Journal of
Nuclear Engineering

Article

SCONE: A Student-Oriented Modifiable Monte Carlo Particle Transport Framework

Mikolaj Adam Kowalski *, Paul Cosgrove , Jakob Broman and Eugene Shwageraus

Engineering Department, University of Cambridge Trumpington St., Cambridge CB2 1PZ, UK;
pmc55@cam.ac.uk (P.C.); jgb45@cam.ac.uk (J.B.); es607@cam.ac.uk (E.S.)
* Correspondence: mak60@cam.ac.uk

Abstract: Over the last decade, the importance of the Monte Carlo as a neutron transport calculation method has greatly increased. This paper describes a Monte Carlo particle transport framework SCONE, which aims to provide with easy-to-learn environment for graduate students to learn about Monte Carlo methods and explore new ideas. The paper lists the steps taken to enhance new user experience of SCONE and briefly discuses how the architecture supports its goals. The current version of the code is compared against Serpent and shown to provide with sufficient accuracy to be used for teaching and proof-of-concept applications.

Keywords: Monte Carlo; neutron transport; Fortran

check for updates

Citation: Kowalski, M.A.; Cosgrove, P.; Broman, J.; Shwageraus, E. SCONE: A Student-Oriented Modifiable Monte Carlo Particle Transport Framework. *J. Nucl. Eng.* **2021**, *2*, 57–64. https://doi.org/10.3390/jne2010006

Academic Editor: Maurizio Angelone

Received: 23 September 2020
Accepted: 4 March 2021
Published: 8 March 2021

1. Introduction & Motivation

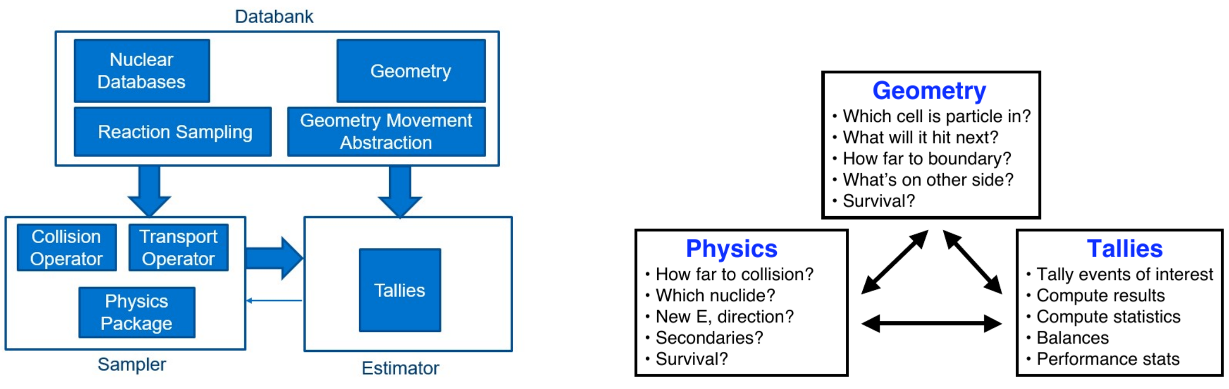
In this paper, we describe a new Monte Carlo particle transport code under development in Nuclear Energy Research Group at the University of Cambridge. It is called Stochastic Calculator Of Neutron Transport Equation (SCONE) and it is designed to be an accessible Monte Carlo particle transport environment for graduate students, which can be used in teaching and to carry out research projects that study new approaches. For this reason the priority when writing SCONE is not placed on the performance (which is the usual focus in production-level Monte Carlo codes), but on modifiability and ease of learning. It is planned to be released as an open source and used for teaching at the Engineering Department of University of Cambridge.

The goal of this paper is to describe the features of SCONE, that help it to realise its goal of simplicity and modifiability. Given that SCONE is still in its early development and very limited experience of students interacting with it has been accumulated, the discussion is largely a speculation often based on personal experience of the authors and anecdotal information gathered from code development discussion fora such as Stack Overflow. Nevertheless the discussion provides the justification for the design choices. It may also include some Fortran-specific concepts. The explanation for these can be found in Fortran Standard [1]. A short section that compares predictions of SCONE against Serpent [2] is also included.

