# Computational Modelling of Inelastic Neutron Scattering for Nanomaterial Characterisation in GPU architectures

Michael T. Garba[a,*], Horacio González–Vélez[a,*], Daniel L. Roach[b]

[a] *IDEAS Research Institute, Robert Gordon University, Aberdeen AB25 1HG, United Kingdom*
[b] *Physics and Materials Research Centre, University of Salford, Salford M5 4WT, United Kingdom*

**Abstract**

Inelastic neutron spectroscopy (INS) is a widely used probe of the vibrational characteristics of materials in condensed matter research; coherent inelastic neutron scattering (CINS) is typically restricted to single-crystal samples, as the analysis of the very complex data sets obtained from coherent inelastic neutron scattering on polycrystalline samples (poly-CINS) remains challenging for all but the simplest of structures. However, with the application of computationally intensive calculation methods (sampling tens of thousands of q-points for a given model) to the simulation of poly-CINS, it is now becoming possible to interpret such data sets by means of comparison and fitting of experimental data to theoretical models. A poly-CINS modelling package developed for the General Utility Lattice Program (GULP), SCATTER has been successfully deployed in multi-core and multi-node architectures. This paper describes a new high-performance implementation of the SCATTER code that provides the ability to generate theoretical poly-CINS data sets from semi-empirical and *ab initio* models in graphics processing unit (GPU) architectures. We present the computing framework behind the GPU implementation, applying an example of a semi-empirical model for the dynamics of two (low and ambient temperature) phases of solid C60 to illustrate the methodology and its successful scalability.

*Keywords:* Inelastic Neutron Scattering, Numerical Linear Algebra, Parallel Computing, General Purpose Computation on Graphics Processing Units, Simulation, Modeling

## 1. Introduction

Traditionally, inelastic neutron scattering measurements have involved either incoherent scattering from polycrystals or coherent scattering (CINS) from single crystals. The reason that CINS from polycrystals has not been employed to a significant extent is that the process of integrating the scattering intensity over crystalline orientations is complex and tends to obscure the useful information available from the direct measurement of dispersion curves using a Triple Axis Spectrometer.

Many important materials, particularly nano-materials, can only be obtained in a polycrystalline form and hence it is of interest to investigate new methods of interpreting the coherent scattering from such samples. Due to its complex nature, the analysis of experimental data sets obtained by the use of modern neutron spectrometers requires new innovative approaches underpinned by advanced computational infrastructures.

Materials researchers already employ neutron scattering simulations as a means of validating and refining their models [1, 2]. Nevertheless, scant research has been devoted to the systematic application of computational solutions for the modelling of polycrystalline materials in current simulation packages (such as $\alpha$-CLIMAX [3] or PHONON) that require model output from *ab initio* software, such as CASTEP [4] or VASP. The SCATTER code provides the capability to generate poly-CINS modelling data using semi-empirical potential models (as well as output from the DFT codes mentioned) via the General Utility Lattice Program (GULP) software package [5, 6], a popular lattice dynamics and simulation package in the materials science community. This capability is crucially important when the computational cost associated with large unit cell models (common for many nanomaterials) is further increased by the need for fine grain sampling of reciprocal space to create theoretical data sets which compare with those generated by modern spectrometers. Such computational costs can arguably be matched by efficiently using modern parallel architectures.

This paper explores the deployment of the SCATTER code on a range of modern parallel architectures for the computational modelling of nanostructures. Its contributions are the formal introduction of the initial parallel implementation of poly-CINS modelling, the effective exploitation of different parallel architectures–multi-core, multinode and graphics processing unit (GPU)–and, ultimately, the modelling of nanomaterials, a crucial element in the synthesis of new high performance materials.

*Corresponding author
*Email addresses:* m.t.garba@rgu.ac.uk (Michael T. Garba), h.gonzalez-velez@rgu.ac.uk (Horacio González–Vélez), d.roach@salford.ac.uk (Daniel L. Roach)

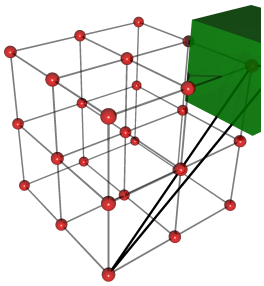


Figure 1: Reciprocal lattice vectors $Q$, $q$ and $\tau$ and the Brillouin zone

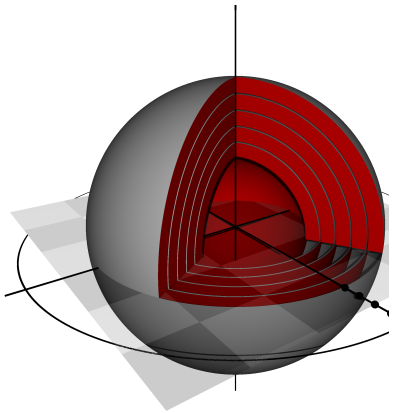


Figure 2: Reciprocal Space Onion Sampling in SCATTER. Concentric shells are traced in reciprocal space as data points are gathered for increasing magnitudes and orientations of the momentum transfer vector, Q, in spherical polar coordinates.

## 2. Inelastic Neutron Scattering from Polycrystalline Materials

The purpose of an INS experiment is to determine the scattering function $S(Q,\omega)$ which carries information about the relative positions and motions of the atoms in a target specimen. Neutron spin and isotope effects result in random variations in the neutron scattering amplitudes of nuclei, causing the scattering to be divided into coherent and incoherent parts. The coherent component, depending on the average value of the scattering amplitude, contains all the information about the relative positions and motions of the nuclei taken in pairs. The incoherent scattering contribution depends only on the motions of each atom taken independently. As described in Van Hove's seminal writings [7], the resulting cross section can be expressed in terms of the corresponding scattering functions, and for the coherent and incoherent scattering functions respectively. These functions, which depend only on the interactions between the nuclei, define the corresponding cross sections (for materials containing only one element) as in Eqns. (1) and (2) [8].

