

Dynamic Load Balancing of Corbino-TBG Electronic Transport Simulations on Heterogeneous Clusters

Arthur M. Passos¹, Calebe P. Bianchini^{1,2}

¹Programa de Pós-Graduação em Computação Aplicada – PPGCA
Faculdade de Computação e Informática – FCI
Universidade Presbiteriana Mackenzie – São Paulo, SP – Brasil

²CESAR – Centro de Estudos e Sistemas Avançados do Recife
Recife, PE – Brasil

arthurmoreirap@gmail.com, calebe.bianchini@mackenzie.br

Abstract. *Electronic transport simulations of twisted bilayer graphene (TBG) in Corbino geometry solve large sparse linear systems at every energy point. This work assigns independent energy points on demand to CPU and GPU workers with distinct sparse solvers, without predefined performance weights. For a 1.5° instance, eight CPU nodes reach speedup 6.62 over one node; two CPU nodes with the two GK210 devices of a Tesla K80 reach 3.17 over the C/PARDISO reference, with 99.85% load-balancing efficiency and a makespan 13.2% below the estimated equal-static partition. The branches differ by at most 1.0×10^{-10} in relative conductance.*

Resumo. *Simulações de transporte eletrônico em grafeno bicamada torcido (TBG) na geometria Corbino resolvem grandes sistemas lineares esparsos a cada ponto de energia. Este trabalho atribui pontos de energia independentes sob demanda a trabalhadores CPU e GPU com solvers esparsos distintos, sem pesos de desempenho predefinidos. Para uma instância de $1,5^\circ$, oito nós CPU atingem speedup 6,62 sobre um nó; dois nós CPU com os dois GK210 de uma Tesla K80 atingem 3,17 sobre a referência C/PARDISO, com 99,85% de eficiência de balanceamento e makespan 13,2% menor que a partição estática estimada. Os ramos diferem em no máximo $1,0 \times 10^{-10}$ na condutância relativa.*

1. Introduction

Twisted bilayer graphene (TBG) is formed by stacking two graphene sheets with a relative rotation angle, producing a Moiré superlattice whose size grows as the angle decreases. In the Corbino configuration, the sample is annular, with inner and outer contacts, avoiding the open lateral edges of ribbon-shaped devices [Novoselov et al. 2012, Bahamon et al. 2013, Löthman et al. 2022].

The simulation uses a tight-binding Hamiltonian and the non-equilibrium Green's function (NEGF) formalism. For each energy value, conductance is obtained from selected columns of a retarded Green's function, yielding a sparse linear system with multiple right-hand sides. Since all right-hand sides at one energy share the same coefficient matrix, one numerical factorization is followed by a batched solution. Energy points are independent and therefore expose coarse-grained parallelism in addition to the parallelism inside the sparse solver.

The experiments use a C transcription of the original Fortran/PARDISO simulator, previously developed and validated [Passos and Bianchini 2025]. Across six complete executions on the same CPU node, elapsed time differed by only -0.06% , and the conductance curves of the two implementations agreed end to end to a maximum relative difference of 4.1×10^{-11} . The validated C implementation is therefore adopted as the performance reference; the transcription itself is not a contribution of this work.

On a heterogeneous cluster, workers based on different processors and solver stacks may complete energy points at different rates. Equal static partitioning can then leave faster workers idle while the slowest worker determines the makespan; excluding slower resources, however, wastes available capacity.

This work proposes and evaluates a heterogeneous execution architecture in which distinct sparse-solver backends are integrated behind a common distributed model and energy points are assigned on demand according to observed completion rates, without predefined hardware weights. The study investigates homogeneous scaling, adaptation to workers with different throughputs, and numerical consistency between solver branches. The contributions are the heterogeneous solver organization, the centralized demand-driven controller–worker runtime, and an end-to-end evaluation of ten resource configurations.

2. Physical and Computational Formulation

The application computes electronic conductance over independent energies for a TBG device in Corbino geometry. The tight-binding Hamiltonian H represents the rotated graphene bilayer; decreasing the twist angle increases the Moiré scale and therefore the matrix dimension [Novoselov et al. 2012, Löthman et al. 2022]. The annular Corbino region is coupled to inner and outer contacts and avoids open lateral borders [Bahamon et al. 2013].

Electronic transport is described with the NEGF formalism [Datta 1997, Datta 2005, Ferry and Goodnick 1999]. For energy E , the retarded Green’s function is

$$G^R(E) = [(E + i\eta)I - H - \Sigma_L^R(E) - \Sigma_R^R(E)]^{-1}, \quad (1)$$

where $\Sigma_L^R(E)$ and $\Sigma_R^R(E)$ are the contact self-energies. Defining

$$A(E) = (E + i\eta)I - H - \Sigma_L^R(E) - \Sigma_R^R(E), \quad (2)$$

gives $G^R(E) = A(E)^{-1}$. Conductance follows the Landauer–Caroli formulation, but computing it does not require the complete inverse. Only the Green-function columns associated with the injection modes of the inner contact are needed. Gathering their right-hand sides in B_L yields

$$A(E)X_L(E) = B_L, \quad X_L(E) = G^R(E)B_L, \quad (3)$$

a multiple-right-hand-side (MRHS) system in which one factorization of $A(E)$ is reused for the complete block of right-hand sides.

