Chapter

Acceleration of Large-Scale Electronic Structure Simulations with Heterogeneous Parallel Computing

Oh-Kyoung Kwon and Hoon Ryu

Abstract

Large-scale electronic structure simulations coupled to an empirical modeling approach are critical as they present a robust way to predict various quantum phenomena in realistically sized nanoscale structures that are hard to be handled with density functional theory. For tight-binding (TB) simulations of electronic structures that normally involve multimillion atomic systems for a direct comparison to experimentally realizable nanoscale materials and devices, we show that graphical processing unit (GPU) devices help in saving computing costs in terms of time and energy consumption. With a short introduction of the major numerical method adopted for TB simulations of electronic structures, this work presents a detailed description for the strategies to drive performance enhancement with GPU devices against traditional clusters of multicore processors. While this work only uses TB electronic structure simulations for benchmark tests, it can be also utilized as a practical guideline to enhance performance of numerical operations that involve large-scale sparse matrices.

Keywords: offload computing, GPU devices, large-scale electronic structure simulations, tight-binding approach, nanoelectronics modeling

1. Introduction

As the dimension of functional semiconductor devices are scaled down to deca-nanometer (nm) sizes, the underlying material can no longer be considered continuous. The number of atoms in the active device region becomes countable in the range of ~50 K to a few millions, and their local arrangements in interfaces, alloys, and strained systems give non-negligible effects on device characteristics [1–3]. Also, most experimentally realized structures are not infinitely periodic, but are finite in sizes; such geometries call for a local orbital basis, rather than a plane wave basis which implies infinite periodicity. As full *ab-initio* methods such as density functional theory are in principle hard to simulate electronic structures of such a huge and discrete atomic structures [4, 5], the necessity of atomistic approaches based on an empirical modeling method is quite huge.

The *spds** 10-band tight-binding (TB) approach, which employs a set of 10 localized orbital bases to describe a single atom, has been extensively used to explain

experimental behaviors of various quantum devices [2, 6–9] through large-scale electronic structure simulations with the well-known nanoelectronics modeling tool (NEMO) [10, 11]. As the NEMO only runs in traditional computing clusters of multicore processors, we also have recently released a new software package for TB simulations (Quantum simulation tool for Advanced Nanoscale Devices (Q-AND)), which supports computation with Intel Xeon Phi PCI-E add-in devices and shows enhanced performance compared to the result obtained with clusters of Intel Xeon multicore processors [12].

The major purpose of this work is to explore the performance benefits that can be obtained with NVIDIA general-purpose graphical processing unit (GPU) devices, which are also PCI-E add-in devices and are popularly adopted by communities to solve various computing-intensive problems. In particular, (1) we present methodological details applied to enhance the performance of TB electronic structure simulations with GPU devices. Then (2) we show the excellence of speed and scalability for end-to-end simulations of realistically sized nanostructures and (3) analyze the economic benefits of latest GPU devices for TB simulations against computing resources of multicore processors (host CPUs). Extending our previous work with Intel Knights Corner coprocessors [12] to the area of GPU computing, this work delivers practical information for technical details that are employed to accelerate empirical modeling of large-scale electronic structures and therefore can serve as a guideline that is beneficial to researchers in the field of nanoelectronics who consider a code migration to heterogeneous computing platforms involving PCI-E communications, which takes a non-negligible portion of top 500 high-performance computing systems in the world [13]. While latest NVIDIA GPU devices also support NVLink communications, here we only consider the PCI-E one for the performance analysis.

2. Methodology

Electronic structures of target nanostructures are described with a $sp^3d^5s^*$ TB model [6, 9, 10], which employs 10 orthogonal orbital bases to represent a single atom assuming nearest-neighbor couplings. As shown in **Figure 1** (top), simulation domains are decomposed in a multidimensional manner with MPI and OpenMP. Hamiltonian matrices, which are stored in compressed sparse row (CSR) format [14], are then decomposed in a row-wise manner. The Schrödinger equation solver, which computes normal eigenvalue problems in a numerical perspective, is developed with the Lanczos method [15] whose computational bottleneck comes from sparse matrix-vector multiplication (SpMV) [12, 15]. The basic idea for the performance improvement would thus be to perform SpMV with a simultaneous utilization of host CPUs and GPU devices, where **Figure 1** (bottom) illustrates this idea. Here, each GPU device has a block matrix belonging to an MPI process, which sends/receives input (Vin)/output (Vout) vectors to/from the associated GPU device. Each MPI process does not need to send the whole Vin since multiplication in an MPI process can be done with only three block vectors of Vin (1 in itself, 2 in its neighbor processes) as our TB model assumes nearest-neighbor couplings. Upon the completion of multiplication, a GPU device just needs to send 1 block of Vout) back to its associated MPI process. Host CPUs and GPU devices can thus perform multiplication simultaneously with no heavy overhead of data transfer. In the next subsections, we present further detailed methodologies for (1) the simultaneous utilization of host CPUs and GPU devices and (2) the implementation of efficient SpMV CUDA kernels.