In Eqns. (1) and (2), $S_{coh}(Q,\omega)$ and $S_{inc}(Q,\omega)$ are the respective coherent and incoherent scattering functions, $N$ represents the number of nuclei in the scattering system, for phonon mode $s$, reciprocal lattice vector $\tau$, scattering length $b_d$, atomic mass $M_d$, Debye-Waller factor $W_d$, momentum transfer vector $\mathbf{Q}$, atomic position $r_d$, for atom $d$, polarisation vector $\mathbf{e}_{ds}$, frequency $\omega$, and phonon wavevector $\mathbf{q}$ with neutron energy gain/loss $\langle n_s + \frac{1}{2} \mp \frac{1}{2} \rangle \, \delta(\omega \pm \omega_s)$.

For a momentum transfer vector $\mathbf{Q}$ representing the momentum change between incident and scattered wave vectors, and a vibrational frequency of the quantised lattice vibration (or phonon) created or annihilated by the scattering event, the frequency change is directly related to the modulus of the energy transfer between the target material and the scattered neutron, as determined by energy conservation and the principle of detailed balance [8, 9].

## 3. Background

Based on the work of Roach [10], SCATTER models the interaction of neutrons incident on single crystal and polycrystalline samples in reciprocal space, predicting the coherent and incoherent scattering intensities of Eqns. (1) and (2) by extensive use of GULP [11, 9]. The coherent component of the scattering intensity in Eqn. (1) takes into account cross-correlative pairwise interactions of the nuclei in the system and describes inelastic interference effects which provide information on the positions and vibrational modes of planes of atoms. Equation (2) represents the self-correlative, incoherent component of the scattering intensity, the vibrational contributions of individual atoms considered in isolation, without this interference term. SCATTER is capable of determining both cross-correlative and self-correlative components, however the primary application of this method is for coherent scattering computations and the majority of simulation runs currently involve this [11].

For single crystal samples in an experimental configuration, it is satisfactory to perform these calculations along fixed directions in reciprocal space. However, general application to materials which may only be available in polycrystallline form requires calculation over the full range of magnitudes and spatial orientations of the momentum transfer vector $\mathbf{Q}$ in three dimensions.

SCATTER implements several space sampling techniques that determine the closest corresponding lattice vectors $\mathbf{q}$, and the Brillouin zones for each $\mathbf{Q}$. The Reciprocal Space Onion (RSO) sampling method, illustrated in Figure 2, takes values of $\mathbf{Q}$ as it is rotated about a series of concentric spheres of varying magnitude $|\mathbf{Q}|$ at angles $\theta$ and $\phi$ in a

$$S_{coh}(Q,\omega) = \frac{1}{2N}\sum_{s}\sum_{\tau}\frac{1}{\omega_s}\left|\sum_{d}\frac{\bar{b}_d}{\sqrt{M_d}}\exp(-W_d)\exp(i\mathbf{Q}\cdot\mathbf{r}_d)(\mathbf{Q}\cdot\mathbf{e}_{ds})\right|^2 \times\left\langle n_s+\frac{1}{2}\mp\frac{1}{2}\right\rangle\delta(\omega\pm\omega_s)\delta(\mathbf{Q}\pm\mathbf{q}-\tau) \quad (1)$$

$$S_{inc}(Q,\omega) = \sum_{d}\left\{\overline{b_d^2}-(\overline{b_d})^2\right\}\frac{1}{2M_d}\exp(-2W_d)\times\sum_{s}\frac{(\mathbf{Q}\cdot\mathbf{e}_{ds})}{\omega_s}\left\langle n_s+\frac{1}{2}\mp\frac{1}{2}\right\rangle\times\delta(\omega\pm\omega_s) \quad (2)$$

spherical polar coordinate system [9]. RSO traces concentric shells that correspond to the movement of the triple-axis spectrometer as it samples scattering contributions at various orientations around the target in an experimental setting.

After determining the scattering contributions for each sampled point, SCATTER performs a polycrystalline averaging. The output may be post-processed with separate visualisation tools to generate 3D plots (Figure 7) and directly compared with empirical data. The investigator's ability to arrive at deductions on the basis of the model is significantly enhanced by the generation of a computational log that details the progression of intermediate computations at several stages.

In practice, the quality of results obtained is improved by the use of a greater number of sample points. Artifacts may otherwise begin to appear in the final output for higher values of $|\mathbf{Q}|$ where space sampling is increasingly sparse. Significant computational load results from the invocation of several GULP routines that generate and process large dynamical matrices for each determination of Eqn. (1) over the full range of values for $\mathbf{Q}$ generated by space sampling. A trade-off is typically necessary between the desired model resolution and the actual execution time.

On the other hand, from a computational perspective, GPU modules have become a frequent presence in new high performance computing platforms on account of their substantial computing potential for the kind of intensive tasks that occur in numerical linear algebra applications [12, 13]. Offloading computation to these devices may alleviate the imperative demand created by increasingly complex SCATTER models. However, GPU kernels with complex control flow, conditional branching and divergent thread execution paths incur a noticeable performance penalty [14, 15].

Despite a number of emerging GPU numerical libraries, no library for eigensystem analysis is available to completely satisfy our requirements. Admittedly, the more modern LAPACK, which has largely superseded the original EISPACK code [16], may have formed a functionally superior basis [17]. However, the inherent architectural complexity and reliance on an efficient BLAS implementation implies a long-term effort that the immediacy of our requirements does not allow. The MAGMA library is such an effort that is in the early stages of providing hybrid multicore-CPU/GPU implementations of LAPACK routines [18].