The tight-binding Hamiltonian is real symmetric and the contact self-energies are complex symmetric, so $A(E) = A(E)^T$. The simulation stores the matrix in upper-triangular CSR form and uses a symmetric LDL^T factorization rather than a general LU factorization [Scott and Tuma 2023]. The Hamiltonian is built once; for each energy, the code assembles and factorizes $A(E)$, solves the MRHS block, and obtains the conductance. This exposes parallelism both within the sparse solver and across independent energy points.

3. Heterogeneous Solution Architecture and Related Work

This section presents the proposed architecture and positions it with respect to existing electronic-transport tools.

3.1. Solution Architecture

The proposed solution is organized around the independence of the energy points. After the common problem data have been made available to the processing resources, each energy point can be treated as an autonomous work unit: a worker receives one point, performs the complete local sequence required to obtain its conductance, returns the result, and becomes available for another point. The architecture therefore separates the global distribution of energy points from the numerical method used inside each worker.

This separation allows different processing resources to coexist in the same execution. A worker is not defined by a particular processor or software stack; it is an execution entity capable of solving one energy point according to the same mathematical formulation. As illustrated in Figure 1a, the common control layer dispatches work to workers that specialize according to the resource on which they execute, while preserving the same sequence of solver phases and the same physical result.

The second element is centralized dynamic load balancing. Instead of dividing the N_E energy points before execution, the control layer maintains the set of unassigned points and provides new work whenever a worker becomes available. Consequently, the number of points processed by each worker is an outcome of its observed throughput during the run rather than an input parameter. Faster workers naturally complete more assignments and receive a larger fraction of the sweep; slower workers still contribute useful capacity without determining the final time through an equal static partition. Figure 1b summarizes this controller–worker organization.

3.2. Related Work

Electronic-transport tools differ in physical model, accelerator support, distributed execution, and the unit and time at which work is partitioned. Kwant provides a general tight-binding toolkit and leaves energy-sweep distribution to user-level orchestration [Groth et al. 2014]. NanoTCAD ViDES can distribute energy points across a cluster but does not provide the GPU path considered here [NanoTCAD ViDES 2023b, NanoTCAD ViDES 2023a]. QuantumATK supports multinode and multi-GPU execution, while its work partition is defined before execution [Synopsys 2024, Synopsys 2025a, Synopsys 2025c, Synopsys 2025b]. TranSIESTA distributes contour energies, but the GPU acceleration described for the surrounding SIESTA workflow does not accelerate the evaluated NEGF stage [Papior et al. 2017, The SIESTA project 2024].

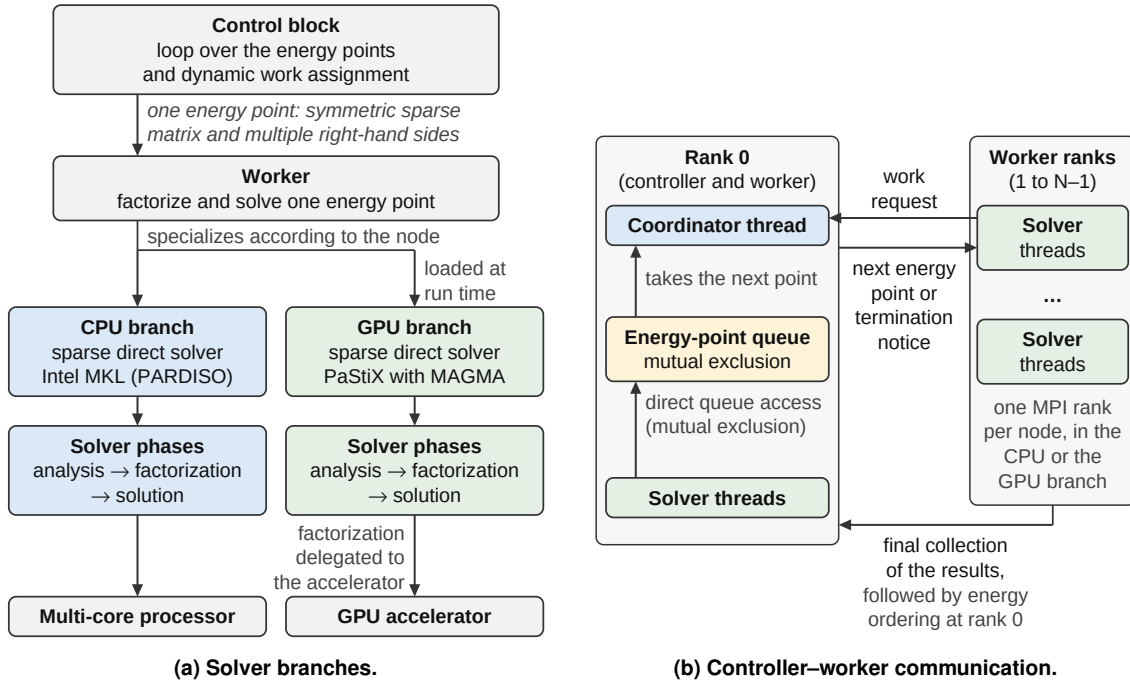


Figure 1. Heterogeneous solver and runtime organization.