2. Architecture

2.1. High-Level Architectural Overview

Figure 1a shows the main sections of the code. They are grouped into three distinct blocks: Databank, Sampler and Estimator. This is an alternative high-level decomposition to the Physics-Geometry-Tallies structure presented in [3] (Figure 1b). Its main difference is that it focuses on the coupling between different sections rather than on their function during particle transport. The arrows in Figure 1a are meant to represent the direction of coupling between blocks. Databank contains all global information about the problem being solved. That includes nuclear data, material compositions and geometry layout. Its main feature is that during particle transport it is accessed in read only fashion. It is not however immutable as in order to avoid needless searches of energy grid and interpolation of reaction cross-sections some caching mechanism needs to be implemented. Thus, elements in databank may have internal state, that significantly influences latency of a query. The role of the Sampler is to perform the stochastic simulation of the particle transport. It consists of collision and transport operators, which define the interaction physics and transition algorithm (e.g., surface or delta tracking). The role of the Physics Package is to define the calculation sequence. For instance, in the case of an eigenvalue calculation it contains code that generates initial source and then runs appropriate number of inactive and active cycles. Sampler by itself is not capable of producing any useful results. It simply sends the events reports (via subroutine calls) to the estimator, which processes them into desired quantities. Ideally Estimator would be independent of the Sampler, but in practice most calculation sequences require access to some results. For example, many source convergence acceleration or adaptive variance reduction schemes need access to the accumulated results that are parameters of the lower-order representation of the system. However, this Estimator-Sampler coupling is limited to only few selected tallies and is much weaker then the Sampler-Estimator interaction via event reports. That is why it was indicated with a thin arrow on Figure 1a.



(**a**) (**b**)

Figure 1. (**a**) An overview of High-Level structure of SCONE. Shows the main code sections and their decomposition into Databank, Sampler and Estimator block. (**b**) Physics-Geometry-Tallies decomposition from [3].

2.2. *Support of Novice User Experience*

It is expected that SCONE will be mainly used by UK's postgraduate students in Nuclear Engineering. Thus, it is reasonable to assume that upon a first introduction to the framework, a student will be already familiar with basic programming concepts and aware but not fully comfortable with more advanced ideas such as inheritance or polymorphism. Therefore, the initial contact with the framework will be challenging to students as they will be required not only to understand the specifics of SCONE, but also become comfortable with object-orientation and relevant design patterns [4]. Fortunately there is a large number of steps that can be taken to improve the initial user experience. They will be outlined in the following paragraphs.

The simplest enhancement of new user experience is the organisation of the source code directory into a folder structure, such that functionally related source files are kept in the same folder. In SCONE the folder structure tries to convey the architecture. Starting from the root directory, each major component of the code has an associated folder, e.g., "NuclearData" or "Geometry". Then each such folder contains source files that specify the abstract classes and data structures, that serve as an interface for this section of the code. A more specific interfaces (abstract classes that inherit from the main interface class) as well as implementations of the interfaces are places in sub-folders. Thus, a folder tree follows the inheritance hierarchy. The main benefit of this approach is that it makes navigating source code easy. A new user, instead of being presented with hundreds of files sorted alphabetically, can see the high-level decomposition of the code into different abstractions from the beginning.

Another decision to improve the readability of SCONE source code was to keep the size of the source files small. At the time of writing only 4 source files out of 279 have more then 1000 lines (which includes comments). The mean size of the file is 221.4 lines. Each source file contains a single Fortran module or program. To avoid confusion, each source file has the same name as the module it contains (e.g., module "RNG_class" is in file "RNG_class.f90"). To improve clarity, no functions or types are used implicitly. Every object or procedure needs to be explicitly imported using the Fortran "use..., only: ..." construct. This rule makes certain that it is straightforward to identify origin of every component that is being used in a module. The exception from this rule is allowed when importing modules that contain only large number of constants (e.g., ENDF MT numbers). The content of every module is indicated by a suffix in its name: "_class" for class, "_inter" for abstract class, "_func" for library of procedures and _mod": for module acting as an object (Singleton Design Pattern [4]).

The fact that the expected new user of SCONE will have a limited programming experience makes the selection of Fortran as a programming language beneficial. Because the student, will have to learn both the unfamiliar Monte Carlo techniques and the programming language at the same time, it is important that all the relevant concepts are not obfuscated by syntax. The Fortran use of verbose attributes such as "pointer" or "allocatable" communicate the meaning clearly even to people who are not well-acquainted with Fortran syntax. It is also important that the MATLAB language shares many features with Fortran, such as array syntax and column major order for indexing. This makes transition to Fortran easier for engineering students who often have extensive experience with MATLAB.