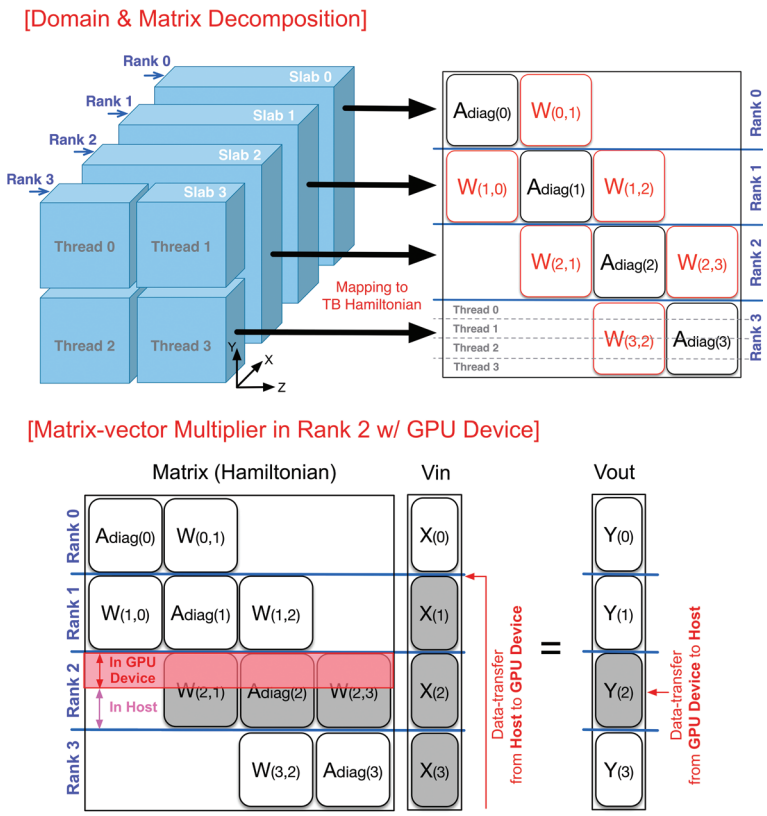

Figure 1.
(Top) scheme for domain decomposition. Hamiltonian matrices representing real-space simulation domains are decomposed in a row-wise manner with a hybrid use of MPI and OpenMP. (bottom) conceptual illustration of how to share the computing load of matrix-vector multiplication into both host CPUs and GPU devices.

2.1 Simultaneous utilization of both CPU and GPU

The following are two ways of how to efficiently utilize the resources of both CPUs and GPUs. One takes the pageable memory when transferring data between CPU and GPU, whereas the other uses the pinned memory. The memory can be allocated in the pinned memory with the *cudaMallocHost* function of CUDA library, which prevents the memory from being paged out and therefore improves the speed of data transfer between host and GPU devices. The pageable (non-pinned) memory can be used with the *malloc* function of standard C library. This subsession will discuss in detail how SpMV can be done with the abovementioned two ways.

Computations of SpMV in host CPU and GPU devices are overlapped in default since the GPU kernel function is called in a non-blocking manner such that its execution can be done in parallel with the execution in CPU code. As shown in **Figure 2(a)**, however, data transfer between host and (GPU) device memory cannot be overlapped with the CPU computation when the pageable memory is used. As depicted in **Figure 2(b)**, the pinned memory enables the CPU calculation to be overlapped with data transfer [16]. Another merit that can be obtained with the pinned memory is that the effective bandwidth of data transfer itself is increased by ~3 times compared to the one obtained with the pageable memory, because the bandwidth of the PCI-E bus to connect CPU and GPU is not fully exploited with the pageable memory [16]. The pinned memory can be used with the *cudaMallocHost* (CUDA library) instead of *malloc* function. As memory offload is much faster and

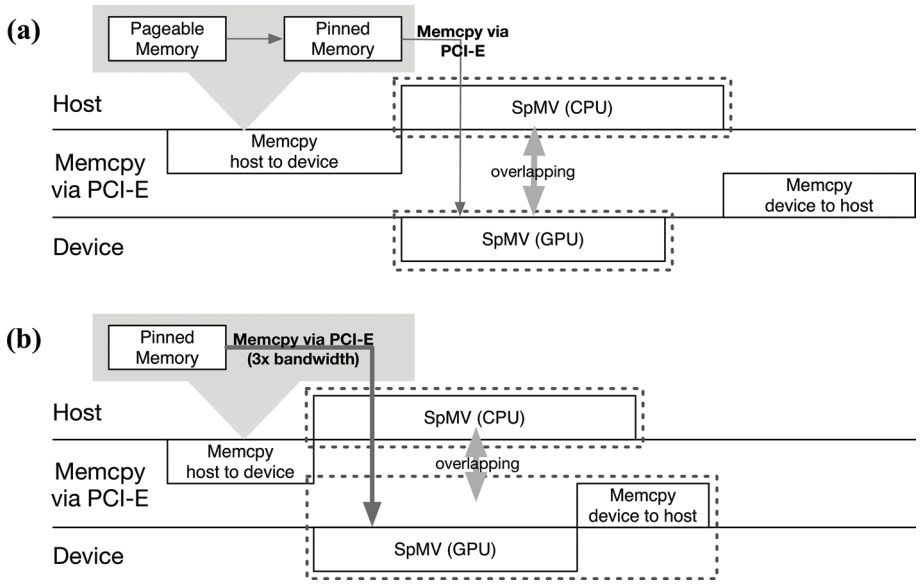

Figure 2.
Comparison of two methodologies for simultaneous utilization of both CPU and GPU. (a) With pageable memory, data transfer between CPUs and GPU devices cannot be hidden behind the SpMV computation. (b) With pinned memory, data transfer can be overlapped with the SpMV computation and can be performed with much higher bandwidth.

communication hiding is possible, the pinned memory would be superior to the page memory for saving the wall time of SpMV calculations with GPU devices.

2.2 CUDA SpMV kernels

We have developed three different CUDA SpMV kernels as illustrated in **Figure 3**: (i) a basic kernel that allocates a single CUDA thread per a row in the matrix (*naïve*), (ii) a kernel that always allocates a single WARP to the SPMV operation for a single row in the matrix (*warp1*) [17], and (iii) a kernel that dynamically allocates multiple WARPs to the SPMV operation for a single row in the matrix (*warp2*) [18].

Firstly, the *naive* kernel is a straightforward approach to map a single CUDA thread to a single row in the matrix. Because the Q-AND code uses the CSR format to describe the Hamiltonian matrix, SpMV operations need an indirect addressing step for every single scalar operation needed for multiplications. Consecutive threads therefore have to access irregularly strided memory as illustrated in **Figure 3(a)**. As noted by Harris [19], such access patterns would degrade performance, because then successive threads may not be able to access the global memory simultaneously to read non-zero elements of the matrix (**Figure 3(a)**).