OMEN employs a hierarchical static decomposition, whereas ELEQTRONeX statically partitions Green-function columns and supports multi-GPU execution for other device geometries [Ben-Nun et al. 2019, Calderara et al. 2015, Sawant et al. 2025, AMReX-Microelectronics 2024].

Table 1 highlights the distinction relevant to this work. Among the reviewed electronic-transport tools, the workload is either not distributed by the tool or is partitioned before execution. In particular, none of the reviewed solver environments assigns independent energy points on demand to heterogeneous processing units whose effective throughputs may differ during the same execution. The present proposal addresses this gap by combining the Corbino-TBG NEGF workload, which produces complex-symmetric sparse MRHS systems, with demand-driven assignment across distinct solver branches. Demand-driven assignment is an established scheduling strategy in scientific applications [Santana 2020, Fernandes 2020, Assis et al. 2019]; here, the contribution is its integration with heterogeneous electronic-transport solver branches.

4. Implementation Details and Bottleneck Characterization

This section describes the solver integration, the distributed runtime, and the profiling that identifies the computational bottleneck.

4.1. Common Code Base and Solver Integration

The implementation starts from the C transcription previously validated against the original Fortran simulator [Passos and Bianchini 2025]. The transcription preserves the mathematical sequence, sparse representation, data dimensions, and double-precision complex arithmetic while enabling integration with the parallel runtime and alternative sparse-solver interfaces. The CPU branch uses Intel one MKL PARDISO, which supports complex-

Table 1. Electronic-transport tool comparison.

Tool	CPU	GPU	Cluster	Work partition	Model
Kwant	Yes	No	No	None	Tight binding
NanoTCAD ViDES	Yes	No	Yes	Static (energy)	Tight binding
QuantumATK	Yes	Multi	Yes	Static (energy)	DFT / tight binding
TranSIESTA	Yes	Yes ^a	Yes	Static (energy)	DFT
OMEN	Yes	Multi	Yes	Static (hierarchical)	Tight binding / DFT
ELEQTRONeX	Yes	Multi	Yes	Static (columns)	Tight binding
This work	Yes	Multi	Yes	Dynamic (energy)	Tight binding (TBG)

^aGPU acceleration is associated with the SIESTA DFT stage, not with the evaluated TranSIESTA NEGF stage.

symmetric sparse systems, separates analysis, factorization, and solution phases, and accepts multiple right-hand sides in one call [Intel Corporation 2023].

The accelerator branch uses PaStiX with MAGMA. PaStiX provides symbolic analysis, numerical factorization, and solution as distinct operations and supports symmetric sparse systems [Hénon et al. 2002, Inria Bordeaux – Sud-Ouest 2024]. Scotch supplies graph ordering [Pellegrini and Roman 1996]. During numerical factorization, PaStiX operates on supernodal dense blocks and can delegate BLAS-3 operations to MAGMA, which provides dense linear algebra for heterogeneous CPU–GPU systems [Tomov et al. 2010, Dongarra et al. 1990]. This stack preserves the MRHS organization required by Equation 3: one numerical factorization of $A(E_i)$ followed by the batched solution of all 285 right-hand sides for that energy.

Two NVIDIA alternatives were examined in this work. Under CUDA 11.0, the available cuSOLVER sparse interfaces did not provide the required complex-symmetric MRHS workflow with reusable analysis, factorization, and solution phases. cuDSS provides a more suitable multistage sparse-direct workflow, but the installed environment did not satisfy its CUDA 12.x requirement [NVIDIA Corporation 2026a, NVIDIA Corporation 2026b]. PaStiX+MAGMA was therefore adopted for the accelerator path. The accelerator solver is packaged as a dynamically loaded module so that the common executable can select the CPU or accelerator branch at run time without changing the physical model.

4.2. Distributed Runtime and Dynamic Assignment

The distributed implementation uses MPI to connect the processing nodes. The Hamiltonian and common input structures are broadcast before the energy sweep. Each CPU node executes one MPI rank, while the two independently addressable GK210 devices of the Tesla K80 are controlled by separate ranks. Rank zero combines calculation with coordination.

The controller–worker mechanism uses Pthreads on rank zero and requests `MPI_THREAD_MULTIPLE`. A dedicated coordinator thread receives work requests and accesses a shared thread-safe queue. After completing an energy point, a remote worker sends a ready message; the coordinator returns the next energy index or a termination message. Rank zero obtains its own work from the same queue through local synchronization. The dispatch granularity is one energy point. At the end, variable-size result arrays are

collected with `MPI_Gatherv` and reordered by energy before output.

On CPU workers, the sparse solve is carried out by PARDISO using the physical cores of the CPU node. On accelerator workers, each rank binds to one GK210 device and uses a single host thread to control data submission and invoke the PaStiX+MAGMA solver; the host CPU cores are not used as compute workers in the GPU scalability or load-balancing experiments. The GPU module was built with GCC 8.3.0 and linked with PaStiX, StarPU 1.4, Scotch, MAGMA, and CUDA 11.0. The main program uses Intel MPI 2021.7 and the ILP64 interface required by the large sparse problem. A separate PARDISO execution on the CPUs of the accelerator node is used only as a control measurement to characterize the difference between the two host CPU platforms.