Consistent and exhaustive documentation of classes and procedures allows the user to quickly determine the purpose and functionality of any given component. In SCONE, documentation is based on the Google docstring style for Python code. A comment is placed before the declaration of a derived-type or a procedure. In all cases it contains the name of the component and a short paragraph that details it purpose. In a case of a procedure (including class methods), the comment contains three sections. "Args" section provides the description and intent (in, out or inout) of arguments as well as any preconditions they must meet (e.g., an array of numbers may need to be sorted). "Errors" section lists the behaviour of the procedure for invalid input or edge-cases. A function also contains "Result" section that describes the output. Documentation comment for classes contains "Public Members" and "Private Members" sections that provide short description of each class member. In addition "Interface" section is included which lists all available public methods with short description of their purpose. When documenting class, inheritance poses a threat to the consistency of documentation if methods in each sub-class have their own full documentation comment. Because, the same information about the specification of the method would be replicated in multiple places, any change would need to be propagated to all instances. This would make it likely that due to mistakes and omissions, with time the descriptions of the same method would diverge. To prevent it, if the method is overriding a super-class method only reduced documentation comment is provided with basic information about the purpose of the method, name of a module that contains full documentation comment and perhaps some sub-class specific information. Full comment is provided only for the top-level declaration of the procedure.

The use of object-oriented architecture should help new users to learn the code. SCONE is based on a plug-in architecture with every major code component such as geometry or nuclear data being associated with an abstract class, which deferred (virtual) methods provide access to all required functionality. The code components are also associated with abstractions, which are simplified mental models of the sections of the code. Every abstraction specifies how the section should behave under interaction through procedure calls. Within this arrangement it is easy to provide multiple implementations of the same functionality using the idea of polymorphism. All implementations are associated with a sub-class of the abstract class and as long as all of them conform to the abstraction they are completely interchangeable from the user perspective. In practical terms it means that it is sufficient for a student to learn only the abstraction and the interface of the abstract class in order to be able to use all implementations. Furthermore it is not necessary to know anything about the details of the implementation. This significantly reduces mental burden when learning the code. The plug-in architecture also makes the code modification easier if the change can be made without changing the abstractions, because there is no need to consider the interaction with all the other sections of the code. It is sufficient that the modified version also implements the abstraction. However, plug-ins can also have a detrimental effect and make the modification much more difficult if it is necessary to make a change in an abstraction. Then it is necessary to ensure that all existing implementation will conform to the new specification of the abstraction, which can require a significant amount of work if the code is mature and multiple implementations are already created. For that reason the design of the SCONE abstraction is crucial for the code usefulness as a research tool. Furthermore, the reliance on abstract classes have a significant performance penalty due to dynamic dispatch on procedure calls.

2.3. Tallies Module

In order to illustrate the principles followed in the design of SCONE, the architecture of tallies will be discussed in detail. Their main purpose in the Monte Carlo particle transport calculation is to calculate statistical estimates of the useful quantities from the samples of various events obtained from the Sampler. This provides a natural interface that can reduce dependency of the tallies on the rest of the code. The tally subroutines need only to interact with a number of defined reports about certain events and require no knowledge about how the reports were generated. In the extreme case it is even possible that the generation of the reports and their processing may be performed by completely different programs as was the case for 05R code [5]. However, in practice, some degree of interaction between transport routine and tallies is required.

Figure 2 shows a sketch of class diagram of SCONE tallies section. Ordinary arrows represent association, white diamond aggregation, black diamond composition and white arrow generalisation (inheritance). The main goal of this arrangement was to contain all the code that governs how event reports are converted into useful results inside a single class (and thus single source file) of tallyClerk. The role of the interface for all tally functionality is served by the tallyAdmin. Sampler send reports about the events by calling the methods of the tallyAdmin, which then routes the reports to all tallyClerks that require them. Furthermore, it is the responsibility of a tallyAdmin to allocate and release memory for all clerks. Each tally admin contains a single instance of scoreMemory, which is a class responsible for management of space for storing results and performing normalisation and batching. The reason why the scoring logic contained in tallyClerks was decoupled from storage space was to simplify the implementation of operations on all results such as archiving into a file. Each tallyClerk is assigned with a contiguous section of scoreMemory for its results by a tallyAdmin that provides the clerk with an address. Following types of reports are available to a clerk to generate results:

- reportCycleStart: Event that indicates the beginning of a cycle. A reference to a particle population with all normalisations applied that is to be transported over the course of the cycle is provided as an argument.
- reportInColl: Event associated with a particle entering collision.
- reportOutColl: Event associated with the particle exiting collision. Argument list contains cosine of scattering angle in LAB frame and MT number of the reaction the particle has underwent.
- reportPath: Report that provides a path a particle has moved within confines of a single material or cell with unique ID. Argument list contains the length of the path.

- reportTrans: Report that indicates a transfer of a particle between collisions along a straight line. Collision with a boundary condition is interpreted as a collision.
- reportHist: Report that indicates the end of a particle history. It includes an integer flag that provides information about the fate of the particle (leaked, absorbed or lost).
- reportCycleEnd: Event that indicates the end of a particle transport cycle. A reference to a particle population that is to be transported in the next cycle without any population normalisation is provided. It also includes the value used to adjust secondary particles generation rate.

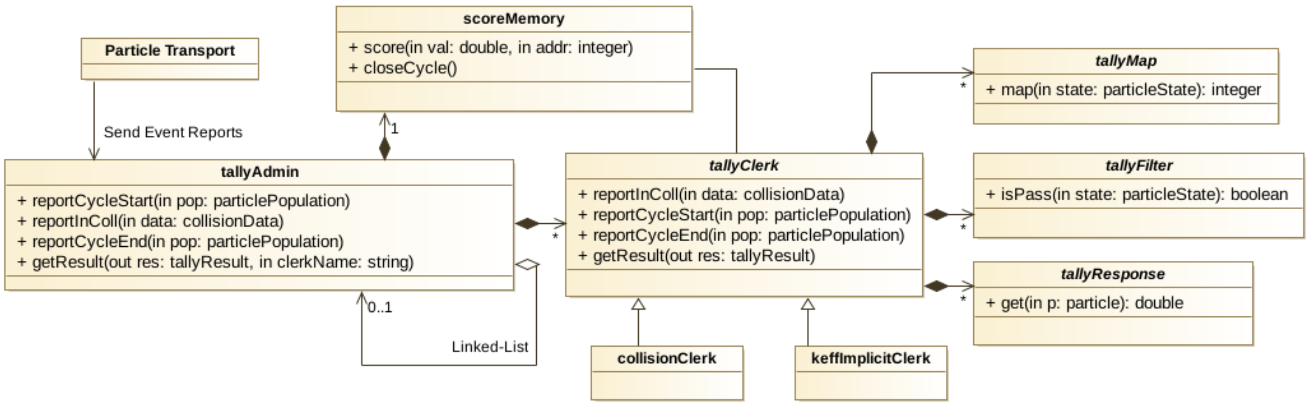

scoreMemory
+ score(in val: double, in addr: integer)
+ closeCycle()
Particle Transport
Send Event Reports
1
tallyAdmin
+ reportCycleStart(in pop: particlePopulation)
+ reportInColl(in data: collisionData)
+ reportCycleEnd(in pop: particlePopulation)
+ getResult(out res: tallyResult, in clerkName: string)
0..1
Linked-List
*
tallyClerk
+ reportInColl(in data: collisionData)
+ reportCycleStart(in pop: particlePopulation)
+ reportCycleEnd(in pop: particlePopulation)
+ getResult(out res: tallyResult)
collisionClerk
keffImplicitClerk
tallyMap
+ map(in state: particleState): integer
tallyFilter
+ isPass(in state: particleState): boolean
tallyResponse
+ get(in p: particle): double

Figure 2. Sketch of class diagram of SCONE tally module. Style of arrows follows the Unified Modelling Language (UML). Only a selection of the available class methods is shown to improve clarity.