Secondly, the *warp1* kernel uses the maximum number of CUDA threads of a single WARP for multiplications of a single row in the matrix. A WARP is defined as the group of threads and consists of contiguous threads (32 threads for Tesla K40 devices) [19, 20]. Since threads in a single WARP can access the global memory simultaneously, we can reduce the number of transactions that are required to access the global memory, and therefore we expect non-negligible performance enhancement for SpMV operations against the *naïve* kernel (**Figure 3(b)**) [18]. However, this solution may not be the best one, since we always use a single WARP for a single matrix row, and therefore we may have idle threads (or WARPs) if the maximum number of WARPs supported by a single GPU device is larger than the number of rows of a block matrix that are belonging to that GPU device.

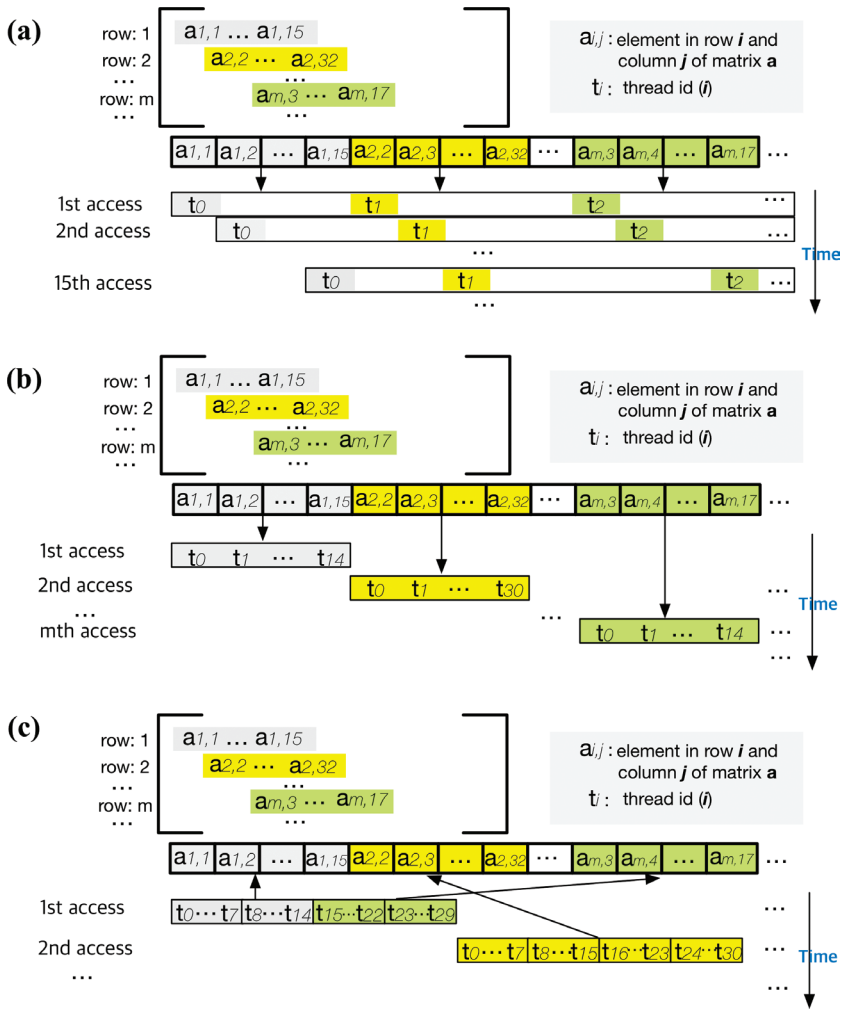

Figure 3.
Conceptual scheme of three SpMV CUDA kernels. (a) A basic kernel that maps a single thread to a single row in the matrix for SpMV (naive). Here, consecutive threads (t_0, t_1,...,t_{n-1}) access nonconsecutive words. (b) A kernel that uses WARP statically (warp1). It always allocates a single WARP (32 threads in Tesla K40) to a single row in the matrix. Consecutive threads (t_0, t_1,...,t_{n-1}) access consecutive words. (c) A kernel that uses WARP dynamically (warp2). It dynamically allocates WARPs to a single row considering the sparsity of matrix.

Thirdly, the *warp2* kernel dynamically allocates WARPs to a single matrix row as depicted in **Figure 3(c)**. Here, the number of WARPs allocated to a single matrix row is dynamically determined by counting the number of corresponding non-zero elements, i.e., an integer value that is closest to "the number of non-zero elements in one matrix row/the number of threads per a single WARP. "We note the *warp2* kernel would be optimal for our problem since the TB Hamiltonian matrix normally has non-zero elements that are larger than 32.

In addition, we can increase the performance by utilizing the texture memory for the vector data retrieval, where texture memory, which is a read-only memory, is cached on-chip and provides higher effective bandwidth by reducing memory requests toward off-chip DRAM. Since the in/out vectors are irregularly accessed by threads from the global memory of GPU devices, the performance improvement could be driven by applying the texture memory. The following section reveals the result of the evaluation.

3. Results and discussions

This section discusses the performance evaluation of our solver from following perspectives: (i) the strong/weak scalability of end-to-end simulations and the optimal *GPU load* (i.e., the portion of SpMV calculation allocated to GPU devices that shows the best speed), (ii) impact of the pinned memory on computing performance, (iii) performances of the three different CUDA SpMV kernels, and (iv) energy efficiency and economic benefit of GPU computing for electronic structure simulations against the results obtained with CPU computing only. All the benchmark tests are performed on the two test-bed machines: one is the K40 test-bed including three computing nodes connected with an infinite-band 4 × FDR (56 Gbps) network, where each node has two Intel Xeon CPUs E5-2650 v3 [21], two NVIDIA Tesla K40 GPU cards [20], and one 128G DDR3 1867 MHz memory, and another is a P100 test-bed including one node with same configuration except two NVIDIA Tesla P100 cards [22]. The codes are compiled with CUDA 8.0 library, Intel C++ compiler 16.0, and OpenMPI 1.10.2. Si:P quantum dots (QDs), which are defined to be huge silicon (Si) layers encapsulating a single phosphorus (P) atom and are studied with a 10-band TB model for designs of Si-based quantum computers [8, 9, 23], are used as target devices for all the benchmark tests.