4.3. Profiling and Bottleneck

Profiling of the validated C/PARDISO implementation identifies the repeated sparse solve as the dominant target for acceleration. Table 2 shows the phase decomposition measured on one 24-core CPU node. Hamiltonian construction takes 2,468.202 s, or 2.8% of the instrumented total, while the energy loop accounts for 97.2%. Inside that loop, factorization and solution take 83,958.362 s, corresponding to 99.49% of the loop. Matrix assembly, CSR addition, and conductance operations together account for less than one percent.

The profile connects the physical formulation to the parallel design. The Hamiltonian is constructed once, whereas almost all repeated work lies in the independent energy-point solves. The measured fractions therefore justify parallelizing the energy sweep and indicate a parallel fraction of $f \approx 0.971$ for strong-scaling analysis. They also show that optimizing auxiliary matrix assembly or result collection cannot produce an impact comparable to accelerating and distributing the sparse factorization and MRHS solution.

Table 2. C/PARDISO phase decomposition.

Phase	Time (s)	Fraction
Hamiltonian construction	2,468.202	2.8% of total
Energy loop	84,390.075	97.2% of total
Assemble $A(E)$	2.841	< 0.01% of loop
CSR matrix addition	316.184	0.37% of loop
PARDISO factorization + solve	83,958.362	99.49% of loop
Conductance extraction	55.509	0.07% of loop
Conductance multiplications	56.286	0.07% of loop
Instrumented total	86,858.277	100%

5. Experimental Methodology

The experiments use one fixed physical instance; the problem size is unchanged across all configurations. The CPU-only series is therefore a strong-scaling experiment, while the heterogeneous series measures how the same workload is processed by different combinations of available resources. Table 3 summarizes the instance and its resulting

linear-algebra problem¹.

Table 3. Simulation instance parameters.

Parameter	Value
Twist angle θ	1.5°
Hamiltonian dimension N	approximately 719,165
Energy points N_E	1,001
Right-hand sides per energy n_L	285
Matrix property	complex symmetric and sparse
Storage/factorization	upper-triangular CSR / LDL^T
Data visibility	Private

The CPU nodes contain two Intel Xeon E5-2650 v4 processors at 2.20 GHz, for 24 physical cores per node. The accelerator node contains two Intel Xeon E5-2630 v4 processors at 2.20 GHz, for 20 physical host cores, and one NVIDIA Tesla K80. The K80 contains two independently addressable GK210 devices, each with 2,496 CUDA cores and 12 GB of GDDR5 memory. The complete board therefore provides 4,992 CUDA cores and 24 GB of device memory; its compute capability is 3.7.

Ten configurations were executed, as listed in Table 4. Each is identified by the label $x\text{C}y\text{G}$, where x is the number of CPU nodes and y the number of GK210 devices, so that the resources of a configuration can be read from its name; REF denotes the C/PARDISO reference. Configurations 1C0G to 8C0G vary the CPU-node count from one to eight. 0C1G and 0C2G use one and two GK210 devices. 1C1G, 3C1G, and 7C1G combine one device with one, three, and seven CPU nodes, and 2C2G combines both devices with two CPU nodes.

Table 4. Experimental configurations.

Label	Processing units	CPU compute cores	GPU devices	MPI ranks
1C0G	Common runtime, 1 CPU node	24	0	1
2C0G	2 CPU nodes	48	0	2
4C0G	4 CPU nodes	96	0	4
8C0G	8 CPU nodes	192	0	8
0C1G	1 GK210 device	0	1	1
0C2G	2 GK210 devices (K80)	0	2	2
1C1G	1 GK210 + 1 CPU node	24	1	2
3C1G	1 GK210 + 3 CPU nodes	72	1	4
7C1G	1 GK210 + 7 CPU nodes	168	1	8
2C2G	2 GK210 + 2 CPU nodes	48	2	4

The C/PARDISO implementation on one CPU node is the complete-application performance reference, with elapsed time $84,347.667 \pm 166.567$ s from six runs [Passos and Bianchini 2025]. Each of the ten configurations was also executed six times, and the complete-application results are reported as mean and sample standard deviation. The observed standard deviations are small: across all configurations, they remain below 0.37% of the corresponding mean elapsed time. Configuration 1C0G uses the

¹The physical input data are proprietary to Universidade Presbiteriana Mackenzie and cannot be shared because of intellectual-property restrictions associated with MackGraphe.

common runtime with one CPU/PARDISO worker and is used as the one-node reference for CPU strong scaling.

Conventional parallel efficiency is not assigned to heterogeneous configurations because a CPU node and a GK210 worker are not equivalent resources. Those cases are instead characterized by end-to-end time, speedup, points processed per worker, measured mean time per energy point, makespan, and finish-time gap. Load-balancing efficiency compares the measured energy-loop makespan with an aggregate-throughput bound obtained from the measured mean per-point throughputs of the worker types. The equal-static makespan is an analytical estimate computed from the same measured per-point times; no peak or theoretical hardware rates are used in this comparison. Full-application measurements are used instead of peak rates [Hoefler and Belli 2015].