The challenges of achieving efficient performance on a GPU architecture can arguably justify the development of custom algorithms suited to the strengths and limitations of the platform [19]. However, we maintain the original algorithms of the legacy EISPACK implementation as this port is motivated by a very practical application for which the EISPACK eigensolver has proven adequate. Furthermore, the accuracy and numerical characteristics of EISPACK have been established by exhaustive application and testing over nearly 40 years of productive use in computational physics [20].

The problem of creating a data-parallel GPU version is conceptually similar to that of creating a vector-processor version of the EISPACK routines. A vector implementation was created for for the IBM 3090VF by Cline and Meyering [21]. While alternative algorithms used in LAPACK may possess superior cache usage characteristics and performance in modern processor configurations, they provide this at the expense of software complexity and reliance on an efficient BLAS implementation.

Current routines in EISPACK and LAPACK are developed to efficiently solve dense linear systems. However, the sparse and predictable structure of the dynamical matrix suggests that the potential exists to apply iterative solvers for sparse systems, such as ARPACK [22], that may reduce the computational cost of diagonalisation. Furthermore, since successive dynamical matrices vary by a bounded Hermitian perturbation, the variation in eigenvalues is similarly bounded by the spectral norm of the perturbation [23]. While, the eigenvectors share no such relationship, we are exploring alternative approaches to diagonalisation with the use of iterative solvers and custom pre-conditioners that are specific to this problem.

Therefore, we strongly believe that a custom, optimised port of the required functional subset of EISPACK to GPUs. It is noteworthy to mention that any optimisation approach must be applicable to a wider range of models and, effectively, reduce the dominant aspect of the computation to eigenvector and eigenvalue determination, leaving a standard numerical linear algebra problem for which efficient numerical solution techniques, as extensively discussed in [24, 25].

## 4. Methods

RSO sampling yields a discrete grid of points in spherical polar coordinates over which SCATTER evaluates the scattering contributions. These evaluations are computationally demanding for systems of even moderate complexity. Therefore, the practical feasibility of INS modelling is heavily dependent on the availability of a high performance implementation. To achieve this we have undertaken performance optimisations and parallelisation targeting shared memory multicore and distributed memory multinode architectures with support for general-purpose computing on GPUs based on the CUDA platform.

The overall optimisation shown in subsection 4.2 reduces the computationally dominant aspects of SCATTER to the determination of polarisation vectors and frequencies of the phonon modes. These are respectively the eigenvectors and eigenvalues of the Hermitian dynamical matrix for each point in RSO space [8]. SCATTER, as part of GULP, makes use of the EISPACK and LAPACK standard libraries for linear algebra.

### *4.1. Predicting* SCATTER *Performance*

We can predict the runtime of a SCATTER model from the relationship of Eqn. (3) where $t$ is the approximate completion time, $k$ is a constant for a given execution environment and model, $|\mathbf{Q}|_{max} - |\mathbf{Q}|_{min}$ is the difference between the maximum and minimum magnitude of the momentum transfer vector $\mathbf{Q}$, $\delta\mathbf{Q}$ is the finite increment in momentum transfer between successive RSO shells and $\delta\theta = \delta\phi$ is the finite change in angular orientation of the momentum transfer vector.

$$t \approx k \left( \frac{|\mathbf{Q}|_{max} - |\mathbf{Q}|_{min}}{\delta|\mathbf{Q}|} \right) \times \left( \frac{2\pi}{\delta\theta} \right)^2 \qquad (3)$$

The runtime $t$ is determined by the number of points $P(|\mathbf{Q}|, \theta, \phi)$ in RSO space for a given model resolution as specified in the parametric inputs to the program. These integer-valued parameters are $\left( \frac{|\mathbf{Q}|_{max} - |\mathbf{Q}|_{min}}{\delta|\mathbf{Q}|} \right)$ for the number of shells and $\frac{2\pi}{\delta\theta} = \frac{2\pi}{\delta\phi}$ for the number of angular steps. Figure 3 compares predicted and actual problem scaling for a 60-atom model. Equation 3 allows the estimation of full model runtime by calibration against a low-resolution test case.

### *4.2. Optimisation*

For each unique triple ($|\mathbf{Q}|$,$\theta$,$\phi$), corresponding to a point $P$ in RSO-sampled space, GULP derives a new dynamical matrix of phased second derivatives and proceeds to compute the associated phonon modes required by SCATTER. The repeated derivation of the dynamical matrix corresponding to each of these points in the spherical grid is a computationally expensive operation for potential models that require large numbers of nearest neighbour interactions.