3.1 Evaluation of utilization of both GPUs and CPUs

Using the pinned memory and the *warp2* kernel with texture memory, simulations are performed for Si:P QDs with a convergence criterion of 10^{-8} eV and are completed when 10^4 Lanczos iterations are reached or 10 lowest energy levels in conduction band are found. Every bar graph of **Figure 4** has the following six elements: MPI communication (*Comm*), data transfer from host to GPU device (*CopyIn*), SpMV + data transfer from GPU device to host (*SpMV + CopyOut*), dot product (*VVDot*), memory operations (*MemOp*), and other portions (*Others*). Note that *SpMV + CopyOut* includes the time required for SpMV on GPU devices and data transfer from GPU device to host memory, while SpMV on CPUs is performed at the same time.

Figure 4(a) and **(b)** presents the strong and weak scalability of the end-to-end simulation at the K40 and P100 test-beds, respectively. The Si:P QD tested for the strong scalability has a cuboid Si layer that consists of a total of 30 × 80 × 80 [100] unit cells and has a dimension of ~16 nm × 43 nm × 43 nm (about 1.5 million atoms). The problem size for the weak scalability test is 15n × 80 × 80 [100] unit cells (n × 7.5 × 10^5 atoms), where n denotes the number of MPI ranks. As we use 10 bases to describe a single atom, the degrees of freedom (DOF) of corresponding Hamiltonian matrices are ~15 million (for strong scalability) and ~7.5 million/rank (for weak scalability), respectively. Here, we see that the strong scalability is generally quite good, where each MPI rank is mapped to 10 CPU cores and one GPU card. The job using six MPI ranks is 2.34 times faster than the one executed with two ranks for the 30 × 80 × 80 unitcells at the K40 test-bed. It shows nice scales according to the number of cores, because a significant portion of the wall time is taken by SpMV that would give a nice scalability as the matrix has a block-tridiagonal shape, and therefore the burden of MPI communications would not become a serious problem. The weak scalability also shows good since the wall time is not significantly affected by MPI communications.

In addition, **Figure 4(c)** and **(d)** illustrates the performance comparison in terms of the wall time according to the GPU load at the K40 and P100 test-beds,

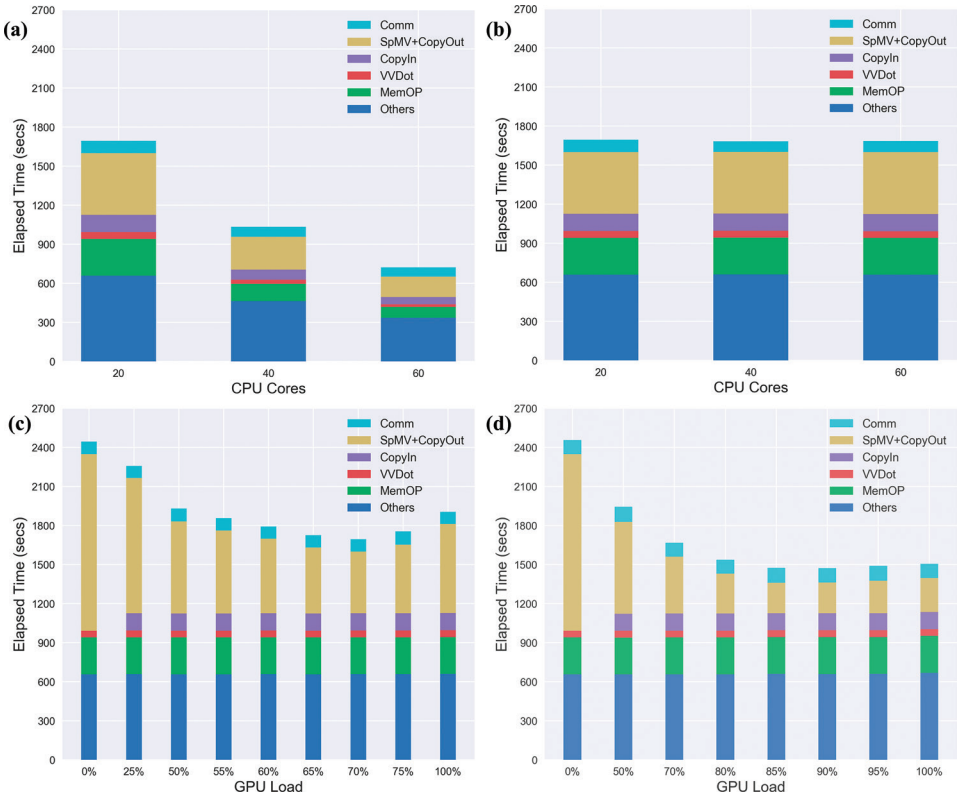

Figure 4.
Performance of Q-AND code with GPUs computing using the pinned memory and the warp2 kernel with texture memory. (a) Strong scalability of computing 30 × 80 × 80 unit cells at the K40 test-bed. (b) Weak scalability of computing 15 × 80 × 80 unit cells per single MPI process at the K40 test-bed. (c) Performance of computing 30 × 80 × 80 unit cells as a function of the GPU load at the K40 test-bed. (d) Performance comparison of computing 30 × 80 × 80 unit cells as a function of the GPU load at the P100 test-bed.

respectively. The QD considered for the experiment here has 30 × 80 × 80 unit cells. The elapsed time is described as a function of computing load for SpMV on GPU devices (GPU load). As described in the previous section, SpMV is the most time-consuming part such that it takes about 56% of the wall time when CPUs perform all the multiplications (*GPU load* is zero) at the K40 test-bed. However, as the GPU load increases, SpMV takes less time and shows the best speed when the GPU load is ~70%. This *optimal* GPU load depends on the hardware performance of GPU devices such that, at the P100 test-bed (with same host CPUs), it is ~90%. The speedup becomes 1.44× and 1.7× for the target simulation at the K40 and P100 test-beds, respectively, against the case when GPU load is zero (only CPUs are used for simulations).