Each report contains information about the particle together with its global position in phase-space at key moments (e.g., start of history). It needs to be noted that the tally architecture was created with batch statistics in mind. ScoreMemory is responsible for the control over the batching process and it also contains the number of cycles in each batch. Clerks that require non-standard normalisation (e.g., Fission Matrix tally) can enquire whether the cycle currently ending requires accumulation of results using the "lastCycle" method of scoreMemory. The coupling with the Sampler is achieved by using a family of tallyResult classes. In "getResult" method each clerk may allocate provided allocatable argument (an unique pointer) to a derived type created to store the clerk's data. For example, tallyResult for k-eff clerks store only two numbers related to expectancy and standard deviation, while tallyResult for a Fission Matrix Clerk contains a matrix. By default, clerks that do not implement coupling allocate tallyResult to a special null type called noResult. This approach has a drawback of a significant overhead that may be associated with the allocation of memory whenever result is requested.

In order to avoid needless reimplementation of common functionality, three class families are available to every Clerk. tallyMap is responsible for mapping a particle global position in phase-space to a bin. It contain method "map" that returns an index given a particle state. If the particle state falls outside a range of a map, index 0 is returned. Even though the results are mapped to a single index, maps can have multiple dimensions. Information about the shape can be obtained from a tallyMap with "binArrayShape()" method. Maps are represented in column-major order following Fortran convention. The existence of tallyFilter, which given a particle state accepts or rejects the event, may be considered redundant given that its functionality can be implemented using a tallyMap composed of a single bin. However it can be used to restrict range of the accepted events without changing the rank of a result array in output file and any scripts written for results post-processing. For this reason it may prove useful during classroom demonstrations or when investigating a new phenomena.

3. Verification

In order to verify the implementation of SCONE algorithms, a comparison with Serpent [2] is shown for three benchmark problems. Since SCONE does not currently support the unresolved resonances probabilities tables and bound thermal scattering, these options were switched off in the reference Serpent calculation. The fast spectrum test cases are taken from the NEA's ICSBEP Critically Benchmark suite. Serpent input files were obtained from Serpent Wiki [6]. The first case is the Jezebel sphere (PU-MET-FAST-001), which is a simple metallic plutonium sphere. The Popsy case (PU-MET-FAST-006) consists of a plutonium sphere surrounded by a natural metallic uranium reflector. For the thermal spectrum, a 2D calculation of 17×17 MOX assembly with reflective boundary was performed. The assembly contained fuel with three different levels of enrichment (4.3 wt%, 7.0 wt% and 8.3 wt%). The exact geometry and material specifications for the assembly were taken from [7]. All calculation were performed on a single core with normalisation to the fission rate of 100. The same ACE-formatted JEFF 3.1.1 nuclear data library was used by both SCONE and Serpent. Every case was run with a population of 200,000 neutrons and 500 active cycles. MOX and Popsy cases used 100 inactive cycles, while Jezebel used only 20. For all cases Serpent used optimisation setting of 4, which is recommended for small problems as it performs the most aggressive optimisation of run time at the expense of memory consumption [6].

Table 1 compares the criticality predicted by Serpent and SCONE for the benchmark cases. It is clear that there is good agreement between the predictions of two codes. The only significant deviation by 69 pcm can be observed for the Popsy test case. The reason for it has not yet been identified. Figures 3 and 4 reveals that the spectrum predicted by SCONE is within standard deviation from the Serpent result for for both fast and thermal spectrum cases. For the MOX case, SCONE executes about 32% slower than Serpent. The reason is that the current version of SCONE lacks a mechanism to accelerate energy grid searches such as unified energy grid or energy grid hashing. Thus, the runtime scales approximately linearly with the number of nuclides in the problem. The significantly faster performance of SCONE for the fast spectrum cases is unexpected. The use of different optimisation setting in Serpent did not succeed in bringing the Serpent performance closer to SCONE. By default Serpent calculates adjoint weighted kinetic parameters using iterated fission probability (IFP) method. It likely that the extra overhead of IFP tallies is responsible for the difference in performance in fast problems, where neutron histories are short.

Table 1. Comparison of runtime and predicted criticality for the benchmark cases between SCONE and Serpent.