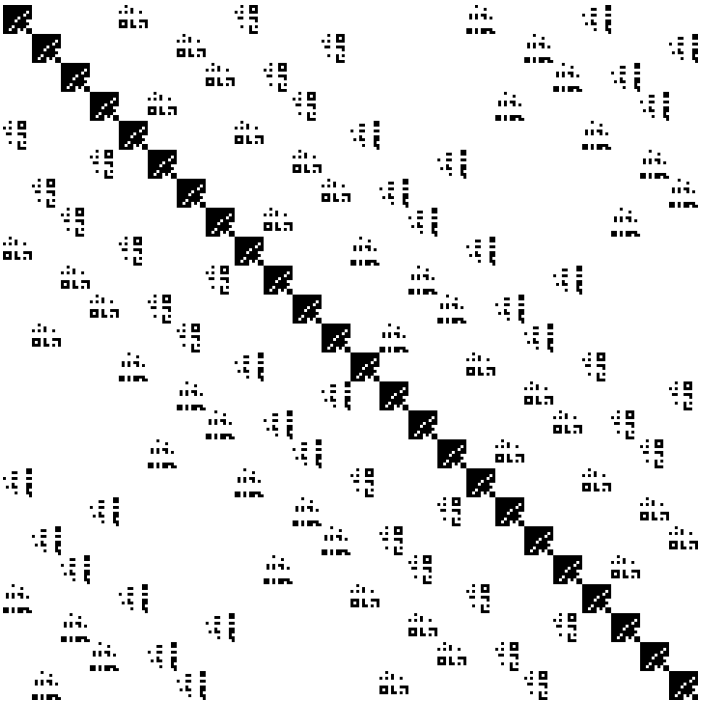


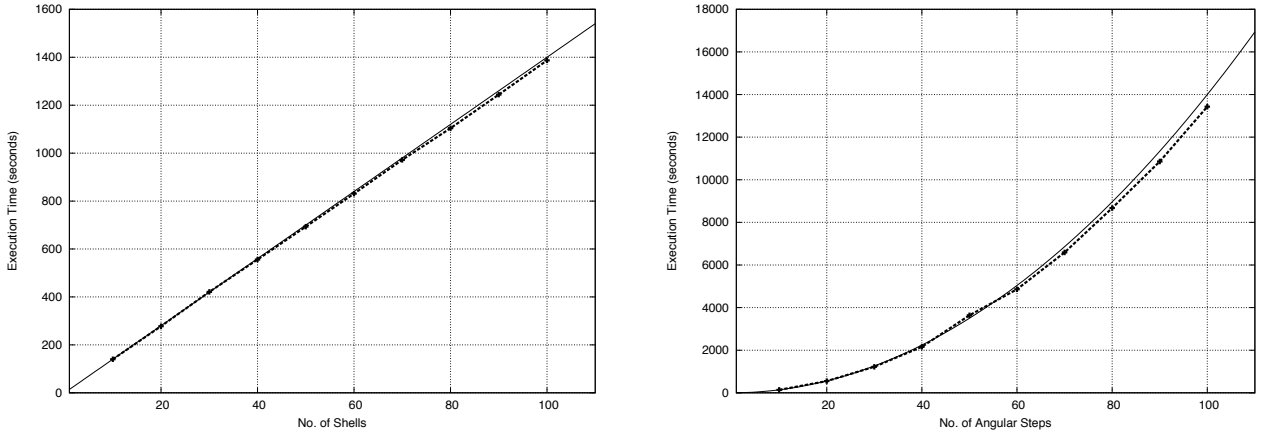
Figure 4: Complex non-zero entries in the dynamical matrix for a 240-atom Carbon nanotube model with 720 modes. The dynamical matrix is Hermitian and sparse.

Figure 4 is a visualisation of a representative dynamical matrix based on the potential model of D.W. Brenner [26] for a 240-Carbon atom system. It illustrates a pattern of sparsely regular non-zero blocks corresponding to the pairwise derivatives of the $n$th nearest neighbour atoms at a given phase angle. The Brenner potential is notably expensive to calculate in GULP as it accounts for all possible neighbour interactions. As an optimisation strategy, we cache the intermediate first and second order derivative vectors for each atom in a space-efficient dynamic linked list to avoid recalculation. Subsequent dynamical matrices are generated by summation of the cached vectors at the appropriate phase angle.

This optimisation significantly lowers the computational cost for the class of models that use the Brenner potential, yielding model-dependent performance increases of between 10× to 50× in overall runtime (Figure 5).

### *4.3. Parallelisation*

The evaluations of scattering for points in RSO-sampled space are independent operations that allow the adoption of several possible parallel partitioning schemes. For simplicity, we choose a cyclic space decomposition, with a round-robin assignment of circles of constant $\phi$ between processes. This decomposition is illustrated in figure X and provides sufficient independent work units to scale to a large number of parallel processes. As a potential optimisation, block partitioning may eliminate some of the redundancy associated with the calculation of energies for




(a) $\left(\frac{|\mathbf{Q}|_{max}-|\mathbf{Q}|_{min}}{\delta\mathbf{Q}}\right)$ shells (b) $\frac{2\pi}{\delta\theta}$ angular steps

Figure 3: Predicted vs. actual SCATTER problem scaling by resolution for a 60-atom Carbon model with (a) $\left(\frac{|\mathbf{Q}|_{max}-|\mathbf{Q}|_{min}}{\delta\mathbf{Q}}\right)$ shells demonstrating linear scaling and (b) $\frac{2\pi}{\delta\theta}$ angular steps in $\theta$ and $\phi$ demonstrating quadratic scaling. The predicted scaling is denoted by the solid line.

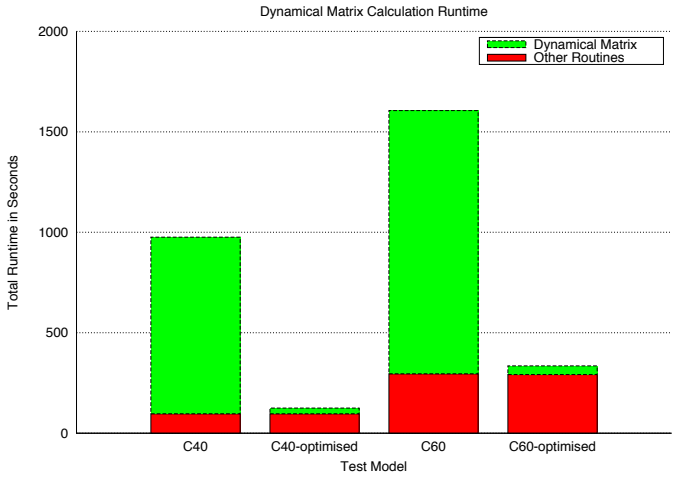



Figure 5: Pre and Post Optimisation Runtimes for models

a given value of $\theta$, however this offers only marginal performance gains for current models when other optimisations are applied.