Then let us discuss why this optimal GPU load becomes about 70 and 90% at the K40 and P100 test-beds, respectively. Since the performance of SpMV depends on various factors such as computing units, memory bandwidth and latency, network speed, and PCI-E bandwidth between host and GPU device, it is not easy to clearly extract the exact value of the optimal GPU load. However, the "ideal value" of the *GPU load* could be approximately calculated using only the theoretical peak performance of computing units, because the performance of SpMV would be maximized when both CPUs and GPUs complete computing operations at the same time. If we denote the peak performance (in the unit of floating point operations per second (FLOPS)) of host CPUs and PCI-E

connected devices by P_H and P_D, respectively, the optimal GPU load (x) can be calculated as the following equation (Eq. (1)):

$$x = \frac{100 \times P_D}{P_D + P_H} \tag{1}$$

Since a single computing node of the K40 test-bed has a P_H of about 0.736×10^{12} FLOPS for twenty CPU cores of Xeon E5-2650 v3 [21], and a P_D of about 2.620×10^{12} FLOPS for two Tesla K40 devices [20], x is derived to about 78.1%, which it is a little higher than the measured value (70%) due to the ignorance of other factors (memory bandwidth, etc.). For the P100 test-bed (P_D of about 10.600×10^{12} FLOPS) [22], x is also evaluated to about 93.5%, while we find it at ~90%. Even though the derived values are not strictly accurate, we can still explain why the optimal GPU load of the K40 and P100 test-beds turns out to be higher than the one (~65%) measured with Xeon Phi Knights Corner coprocessors [12].

3.2 Effects of pinned memory on performance

As explained in the previous section, the pinned memory may make a non-negligible impact on the overall performance of large-scale simulations. **Figure 5(a)** shows the performance measured with the pinned and pageable memory at a 70% (K40) and 90% (P100) GPU load, where a single computing node is used with the *warp2* kernel and texture memory. The Si:P QD has 30 × 80 × 80 unit cells. Results indicate that the case with pinned memory shows better performance than the one with pageable memory due to the following two points: (1) The reduction of *CopyIn* time due to the increased bandwidth of PCI-E bus with pinned memory and (2) *SpMV* + *CopyOut* time as communication hiding behind the computation. We observed the effective bandwidth of PCI-E communication is ~3.31 GB/s with the pageable memory on every test-bed, while it reaches ~10.40 GB/s with the pinned memory, driving ~3.14 speedup in data transfer. The effective speed of SpMV operations increases by a factor of 1.36 and 1.19 with pinned memory compared to the speed with pageable memory at the K40 and P100 test-beds, respectively, since utilization of the pinned can overlap computation and data transfer. The performance for end-to-end simulations therefore becomes 1.27 and 1.21 times faster with pinned memory against the performance obtained with pageable memory at 70% GPU load (K40) and 90% GPU load (P100), respectively.

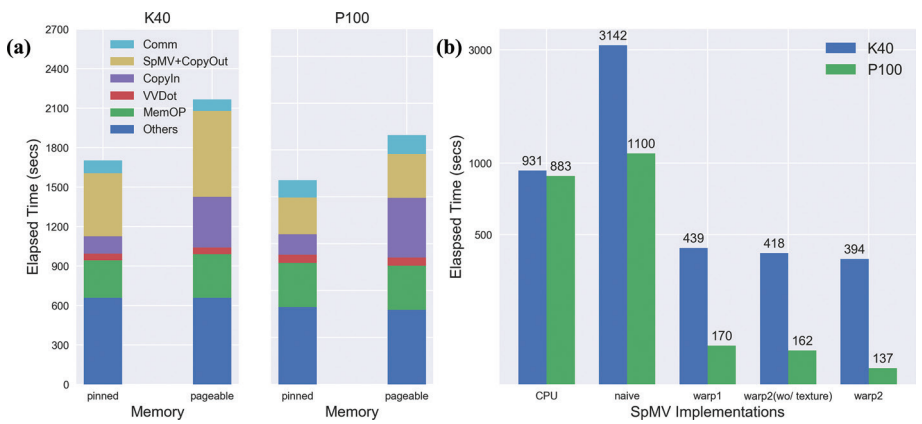

Figure 5.
Performance measured in a single computing node for end-to-end simulations of 30 × 80 × 80 unit cells at the optimal GPU load (70% for K40 and 90% for P100). (a) Performance measured with the pinned and pageable memory when the warp2 kernel is used. (b) Performance of three different SpMV CUDA implementations. SPMV calculations are slightly accelerated with utilization of the texture memory.

Though here we only focused on the performance of PCI-E communications, it is possible to estimate the performance benefit that may be obtained with NVLink communications. For this purpose, we investigate how the bandwidth of communications between CPU and GPU affects the overall performance, where we find that the overall speedup is ~1.21× due to the ~3.1× enhancement of PCI-E bandwidth on the effects of pinned memory in PCI-E add-in P100 devices. Because the bandwidth improvement with NVLink connectivity between CPU and GPU is ~3× compared to the PCI-E [24], we may roughly expect that there will be another ~1.06× speedup for the end-to-end simulation with NVLink add-in P100 devices.