6. Results and Analysis

The results cover CPU strong scaling, heterogeneous performance, dynamic load balance, and the consistency of the solver outputs.

6.1. CPU Strong Scaling

Figure 2 presents CPU strong scaling for the same fixed problem using one, two, four, and eight CPU nodes. It compares the measured speedup and parallel efficiency with ideal scaling and the Amdahl prediction for $f \approx 0.971$. Configuration 1C0G is the one-node reference, so the measured curve reflects scaling of the same common runtime. The measured speedups are 1.00, 1.93, 3.67, and 6.62, corresponding to efficiencies of 100.0%, 96.7%, 91.7%, and 82.7%, respectively.

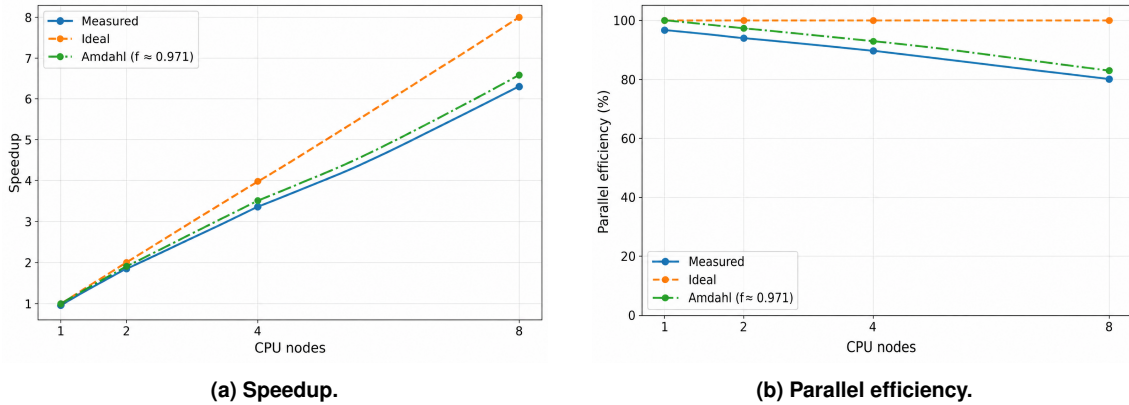


Figure 2. CPU strong scaling.

The measured behavior is consistent with the profile in Table 2. Using $f \approx 0.971$, Amdahl's law predicts speedups of 1.94, 3.69, and 6.67 and parallel efficiencies of 97.2%, 92.1%, and 83.4% for two, four, and eight nodes, respectively. Both predictions closely track the measured values. The reduction in efficiency at higher node counts is therefore consistent with the fixed Hamiltonian-construction cost and the small runtime overheads becoming more visible as the parallel energy loop is shortened.

Table 5 retains the complete-application comparison against the validated C/PARDISO reference. Under that definition, the common one-node runtime has speedup

0.97, while two, four, and eight CPU nodes reach 1.88, 3.56, and 6.43, respectively. The one-node common runtime is 2.98% slower than the validated C/PARDISO reference; this difference is the observed overhead of the common heterogeneous runtime required to provide centralized dynamic load balancing and runtime solver selection.

Table 5. Homogeneous performance.

Label	Configuration	Elapsed time (s)	Speedup
REF	C/PARDISO reference, 1 CPU node	84,347.667 $\pm$ 166.567	1.00
1C0G	Common runtime, 1 CPU node	86,862.833 $\pm$ 230.798	0.97
2C0G	2 CPU nodes	44,906.500 $\pm$ 89.663	1.88
4C0G	4 CPU nodes	23,671.667 $\pm$ 54.386	3.56
8C0G	8 CPU nodes	13,123.833 $\pm$ 24.194	6.43

6.2. Heterogeneous Performance

The heterogeneous configurations in Table 6 show that the accelerator branch does not provide a uniformly faster individual worker on the evaluated platform. One GK210 device takes 116,769.333 s and is slower than a dedicated CPU node. A control execution of PARDISO on the host CPUs of the accelerator node took 118,841.833 s, indicating that the accelerator node also has a lower host baseline. When both GK210 devices are used, elapsed time falls to 57,938.333 s, a factor of 2.02 relative to one device and a speedup of 1.46 relative to the C/PARDISO reference.

Table 6. Heterogeneous performance.

Label	Configuration	Elapsed time (s)	Speedup
REF	C/PARDISO reference, 1 CPU node	84,347.667 $\pm$ 166.567	1.00
0C1G	1 GK210 device	116,769.333 $\pm$ 303.549	0.72
0C2G	2 GK210 devices (K80)	57,938.333 $\pm$ 212.461	1.46
1C1G	1 GK210 + 1 CPU node	51,637.000 $\pm$ 134.030	1.63
3C1G	1 GK210 + 3 CPU nodes	25,296.667 $\pm$ 21.528	3.33
7C1G	1 GK210 + 7 CPU nodes	13,537.000 $\pm$ 20.268	6.23
2C2G	2 GK210 + 2 CPU nodes	26,624.333 $\pm$ 48.227	3.17

The mixed configurations therefore characterize throughput aggregation rather than accelerator superiority. For the same number of logical workers, the CPU-only alternatives remain faster: 2C0G completes in 44,906.500 s versus 51,637.000 s for 1C1G; 4C0G completes in 23,671.667 s versus 26,624.333 s for 2C2G; and 8C0G completes in 13,123.833 s versus 13,537.000 s for 7C1G. These comparisons are not resource-cost equivalences; they show that the dynamic runtime can incorporate a slower worker without assigning it the same amount of work as a faster worker.