A final parallel summation reduction over the polycrystalline averages of the contributions calculated by the independent processes determines the scattering function $S(Q,\omega)$ for the model and concludes the simulation

## 5. Implementation and Verification

As part of the General Utility Lattice Program (GULP), SCATTER relies on the underlying lattice dynamics functionality to perform the bulk of the computations necessary. The most computationally significant aspects of the simulation are based on this functionality, we have optimised and adapted components of GULP to deliver higher

Figure 6: Parallel partitioning. Circles of constant $\phi$

performance.

### *5.1. MPI*

We base our parallel implementation of SCATTER on the SPMD approach taken in GULP. The absence of data dependencies between successive evaluations of Eqns. (1) and (2) allows a replicated-data-with-reduction pattern on a supporting a task-parallel algorithmic structure.

Program input is distributed to cooperating processes by the root process in a MPI broadcast operation at program initiation. With complete details of all execution parameters, each process independently computes $S(Q,\omega)$ over an appropriate subset of the global sample space in a rank-based domain-decomposition of the primary SCATTER loop iteration space. A final global reduction operation communicates these local contributions to the root process in a summation that merges the data generated from each process into a final polycrystalline average for the scattering system before output.

As the intermediate results of these calculations are of analytical importance, MPI/IO provides scalable, distributed, simultaneous output of this large dataset from the parallel processes. MPI/IO allows nodes in a cluster environment to take advantage of specialised MPI-aware hardware that is capable of significantly reducing the associated communication overhead.

### *5.2. CUDA*

For compatibility with the C-based CUDA SDK, the Fortran EISPACK source code has required source-level translation into equivalent C sources. These functions serve as a basis for the creation of three functionally equivalent GPU kernels. Performance gains emerge as data-parallel intensive loops are distributed between cooperating threads in a block and race conditions are avoided by the insertion of synchronisation primitives. These loops are identified from source-level line-profiling on the original CPU version of EISPACK, the assumption being that CPU performance is strongly indicative of potential performance bottlenecks in the GPU kernels. This is a necessary workaround as CUDA profiling tools provide relatively limited functionality. To achieve reasonable performance benefits, it is necessary to augment traditional development techniques with low-level knowledge of the underlying GPU architecture.

A number of thread blocks independently handle the solution of multiple eigensystems in parallel. In our implementation, a thread block or cooperative thread array (CTA) is mapped to an input problem set, allowing parallelism at both independent block and cooperative thread levels. Some performance optimisations applied include:

1. Asynchronous transfers to and from the host over multiple streams allow concurrent kernel execution and overlapped I/O.
2. Algorithm reorganisation for improved coalesced memory access. Matrix layout is transposed in some code sections to achieve higher memory transfer bandwidth.
3. Extraneous register variables are eliminated or reused–when appropriate–to improve GPU occupancy and facilitate latency hiding on the streaming multiprocessors.
4. Costly global memory access is limited when possible by explicit caching in shared memory.
5. The launch configuration is determined heuristically by trial and error. While, the guidelines recommend that thread blocks sizes should be multiples of a warp size to allow latency hiding for multiple warps, it is necessary to determine actual optimal block sizes by testing. The different kernels performed optimally at distinct block dimensions.

### *5.3. Verification (From Daniel)*

Testing models with high performance version.

The models with brief descriptions: (i) C40 model? (ii) C60 models, (iii) C240?. Questions. Why are they interesting? This is also an opportunity to describe the relevant entries in the output file for future users. Theoretical vs. actual S(Q,w) plots if available. What does the output of SCATTER reveal about the models? Is a molecular visualisation of any model available? May be able to plot one in 3d with atom locations and bonds if it does not exist.

## 6. Performance Analysis

To establish the performance characteristics, we conducted performance testing of an early SCATTER version on the Huygens supercomputer [27] at SARA and, subsequently, the optimised version on an IBM JS21 BladeCenter–a multi-core and multi-node platform based on the IBM Power Architecture technologies. The GPU implementation, despite being at an early stage of integration with GULP, was evaluated on a single workstation with an NVIDIA Tesla C2050 GPU. Table 1 lists relevant hardware specifications of the testing configuration.

### *6.1. Parallel Scaling*

Figure 8 presents the execution times for the 60-Carbon atom model with $4, 8$, and $16$ processes on the IBM JS21 BladeCenter. SCATTER demonstrates linear scaling across multiple cores and multiple nodes. Scaling is compared against the estimated sequential runtime obtained from a calibration run with a coarse RSO grid as predicted by Eqn. (3).

An early version of the program was deployed on the Huygens prototype supercomputer at SARA in 2010 and demonstrated near ideal scaling at up to 1024 MPI Processes [28]. Subsequent improvements in efficiency and optimisation have reduced runtime by up to two orders