3.3 Performance analysis of SpMV CUDA kernels

Here we investigate the performance of three different SpMV CUDA kernels and present a short discussion about the effects of the texture memory on the performance. **Figure 5(b)** shows the performance of the three SpMV implementations at the single computing node of the K40 and P100 test-beds, where the pinned memory is utilized with a 70% (K40) and 90% (P100) GPU load. The Si:P QD for target simulations has 30 × 80 × 80 unit cells. The grid/block size is set to 21,000/256 and 672,000/256 of the *naive* and *warp1* kernel, respectively. For the *warp2* kernel, the grid/block size is set to 30/1024 and 112/1024 at the K40 and P100 test-beds, respectively, since the number of streaming multiprocessors is 56 for P100 devices, while it is 15 for K40 (the grid size is set to an integer multiple of the number of available streaming multiprocessors).

Among the *naive*, *warp1*, and *warp2* kernel, the *warp2* outperforms as expected. The speedup of the *warp2* kernel is the 7.96/6.73 (K40/P100) and 1.12/1.24 compared to the *naive* and the *warp1* kernel, respectively. The huge performance enhancement that is particularly achieved against the *naive* kernel reflects the importance of coalescing global memory access as Liu et al. also reported that the effective bandwidth is poor for large strided memory access [18]. The *warp2* kernel also works faster than the *warp1* kernel since less threads would be idle with dynamic allocations as discussed in the "Methodology" section. While multiple WARPs can be involved to process a single row in the matrix (and threads in a single WARP can concurrently access the global memory), there is an inter-WARP time lag (only a single WARP can process multiplications at a time). The performance gain, however, is remarkable such that 15–20% of the wall time is reduced with a dynamic allocation of WARPs.

All the three kernels show faster operations in P100 devices than in K40 devices, and the speedup in P100 devices turns out to be ~2.77 on average. Although the peak performance of P100 devices is 4.05× higher than that of K40 devices (in FLOPS for double precision), the measured average performance gain (2.77) is much lower than this value (4.05), since the performance of SpMV is mainly limited by the bandwidth of global memory rather than the core clock of GPU devices [25]. **Figure 5(b)** also shows the performance difference created by utilization of the texture memory for retrieval of vector data retrieval. With the texture memory, the speed of the *warp2* kernel improves by a factor of 1.06 (K40) and 1.19 (P100) against the case without the texture memory, since the texture memory enables fast random accesses to vector data and uses a cache to provide broad bandwidth.

3.4 Energy efficiency and economic benefits of GPU computing

Not only is the elapsed time an important metric, but also the energy efficiency is a significant one to explore. The power usage of host and the two PCI-E connected devices is evaluated as a function of elapsed time (**Figure 6**), where we

consider the power consumed by host (CPU packages with off-chip DRAMs) and Tesla GPU devices. The power usage in host and GPU devices is measured with Intel Running Average Power Limit (RAPL) library [26] and NVIDIA Management Library (NVML) [27], respectively.

Figure 6(a), (b) and **(c)** shows the real-time power consumption of a single computing node at a 0% GPU load (CPU only), 70% GPU load with K40, and 90% GPU load with P100 GPU devices, respectively. A Si:P QD consisting of 30 × 80 × 80 unit cells is simulated with the *warp2* kernel where pinned memory and texture memory are used. Here, all the results show similar patterns such that *(i)* the power consumption starts to increase during the initial processes of electronic structure simulations, i.e., matrix construction that requires memory access to store non-zero elements, row/column indices. *(ii)* The power usage then shows a rapid oscillation during the process of Lanczos iterations, and *(iii)* it finally returns to the normal (standby) value when all the calculation is finished. **Figure 6(d)** informs that the average instantaneous power consumption of a single computing node with K40 and P100 devices is 157.58 and 117.55 Watt, whereas the host of test-beds uses 279.76 and 270.05 Watt, respectively. **Figure 6(e)** shows the total energy consumed by the end-to-end simulation, which can be calculated by multiplying the time-averaged power usage by the wall time. During the execution in a single computing node of the K40 test-bed, CPUs and GPUs consume about 542.32 and 305.40 KJ, respectively, while corresponding values with P100 devices become 331.44 and 144.33 KJ, respectively. ~614.18KJ is consumed for the CPU-only case. Compared to the results measured with K40 GPU devices, a single computing node with P100 devices consumes ~1.34× less energy, while it finishes the target simulation ~2.88× faster. **Figure 6(f)** shows the total energy consumed by the three SpMV kernels in the single computing node of the K40 and P100 test-beds, where the pinned memory is utilized with a 70% (K40) and 90% (P100) GPU load for simulations of a Si:P QD consisting of 30 × 80 × 80 unit cells. Coalescence of global memory access (**Figure 3(c)**) drives a significant performance improvement, such that the *warp2* kernel not only shows the smallest energy consumption but

