mobility of electrons and holes in <001>, <110> and <111>-oriented Si nanowires.*
Y. M. Niquet, C. Delerue, D. Rideau et B. Videau,
IEEE Transactions on Electron Devices **59**, 1480 (2012).
- [14] : *Impurity-limited mobility and variability in gate-all-around silicon nanowires.*
Y. M. Niquet, H. Mera et C. Delerue,
Appl. Phys. Lett. **100**, 153119 (2012).
- [15] : *Atomistic modeling of electron-phonon coupling and transport properties in n-type [110] silicon nanowires.*
W. Zhang, C. Delerue, Y. M. Niquet, G. Allan et E. Wang,
Phys. Rev. B **82**, 115319 (2010).
- [16] : *Charged impurity scattering and mobility in gated silicon nanowires.*
M. P. Persson, H. Mera, Y.-M. Niquet, C. Delerue et M. Diarra,
Phys. Rev. B **82**, 115318 (2010).
- [17] : *Band structure effects on the scaling properties of [111] InAs nanowire MOSFETs.*
E. Lind, M. Persson, Y. M. Niquet et L. E. Wernersson,
IEEE Trans. Electron Devices **56**, 201 (2009).
- [18] : *Orientational dependence of charge transport in disordered silicon nanowires.*

- M. P. Persson, A. Lherbier, Y. M. Niquet, F. Triozon et S. Roche,
Nano Letters **8**, 4146 (2008).
- [19] : *Quantum transport length scales in silicon-based semiconducting nanowires : Surface roughness effects.*
A. Lherbier, M. P. Persson, Y. M. Niquet, F. Triozon et S. Roche,
Phys. Rev. B. **77**, 085301 (2008).
- [20] : *Two-dimensional graphene with structural defects: Elastic mean free path, minimum conductivity, and Anderson transition.*
A. Lherbier, S. M. M. Dubois, X. Declerck, S. Roche, Y. M. Niquet et J. C. Charlier,
Phys. Rev. Lett. **106**, 046803 (2011).
- [21] : *Charge transport in chemically doped 2D graphene.*
A. Lherbier, X. Blase, Y. M. Niquet, F. Triozon et S. Roche,
Phys. Rev. Lett. **101**, 036808 (2008).
- [22] : *Transport length scales in disordered graphene-based materials : Strong localization regimes and dimensionality effects.*
A. Lherbier, B. Biel, Y. M. Niquet et S. Roche,
Phys. Rev. Lett. **100**, 036803 (2008).
- [23] : *Chemical functionalization effects on armchair graphene nanoribbon transport.*
A. Lopez-Bezanilla, F. Triozon et S. Roche,
Nano Letters **9**, 2537 (2009).
- [24] : *Chemically induced mobility gaps in graphene nanoribbons: a route for upscaling device performances.*
B. Biel, F. Triozon, X. Blase et S. Roche,
Nano Letters **9**, 2725 (2009).
- [25] : *Ionization energy of donor and acceptor impurities in semiconductor nanowires : Importance of dielectric confinement.*
M. Diarra, Y. M. Niquet, C. Delerue et G. Allan,
Phys. Rev. B **75**, 045301 (2007).
- [26] : *Screening and polaronic effects induced by a metallic gate and a surrounding oxide on donor and acceptor impurities in silicon nanowires.*
M. Diarra, C. Delerue, Y. M. Niquet et G. Allan,
J. Appl. Phys. **103**, 073703 (2008).
- [27] : *Ab initio calculation of the binding energy of impurities in semiconductors: Application to Si nanowires.*
Y. M. Niquet, L. Genovese, C. Delerue et T. Deutsch,
Phys. Rev. B **81**, 161301(R) (2010).
- [28] : *Band offsets, wells, and barriers at nanoscale semiconductor heterojunctions.*
Y. M. Niquet et C. Delerue,
Phys. Rev. B **84**, 075478 (2011).
- [29] : *On-site matrix elements of the tight-binding hamiltonian of a strained crystal: Application to silicon, germanium and their alloys.*
Y. M. Niquet, D. Rideau, C. Tavernier, H. Jaouen et X. Blase,
Phys. Rev. B **79**, 245201 (2009).