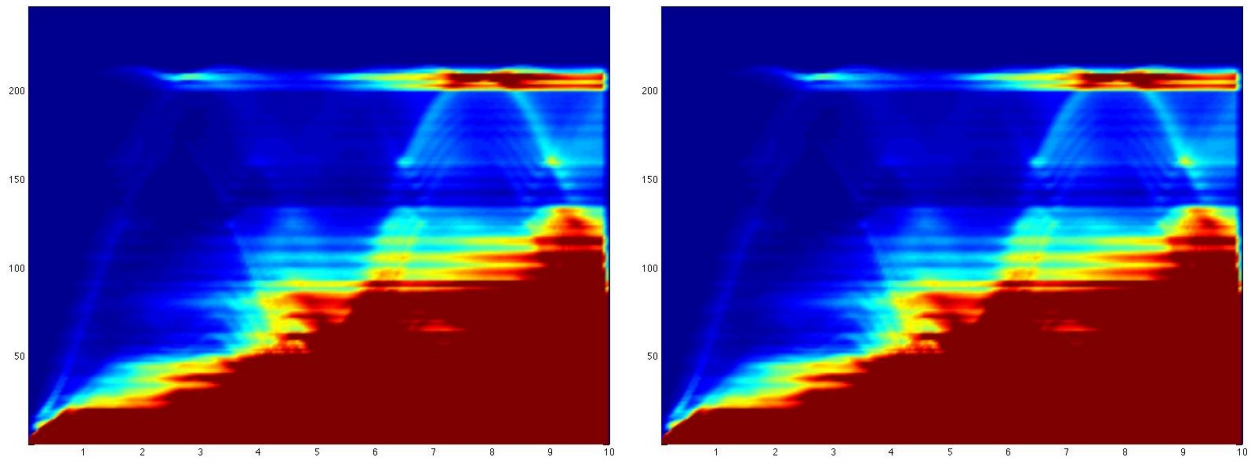



Figure 7: C60 Nanotube Model, $S(Q,\omega)$ Plot. Experimental vs. SCATTER model (PLACEHOLDER. Needs actual plots from Daniel)

| | Dell Precision XX Workstation... | IBM BladeCenter JS21 | IBM pSeries 575 (Huygens) |
|---|---|---|---|
| Nodes | 1 | 4 | 101 |
| Processing Elements per Node | 4 | 4 | 32 |
| Processor Clock | 2.0 GHz | 2.3 GHz | 4.7 GHz |
| Architecture | Intel x86-64 | IBM PowerPC 970MP | IBM Power6 |
| Total Memory | 4 GB | 16 GB | 128 GB |
| Network | FastEthernet 100Mbps | FastEthernet 100Mbps | Infiniband 160 Gbit/s |
| Operating System | Ubuntu Linux ... | Redhat Linux (kernel 26.18-8) | GNU Linux (kernel 2.6.27) |
| Compiler | GCC gfortran 4... | GCC gfortran 4.1.2 | GCC gfortran 4.3.2 |
| MPI Version | | | OpenMPI 1.3.3 |

Table 1: Test Machine Specifications

Figure 8: Scaling for the C60 model across cores

of magnitude, nevertheless the scaling characteristics remain consistent. As an illustration, the Huygens supercomputer completed the C40 model in 15 minutes using 1024 processing elements at 4.7GHz and a notably effective auto-tuning compiler. In contrast, the smaller IBM JS21 BladeCenter, with 16 2.3GHz processing elements and the GCC gfortran compiler, requires 6 hours with the current version of the program.

### *6.2. GPU Performance*

Performance evaluations have been carried out using a 64-bit Dell Precision T7500 Server with 4 Intel Xeon 2GHz CPU cores, 4GB RAM a host CPU machine and a NVIDIA Tesla C2050 GPU with a PCI express interface running Version 3.2 of the CUDA SDK on 64-bit Ubuntu 10.04 Linux. Given that the second generation NVIDIA Tesla C2050 GPU is designed specifically for scientific and numerical computing applications, it furnishes 14 streaming multiprocessors (SM), each providing 32 streaming processors (SP), and offers 448 parallel cores in total. While earlier GPUs completely lacked double precision support, the Tesla GPU provides improved double-precision floating point performance.

Using a dedicated test program, the execution times for 1000 $N$-order input matrices with EISPACK and LAPACK on a single CPU core and on the GPU are shown in Figure 9. GPU times have been collected via the platform timers and are inclusive of memory transfer overhead.

Within a critical window ($N = 512$—$2048$), the current GPU implementation is capable of yielding performance increases of between 50—100$\times$ over the reference EISPACK implementation, a result of performance gains at both thread and block levels. As the matrix order increases, the GPU memory is able to accommodate fewer matrices to provide any block-level performance advantage and resources begin to idle. Therefore, the scalability of the approach is restricted for higher values of $N$ by the hard limit that memory places on GPU occupancy despite the still-observable benefits of thread-level parallelism.

The superior LAPACK cache behaviour delivers consistently higher performance over EISPACK for larger values of $N$. While equivalent routines in both LAPACK and EISPACK are of storage order $O(n^2)$, LAPACK reuses the same input matrix memory for output and is therefore more memory efficient.

These performance results reflect expectations from an ideal implementation. However, architectural considerations create challenges in the effective integration of these routines with GULP. In a mature implementation, CPU and GPU execution may be overlapped to prevent resource idling. Nevertheless, we have tested these routines with the C240 model at coarse resolution and obtained a two order of magnitude performance gain over the CPU implementation (Figure 10).

## 7. Conclusions and Outlook

The current parallel SCATTER implementation has demonstrated ideal linear scaling and makes it possible to model INS systems of unprecedented size in affordable computational platforms. Nevertheless, it is our experience that the size and complexity of these models rapidly outgrows the computational capabilities available, as the demands for enhanced resolution in nanomaterial characterisation increases. This reflects the strong demand not only for computational tools of this kind among materials researchers, but also for the interdisciplinary link between computational and material scientists to evolve the emerging field of computational materials science.