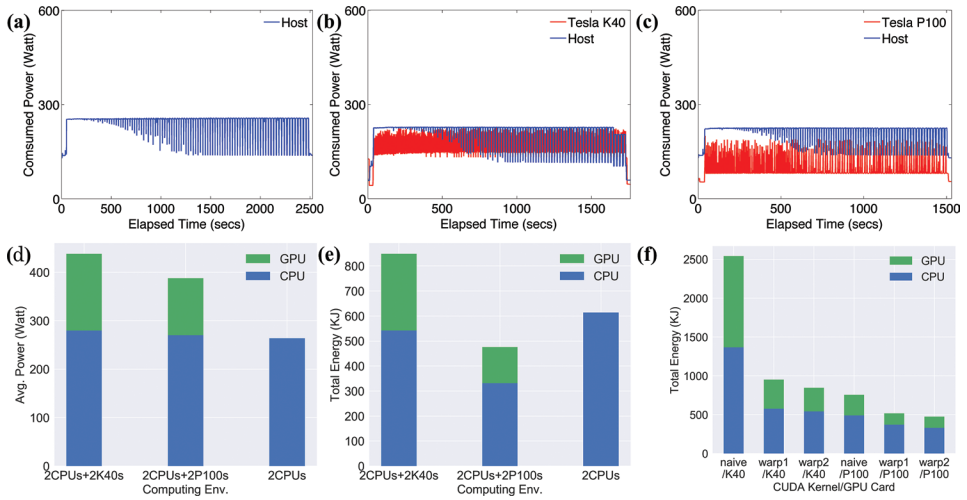

Figure 6.
Power usage and energy consumption associated with the target simulation. The real-time power consumption measured in a single computing node at (a) 0% GPU load (CPU only), (b) 70% GPU load with K40 devices, and (c) 90% GPU load with P100 devices. (d) Time-averaged power usage and (e) total energy consumption measured in a single computing node at the optimal GPU load (70% for K40 and 90% for P100). (f) Total energy consumption of three different SpMV CUDA implementations in a single computing node at the optimal GPU load.

has the best wall-time performance among the three kernels. Reduction in energy consumption of the *warp2* kernel turns out to be 2.99×/1.59× (K40/P100) and 1.12×/1.09× compared to the *naive* and the *warp1* kernel, respectively.

Now let us talk about the energy efficiency of our code for electronic structure simulations. Without losing generality, we roughly can define the energy efficiency as the rate of SpMV operations performed for the unit power consumption (1 W). The rate of SpMV operations can be estimated by the ratio of the total number of floating point operations that a single simulation performs (NF) and the total wall time taken until the simulation is completed. Therefore, the energy efficiency of simulations (η) can be approximated with Eq. (2):

$$\eta = \frac{NF}{T_{total}} \times \frac{1}{W} = \frac{NF}{E_{total}} \tag{2}$$

where E_{total} and T_{total} represent the total energy consumption and wall time required to complete a single simulation, respectively. From the derived equation, the energy efficiency could be calculated for K40 and P100 devices and host CPUs. Although it is not easy to exactly quantify NF, we can at least compare the energy efficiency of different computing devices by assuming that NF would be linearly proportional to the workload of SpMV allocated to specific computing platforms. By setting the energy efficiency of CPU-only computing to 1, the energy efficiency of K40 devices (70% GPU load) and P100 devices (90% GPU load) would be ~1.43 and 3.81, respectively. We conclude the energy efficiency of P100 devices is ~2.66× and 3.81× better than that of K40 and CPU devices for the target simulation. Note that these quantities are hard to be obtained just with officially known hardware specifications [20–22].

Finally, let us close this section with a short discussion about the economic benefits that can be delivered by GPU computing for TB simulations. Since GPU devices are not cheap [28, 29], it would be interesting to compare the "time saving" achieved by a single US dollar spent for additional GPU devices. As we already have shown in **Figure 4**, the CPU-only simulation of 30 × 80 × 80 unit cells is finished in ~2476 s, which is average of the results measured with K40 and P100 devices (**Figure 5(c)** and **(d)**). While the simulation is finished in ~1701 s with K40 at the optimal GPU load (70%), we must additionally pay ~4.6 K US dollars to buy two K40 GPU devices [28]. With P100 devices, the simulation takes ~1472 s at the optimal GPU load (90%) and requires ~14.7 K US dollars to buy two P100 GPU devices [29]. Thereby, we get ~0.17 and ~0.07 s/USD for K40 and P100 devices, respectively. While the performance enhancement driven by GPU computing may be impressive in the perspective of computing time, we claim more expensive devices may not always deliver better economic benefits. Readers are therefore strongly encouraged to build a careful budget plan whether they are thinking to buy new GPU devices.

4. Conclusion

The cost efficiency of general-purpose graphical processing unit (GPU) devices for tight-binding (TB) simulations of extremely large-scale electronic structures has been examined with a focus on the speed and the amount of energy consumption. Technical strategies used to exploit the strength of GPU-coupled offload computing have been elaborated in detail with a short but clear description of the main numerical method employed to tackle large-scale Schrödinger equations. Benchmark tests have been performed against realistically sized solid Si:P quantum dot devices that contain several million atoms. Tesla K40 and latest P100 GPU devices are

considered as the test platform. The technics we employed for the efficient offload computing of large-scale TB simulations drive a non-negligible enhancement of the computing speed. Compared to the performance tested with Intel Xeon V3 host CPU only, K40 and P100 devices can achieve up to ~2× and ~6× speedup for sparse matrix-vector multiplication (SpMV), which is the numerical operation needed to solve electronic structures. In terms of the amount of total energy consumption, however, K40 shows worse performance compared to the CPU-only case, while P100 still holds the strength.

Acknowledgements

This work has been carried out as Intel Parallel Computing Centre (IPCC) project under the financial support from Intel Corporation, USA. Authors acknowledge the extensive use of KISTI Accelerator Test-bed (KAT) computing resources that are supported by Korea Institute of Science and Technology Information.

Author details

Oh-Kyoung Kwon† and Hoon Ryu*†
Korea Institute of Science and Technology Information, Daejeon, Republic of Korea

*Address all correspondence to: elec1020@kisti.re.kr

† These authors contributed equally.