The per-point profile in Table 7 explains the observed ratio. On a CPU node, the sparse-solver phase takes 83.874 s of an 84.305 s iteration. On a GK210 worker, the corresponding PaStiX+MAGMA phase takes 110.575 s of a 111.032 s iteration. In both branches, more than 99% of the iteration is therefore spent in the complete sparse-solver stack. The CPU and accelerator branches intentionally represent different end-to-end solver paths; hardware and solver therefore change together, and the comparison is interpreted as the throughput delivered by each worker type rather than as an isolated measurement of GPU offloading.

Table 7. Per-point branch timings.

Branch	Configurations	Solver time (s)	Solver fraction	Iteration time (s)
CPU / PARDISO	1C0G–8C0G, 1C1G–2C2G	83.874	99.49%	84.305
GK210 / PaStiX+MAGMA	0C1G–2C2G	110.575	99.59%	111.032

6.3. Dynamic Load Balance

The performance results above establish that the two worker types have different throughputs; the next question is whether the centralized dynamic assignment converts those unequal rates into useful aggregate throughput. Table 8 shows the number of energy points actually processed by each worker. The CPU-only configurations are nearly uniform, as expected for homogeneous workers. The heterogeneous configurations are not: workers associated with CPU nodes receive more energy points than GK210 workers because they become available more frequently on this platform.

Table 8. Energy-point distribution among ranks.

Label	Configuration	Points processed by each rank
2C0G	2 CPU nodes	500, 501
4C0G	4 CPU nodes	248, 252, 250, 251
8C0G	8 CPU nodes	125, 125, 125, 125, 125, 125, 125, 126
1C1G	1 GK210 + 1 CPU	CPU: 578 GK210: 423
3C1G	1 GK210 + 3 CPU	CPU: 266, 270, 270 GK210: 195
7C1G	1 GK210 + 7 CPU	CPU: 129, 130, 130, 129, 130, 129, 130 GK210: 94
2C2G	2 GK210 + 2 CPU	CPU: 284, 284 GK210: 216, 217

Figure 3 illustrates the most symmetric heterogeneous case, with two CPU nodes and both GK210 devices. Accelerator workers process fewer, longer points, whereas CPU workers process more, shorter points; nevertheless, the four timelines end close to one another. This is the intended behavior of the demand-driven policy: equalize completion time rather than task count.

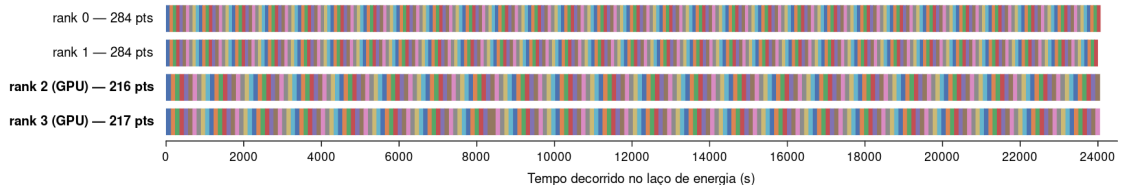
**Figure 3. Energy-point execution timeline.**

Table 9 consolidates the load-balancing metrics of the two instrumented heterogeneous executions, 7C1G and 2C2G.

Table 9. Heterogeneous load-balancing metrics.

Label	Point distribution	t_{CPU}/t_{GPU} (s)	Dynamic makespan (s)	Throughput bound (s)	LB efficiency	Equal-static (s)	Gain	Finish gap
7C1G	CPU: 129–130; GPU: 94	84.807 / 117.001	11,026.171	10,989.391	99.67%	14,625.119	24.6%	72.217 s (0.65%)
2C2G	CPU: 284, 284; GPU: 216, 217	84.878 / 111.243	24,130.712	24,095.600	99.85%	27,810.752	13.2%	73.195 s (0.30%)

In 7C1G, the measured loop makespan lies within 0.33% of the aggregate-throughput bound, giving 99.67% load-balancing efficiency, and is 24.6% below the

equal-static estimate, which the slower GK210 worker determines. In 2C2G, an equal partition of 250 points per rank would take 21,219.587 s on a CPU node but 27,810.752 s on a GK210 worker; the dynamic execution measured 24,130.712 s against the 24,095.600 s bound, giving 99.85% efficiency, a makespan 13.2% below the equal-static estimate, and a finish-time gap of only 0.30% of the makespan. Both calculations use the measured mean per-point times of the instrumented executions.