From a computer science perspective, we expect this work to eventually shed light on more fundamental parts of parallel and distributed computing. Firstly, the evaluations of independent scattering contributions can be regarded as a divisible load, and therefore tackled using divisible load theory [29]. In fact, our ongoing research investigates allocation and scheduling heuristics to optimise system performance in complex dynamic heterogeneous computing environments with processing elements of variable capability and cost [30]. Subsequent SCATTER versions will therefore seek to examine the possibility of adaptive scheduling based on the structural and parametric characteristics of the physical models and execution environment. Secondly, we envision an eventual deployment using algorithmic skeletons [31], possibly in the form of a refined task farm [32], to achieve closer-to-optimal resource utilisation. Finally, it has become clear that achieving efficient utilisation of GPU resources is a challenging proposition in nanomaterial chracterisation, given the need to

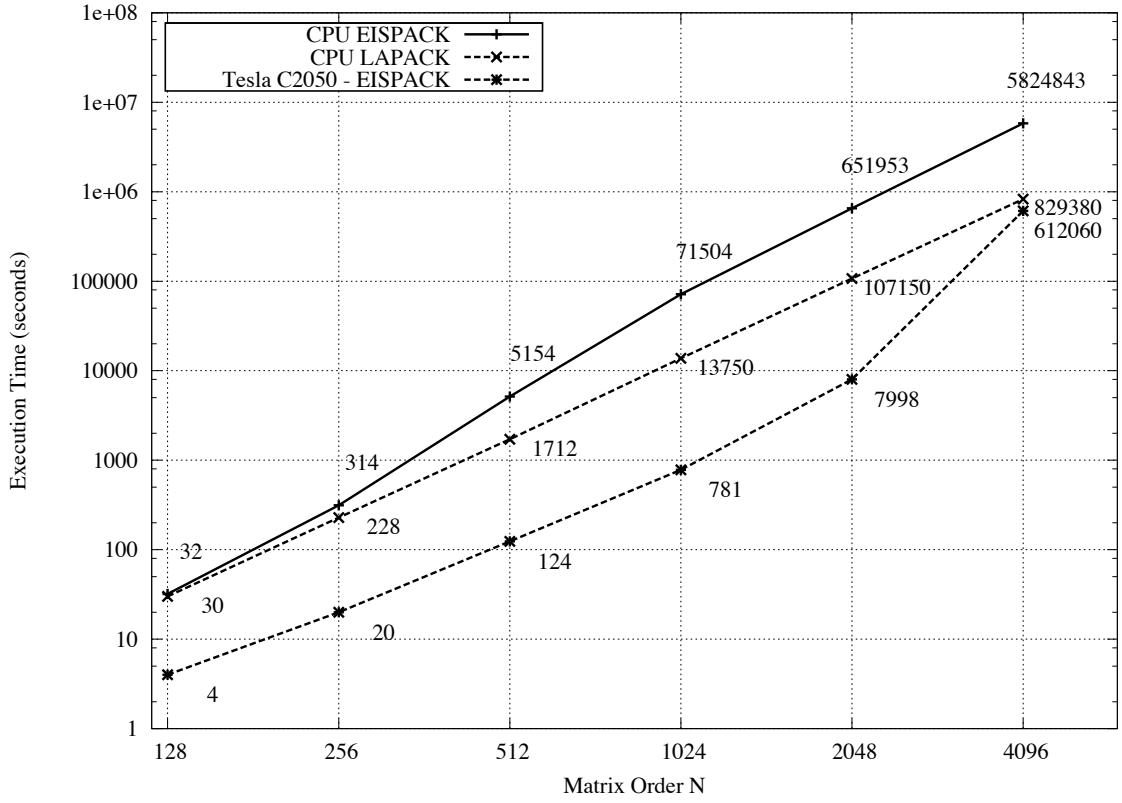



Figure 9: CPU v GPU

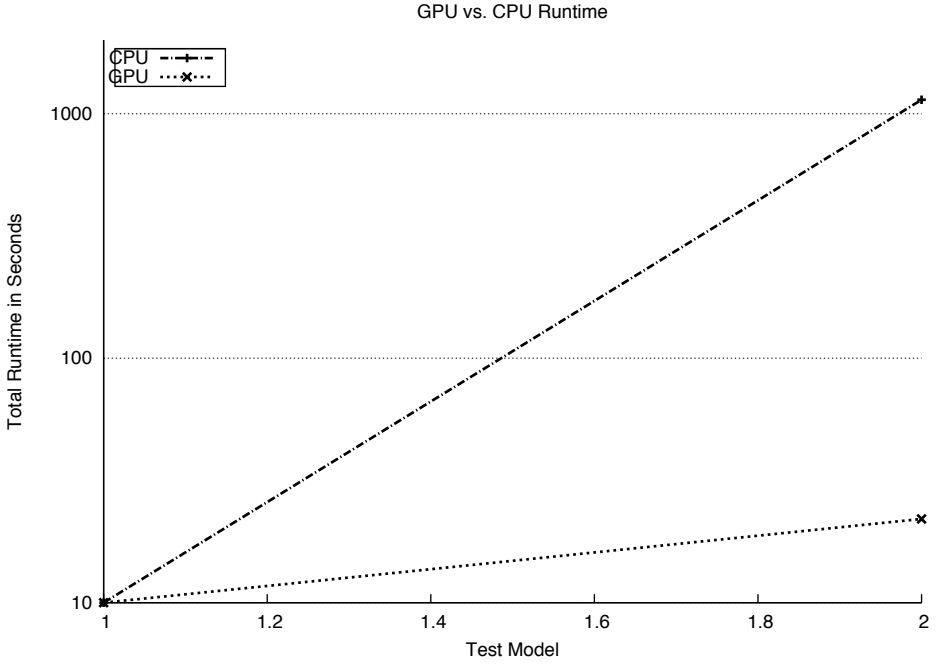



Figure 10: GPU Performance with test model

balance between CPU and GPU computational potential and these are closely tied with the parametric details of the actual model under consideration. Larger models tend to use GPUs more efficiently. Therefore, we will strive to investigate generic approaches to synthesise large parametric problems and their extension in heterogeneous CPU-GPU architectures in our future work.

## 8. Acknowledgements

The authors would like to thank the Partnership for Advanced Computing in Europe (PRACE) for their support and grant of computing time in the SARA supercomputing facilities. The PRACE project receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528. The Authors would also like to acknowledge the support of a collaboration travel grant awarded by the STFC Collaborative Computational Project 5 (CCP5), NVIDIA Corporation for the donation through the Professor Partnership programme of the GPU Tesla equipment employed in this work, and Julian Gale of Curtin University in Australia for making the GULP source code available. One of the authors (Roach) would like to acknowledge the support of EPSRC (EP/G049130) in the development of the SCATTER code.