References

[1] Shinada T, Okamoto S, Kobayashi T, Ohdomari I. Enhancing semiconductor device performance using ordered dopant arrays. Nature. 2005;**437**:1128-1131

[2] Usman M, Ryu H, Woo I, Ebert DS, Klimeck G. Moving toward Nano-TCAD through multimillion-atom quantum-dot simulations matching experimental data. IEEE Transactions on Nanotechnology. 2009;**8**:330-344

[3] Lee S, Ryu H, Campbell H, Hollenberg LCL, Simmons MY, Klimeck G. Electronic structure of realistically extended atomistically resolved disordered Si:P δ-doped layers. Physical Review B. 2011;**84**:205309

[4] Carter DJ, Warschkow O, Marks NA, McKenzi DR. Electronic structure models of phosphorus δ-doped silicon. Physical Review B. 2009;**79**:033204

[5] Carter DJ, Marks NA, Warschkow O, McKenzi DR. Phosphorus δ-doped silicon: Mixed-atom pseudopotentials and dopant disorder effects. Nanotechnology. 2011;**22**:1-10

[6] Ryu H, Lee S, Weber B, Mahapatra S, Hollenberg LCL, Simmons MY, et al. Atomistic modeling of metallic nanowires in silicon. Nanoscale. 2013;**5**:8666-8674

[7] Weber B, Mahapatra S, Ryu H, Lee S, Fuhrer A, Reusch TCG, et al. Ohm's law survives to the atomic scale. Science. 2012;**335**:64-67

[8] Ryu H, Lee S, Fuechsle M, Miwa JA, Mahapatra S, Hollenberg L, et al. A tight-binding study of single-atom transistors. Small. 2015;**11**:374-381

[9] Fuechsle M, Miwa JA, Mahapatra S, Ryu H, Lee S, Warschkow O, et al. A single-atom transistor. Nature Nanotechnology. 2012;**7**:242-246

[10] Klimeck G, Shahid Ahmed S, Bae H, Kharche N, Clark S, Haley B, et al. Atomistic simulation of realistically sized nanodevices using NEMO 3-D—Part I: Models and benchmarks. IEEE Transactions on Electron Devices. 2007;**54**:2079-2089

[11] Lee S, Ryu H, Jiang Z, Klimeck G. Million atom electronic structure and device calculations on peta-scale computers. In: Proceedings of 13th International Workshop on Computational Electronics (IWCE). 2009. pp. 1-4. DOI: 10.1109/IWCE.2009.5091117

[12] Ryu H, Jeong Y, Kang J-H, Cho KN. Q-AND: Time-efficient modelling of tight-binding electronic structures with many-core computing. Computer Physics Communications. 2016;**209**:79-87. DOI: 10.1016/j.cpc.2016.08.015

[13] Top 500 Supercomputer Sites. Available from: https://www.top500.org/ [Accessed: 03-04-2018]

[14] Buluç A, Fineman JT, Frigo M, Gilbert JR, Leiserson CE. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In: Proceedings of the Annual Symposium on Parallelism in Algorithms and Architectures (SPAA). 2009. pp. 233-244. DOI: 10.1145/1583991.1584053

[15] Lanczos C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. Journal of Research of the National Bureau of Standards. 1950;**45**:255-282

[16] Harris M. How to Optimize Data Transfers in CUDA C/C++, NVIDIA PARALLEL FORALL. 2012. Available from: https://devblogs.nvidia.com/parallelforall/how-optimize-data-transfers-cuda-cc/ [Accessed: 02-03-2018]

[17] Bell N, Garland M. Efficient Sparse Matrix-Vector Multiplication on CUDA, NVIDIA Technical Report NVR-2008-004. 2008

[18] Liu Y, Schmidt B. LightSpMV: Faster CSR-based sparse matrix-vector multiplication on CUDA-enabled GPUs. In: 26th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2015). 2015. pp. 82-89

[19] Harris M. How to Access Global Memory Efficiently in CUDA C/C++ Kernels, NVIDIA PARALLEL FORALL. 2013. Available from: https://devblogs.nvidia.com/parallelforall/how-access-global-memory-efficiently-cuda-c-kernels/ [Accessed: 02-03-2018]

[20] NVIDIA Tesla K40 GPU Accelerator. Available from: http://www.nvidia.com/content/PDF/kepler/Tesla-K40-PCIe-Passive-Board-Spec-BD-06902-001_v05.pdf [Accessed: 02-03-2018]

[21] Intel Xeon Processor E5-2650 v3. Available from: https://ark.intel.com/products/81705/Intel-Xeon-Processor-E5-2650-v3-25M-Cache-2_30-GHz. [Accessed: 02-03-2018]

[22] Whitepaper of NVIDIA Tesla P100 GPU Accelerator. Available from: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf [Accessed: 02-03-2018]

[23] Weber B, Tan YHM, Mahapatra S, Watson TF, Ryu H, Rahman R, et al. Spin blockade and exchange in coulomb-confined silicon double quantum dots. Nature Nanotechnology. 2014;**9**:430-435

[24] Comparing NVLink vs PCI-E with NVIDIA Tesla P100 GPUs on OpenPOWER Servers. Available from: https://www.microway.com/hpc-tech-tips/comparing-nvlink-vs-pci-e-nvidia-tesla-p100-gpus-openpower-servers/ [Accessed: 10-07-2018]

[25] Xu S, Xue W, Lin HX. Performance modeling and optimization of sparse matrix-vector multiplication on NVIDIA CUDA platform. Journal of Supercomputing. 2013;**63**:710-721. DOI: 10.1007/s11227-011-0626-0

[26] Rountree B, Ahn D, de Supinski B, Lowenthal D, Schulz M. Beyond DVFS: A first look at performance under a hardware-enforced power bound. In: Proceedings of IEEE international parallel and distributed processing symposium workshops & PHD forum (IPDPSW). 2012. pp. 947-953. DOI: 10.1109/ IPDPSW.2012.116

[27] NVIDIA management Library (NVML). Available from: https://developer. nvidia.com/nvidia-management-library-nvml [Accessed: 02-03-2018]

[28] Price of NVIDIA Tesla K40 Computing Processor GPU Cards. Available from: https://www.amazon.com/NVIDIA-Computing-Processor-Graphic-900-22081-2250-000/dp/B00KDRRTB8 [Accessed: 02-03-2018]

[29] Price of NVIDIA Tesla P100 Computing Processor GPU Cards. Available from: https://www.microway.com/hpc-tech-tips/nvidia-tesla-p100-price-analysis/ [Accessed: 02-03-2018]