6.4. Solver Output Consistency

The final requirement is that changing the worker branch must not change the physical result. The C/PARDISO and PaStiX+MAGMA conductance curves were therefore compared point by point over the same 1,001-energy grid, and the figures below are measured end to end between those two branches. The maximum absolute difference is 9.8×10^{-10} , the mean absolute difference is 3.0×10^{-11} , and the root-mean-square difference is 8.5×10^{-11} . The maximum and mean relative differences are 1.0×10^{-10} and 3.2×10^{-12} , respectively. The two-device branch presents the same order of difference, with a maximum relative value of 1.5×10^{-10} .

The differences are consistent with changes in floating-point operation order, sparse permutations, and pivoting decisions between the two solver stacks. Conductance values are of order 10^0 to 10^1 , so the observed differences remain many orders below the scale of the reported result. The heterogeneous performance results therefore do not depend on a reduced-precision or approximate physical formulation.

7. Discussion

The results support three main conclusions. First, the physical problem exposes a large set of independent energy points, each dominated by the same type of sparse MRHS solve. The CPU-only experiment confirms that this coarse-grained parallelism scales effectively: relative to the one-node common runtime, eight CPU nodes achieve speedup 6.62 and 82.7% parallel efficiency, closely matching the Amdahl prediction from the measured parallel fraction. Second, the heterogeneous experiments show that the worker types do not have equal throughput on the evaluated system. A single GK210 is slower per energy point than a dedicated CPU node, so the relevant contribution of the heterogeneous architecture is not an assumption of accelerator superiority but the ability to aggregate whatever capacity is available. Third, the dynamic assignment responds to that asymmetry by changing the number of points assigned to each worker and reaches 99.85% load-balancing efficiency in the 2C2G case.

This behavior also distinguishes the proposed architecture from the tools summarized in Table 1. The proposal does not introduce a new family of dynamic schedulers, nor does it claim a universally faster accelerator solver. Its contribution is to place centralized demand-driven scheduling between the independent NEGF energy points and heterogeneous worker implementations, so that workload allocation is determined by observed completion rather than a predefined hardware weight. Numerical agreement between the two solver branches shows that this scheduling and solver substitution preserve the conductance result within the reported floating-point differences.

The conclusions are limited to the single 1.5° problem instance, the tested CPU nodes, Tesla K80, CUDA 11.0, and the selected solver stacks. Matrix structure, right-hand-side count, and energy-grid size are not varied. Hardware and solver change together

between the CPU and accelerator branches, so the reported comparison characterizes the end-to-end worker types rather than separating the individual contribution of each component. The proprietary input data also limit independent reproduction of the complete physical experiment. These limitations prevent generalizing the measured CPU/GPU throughput ratio to other hardware, while leaving the observed behavior of the demand-driven assignment valid for the reported configurations.

8. Conclusion

This work addressed the computational cost of electronic-transport simulation in Corbino TBG by exploiting the structure produced by the NEGF formulation. For each energy point, the required Green-function columns are obtained from a large complex-symmetric sparse MRHS system; energy points are independent, while all right-hand sides of a point share the same coefficient matrix. The proposed architecture uses these independent energy points as work units and assigns them on demand to heterogeneous workers through centralized dynamic load balancing. The implementation realizes this design with distinct CPU and accelerator solver branches while preserving one factorization followed by a batched MRHS solution.

Eight CPU nodes reach speedup 6.62 and 82.7% efficiency relative to the one-node common runtime, or 6.43 relative to the C/PARDISO reference. The full Tesla K80 reaches 1.46, and both GK210 devices with two CPU nodes reach 3.17, with 99.85% load-balancing efficiency and a makespan 13.2% below the estimated equal-static partition. The two solver branches differ by at most 1.0×10^{-10} in relative conductance.

The results therefore support demand-driven energy assignment as a mechanism for aggregating heterogeneous resources without predefined performance weights. They do not establish accelerator superiority on the evaluated platform: CPU-only configurations remain faster in equal-logical-worker comparisons. The main result is instead that workers with different throughputs can contribute to the same NEGF energy sweep without an equal partition forcing the slower branch to determine the makespan. Broader generalization requires additional problem instances and evaluation on newer heterogeneous hardware.

Acknowledgments

The authors acknowledge support from (i) MackCloud (<https://mackcloud.mackenzie.br/>); (ii) the SPRACE project – Grant No. 2018/25225-9, São Paulo Research Foundation (FAPESP); and (iii) the Mackenzie Research and Innovation Fund (MackPesquisa) – Project No. 231009.

Use of Artificial Intelligence

AI tools assisted in restructuring and condensing the manuscript, drafting preliminary English wording for selected passages (particularly the Abstract, table formatting, and the Conclusion), and checking LaTeX phrasing and consistency. They were not used to collect or fabricate data, perform experiments, calculate the reported metrics, or independently conduct the evaluation, nor to generate the references, which the authors selected and verified independently. The authors critically reviewed, revised, and verified all AI-assisted content, factual claims, citations, and conclusions, and take full responsibility for the final content.