## References

[1] W. F. van Gunsteren, A. E. Mark, Validation of molecular dynamics simulation, Journal of Chemical Physics 108 (15) (1998) 6109–6116.

[2] B.-L. Huang, M. Kaviany, Ab initio and molecular dynamics predictions for electron and phonon transport in bismuth telluride, Physical Review B 77 (12) (2008) 125209:1–19.

[3] D. Champion, J. Tomkinson, G. Kearley, a-CLIMAX: a new INS analysis tool, Applied Physics A: Materials Science & Processing 74 (2002) 1302–1304.

[4] M. Segall, P. Lindan, M. Probert, C. Pickard, P. Hasnip, S. Clark, M. Payne, First-principles simulation: ideas, illustrations and the castep code, Journal of Physics: Condensed Matter 14 (2002) 2717.

[5] J. Gale, GULP: A computer program for the symmetry-adapted simulation of solids, Journal of the Chemical Society, Faraday Transactions 93 (4) (1997) 629–637.

[6] J. Gale, A. Rohl, The general utility lattice program (GULP), Molecular Simulation 29 (5) (2003) 291–341.

[7] L. Van Hove, Correlations in space and time and born approximation scattering in systems of interacting particles, Physical Review 95 (1) (1954) 249.

[8] G. Squires, Introduction to the theory of thermal neutron scattering, Cambridge Univ. Press, 1978.

[9] D. L. Roach, J. Gale, D. Ross, Scatter: A New Inelastic Neutron Scattering Simulation Subroutine for GULP, Neutron News 18 (3) (2007) 21–23.

[10] D. L. Roach, K. Ross, J. D. Gale, The application of coherent inelastic neutron scattering to the study of polycrystalline materials, Physical Review B(Under Revision).

[11] D. L. Roach, Computational investigations of polycrystalline systems using inelastic neutron scattering techniques, Ph.D. thesis, University of Salford, Salford M5 4WT, UK (2006).

[12] V. Volkov, J. W. Demmel, Benchmarking GPUs to tune dense linear algebra, in: SC '08: ACM/IEEE Conf on Supercomputing, IEEE, Austin, 2008, pp. 1–11.

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

[14] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips, GPU computing, Proceedings of the IEEE 96 (5) (2008) 879–899.

[15] D. Kirk, W. Wen-mei, Programming massively parallel processors: A Hands-on approach, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2010.

[16] B. T. Smith, J. M. Boyle, J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler, Matrix Eigensystem Routines–EISPACK Guide, 2nd Edition, Vol. 6 of Lecture Notes in Computer Science, Springer, 1976.

[17] J. Dongarra, V. Eijkhout, Numerical linear algebra algorithms and software, Journal of Computational and Applied Mathematics 123 (1-2) (2000) 489–514.

[18] S. Tomov, R. Nath, H. Ltaief, J. Dongarra, Dense linear algebra solvers for multicore with GPU accelerators, in: IPDPS 2010 Workshops, IEEE, Atlanta, 2010, pp. 1–8.

[19] F. Vázquez, J. J. Fernández, E. M. Garzón, A new approach for sparse matrix vector product on NVIDIA GPUs, Concurrency and Computation: Practice and Experience 23 (8) (2011) 815–826.

[20] B. Garbow, EISPACK–a package of matrix eigensystem routines, Computer Physics Communications 7 (4) (1974) 179–184.

[21] A. K. Cline, J. Meyering, Converting eispack to run efficiently on a vector processor, Tech. rep., Pleasant Valley Software, Austin, Texas (1991).

[22] R. Lehoucq, D. Sorensen, C. Yang, ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods, Siam, 1998.

[23] Z. Bai, Templates for the solution of algebraic eigenvalue problems, Vol. 11, Society for Industrial Mathematics, 2000.

[24] V. Hernández, J. Román, A. Tomás, V. Vidal, A survey of software for sparse eigenvalue problems, Technical Report STR-6, Universidad Politécnica de Valencia, `www.grycap.upv.es/slepc/` (2006).

[25] J. Dongarra, I. S. Duff, D. C. Sorensen, H. A. van der Vorst, Numerical linear algebra for high-performance computers, 2nd Edition, SIAM, 1998.

[26] D. Brenner, Empirical potential for hydrocarbons for use in simulating the chemical vapor deposition of diamond films, Physical Review B 42 (15) (1990) 9458.

[27] SARA, Description of the huygens system, Web page, Dutch National High Performance Computing and e-Science Support Center, http://www.sara.nl/systems/huygens/description, (Last Accessed: 20 Jul 2011). (2011).

[28] M. Garba, H. González-Vélez, D. Roach, Parallel computational modelling of inelastic neutron scattering in multi-node and multi-core architectures, in: IEEE HPCC-10: Int Conf on High Performance Computing and Communications, IEEE, Melbourne, 2010, pp. 509–514.

[29] V. Bharadwaj, D. Ghose, T. G. Robertazzi, Divisible load theory: A new paradigm for load scheduling in distributed systems, Cluster Computing 6 (1) (2003) 7–17.

[30] H. González-Vélez, M. Cole, Adaptive statistical scheduling of divisible workloads in heterogeneous systems, Journal of Scheduling 13 (4) (2010) 427–441.

[31] H. González-Vélez, M. Leyton, A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers, Software–Practice and Experience 40 (12) (2010) 1135–1160.

[32] H. González-Vélez, M. Cole, Adaptive structured parallelism for distributed heterogeneous architectures: A methodological approach with pipelines and farms, Concurrency and Computation–Practice and Experience 22 (15) (2010) 2073–2094.