References

- AMReX-Microelectronics (2024). Elqtronex: An open-source, first-principles, and full-band simulator for quantum-transport in electron-phonon-coupled nano-devices. Acessado em: 2025-06-19.
- Assis, I. A. S., Oliveira, A. D. S., Barros, T., Sardina, I. M., Bianchini, C. P., and Xavier De-Souza, S. (2019). Distributed-memory load balancing with cyclic token-based work-stealing applied to reverse time migration. *IEEE Access*, 7:128419–128430.
- Bahamon, D. A., Castro Neto, A. H., and Pereira, V. M. (2013). Effective contact model for geometry-independent conductance calculations in graphene. *Physical Review B*, 88(23):235433.
- Ben-Nun, T., Schneider, T., Ziogas, A. N., Müller, T., Luisier, M., and Hoeffler, T. (2019). Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19*, New York, NY, USA. ACM. Gordon Bell Prize recipient.
- Calderara, M., Brück, S., Pedersen, A., Bani-Hashemian, M. H., VandeVondele, J., and Luisier, M. (2015). Pushing back the limit of ab-initio quantum transport simulations on hybrid supercomputers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15*, New York, NY, USA. ACM.
- Datta, S. (1997). *Electronic transport in mesoscopic systems*. Cambridge University Press, Cambridge.
- Datta, S. (2005). *Quantum transport: atom to transistor*. Cambridge University Press, Cambridge.
- Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. S. (1990). A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17.
- Fernandes, J. B. (2020). Granularity load scheduling auto-tuning for multi-core processors applied to reverse-time migration. Dissertação (mestrado em engenharia elétrica e de computação), Universidade Federal do Rio Grande do Norte, Natal.
- Ferry, D. and Goodnick, S. M. (1999). *Transport in nanostructures*. Cambridge University Press, Cambridge. No. 6.
- Groth, C. W., Wimmer, M., Akhmerov, A. R., and Waintal, X. (2014). Kwant: a software package for quantum transport. *New Journal of Physics*, 16:063065.
- Hoeffler, T. and Belli, R. (2015). Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*, pages 1–12. ACM.
- Hénon, P., Ramet, P., and Roman, J. (2002). PaStiX: A high-performance parallel direct solver for large sparse linear systems. *Parallel Computing*, 28(2):301–321.
- Inria Bordeaux – Sud-Ouest (2024). PaStiX solver documentation.

- Intel Corporation (2023). Intel® oneapi math kernel library (onemkl) documentation.
- Löthman, T., Schmidt, J., Parhizgar, F., Black-Schaffer, A. M., and Lado, J. L. (2022). Nematic superconductivity in magic-angle twisted bilayer graphene from atomistic modeling. *Communications Physics*, 5:92.
- NanoTCAD ViDES (2023a). MPIze - NanoTCAD ViDES documentation.
- NanoTCAD ViDES (2023b). NanoTCAD ViDES. <http://vides.nanotcad.com/vides/>. Accessed: 2025-06-12.
- Novoselov, K. S., Fal'ko, V. I., Colombo, L., Gellert, P. R., Schwab, M. G., and Kim, K. (2012). A roadmap for graphene. *Nature*, 490(7419):192–200.
- NVIDIA Corporation (2026a). cuSOLVER library documentation.
- NVIDIA Corporation (2026b). NVIDIA cuDSS: A high-performance CUDA library for direct sparse solvers — getting started.
- Papior, N. R., Gunst, T., Stradi, D., and Stokbro, K. (2017). The TranSIESTA method for efficient and parallel electronic transport calculations. *Computer Physics Communications*, 212:8–24.
- Passos, A. and Bianchini, C. (2025). Comparação de desempenho de implementações fortran e c: Corbino tbq. In *Anais da XVI Escola Regional de Alto Desempenho de São Paulo*, pages 98–101, Porto Alegre, RS, Brasil. SBC.
- Pellegrini, F. and Roman, J. (1996). SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking (HPCN Europe 1996)*, volume 1067 of *Lecture Notes in Computer Science*, pages 493–498. Springer.
- Santana, C. d. S. (2020). Workload scheduling analysis in geophysical numerical methods. Dissertação (mestrado em engenharia elétrica e de computação), Universidade Federal do Rio Grande do Norte, Natal.
- Sawant, S. S., Léonard, F., Yao, Z., and Nonaka, A. (2025). ELEQTRONeX: A GPU-accelerated exascale framework for non-equilibrium quantum transport in nanomaterials. *npj Computational Materials*, 11:110.
- Scott, J. and Tũma, M. (2023). *Algorithms for sparse linear systems*. Springer, Cham.
- Synopsys (2024). QuantumATK. <https://docs.quantumatk.com>. Accessed: 2025-06-12.
- Synopsys (2025a). Parallelization of QuantumATK calculations - QuantumATK documentation.
- Synopsys (2025b). QuantumATK new release notes.
- Synopsys (2025c). Technical hardware requirements - quantumatk documentation. https://docs.quantumatk.com/intro/installation/technical_hardware.html. Acessado em: 2025-06-14.
- The SIESTA project (2024). TranSIESTA Benchmarks. <https://siesta-project.org/siesta/benchmarks/>. Accessed: 2025-06-12.

Tomov, S., Dongarra, J., and Baboulin, M. (2010). Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36(5–6):232–240.