Case	SCONE k-eff	SCONE Runtime [s]	Serpent k-eff	Serpent Runtime [s]
Jezebel	1.00015 ± 9 pcm	257	0.99999 ± 10 pcm	765
Popsy	1.00307 ± 11 pcm	3250	1.00376 ± 11 pcm	5036
MOX	1.19871 ± 6 pcm	9367	1.19872 ± 6 pcm	7091

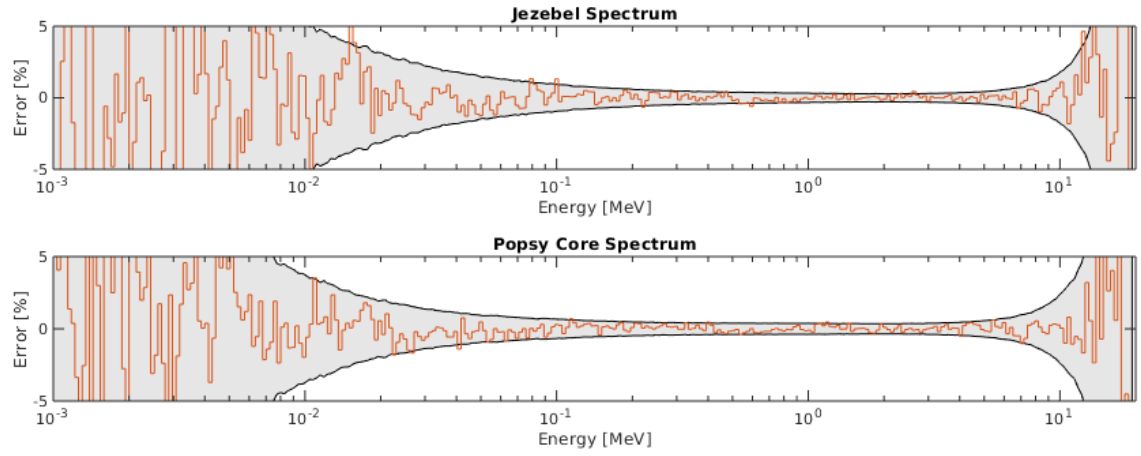


Figure 3. Error in flux spectrum for the fast benchmark cases. Grey region represents the extent of two standard deviations.

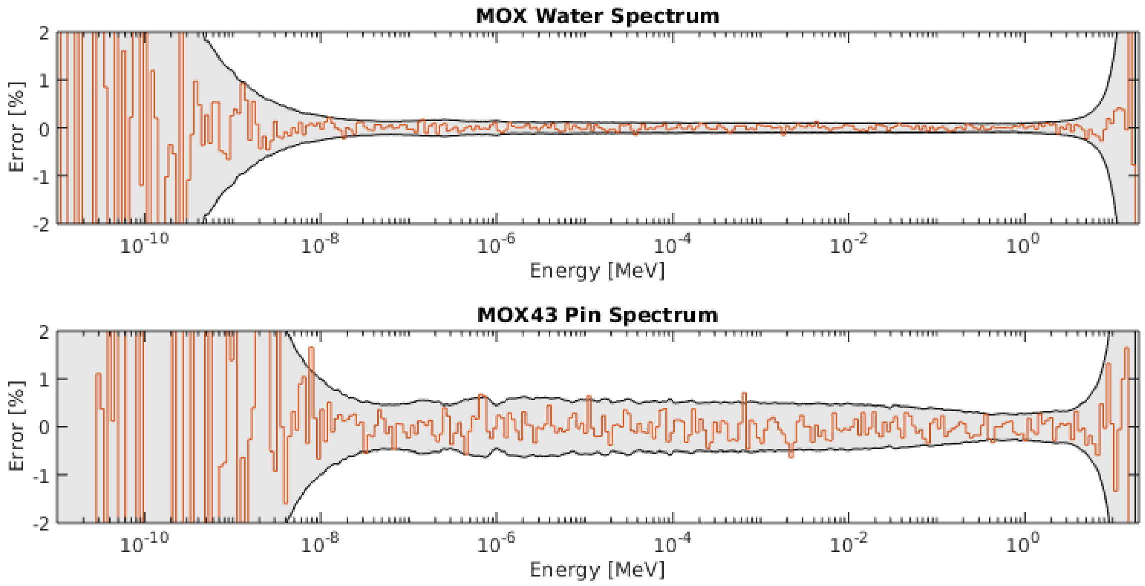


Figure 4. Error in flux spectrum for the MOX thermal benchmark case. Grey region represents the extent of two standard deviations.

4. Conclusions & Further Work

When writing SCONE significant effort has been dedicated to support the new user of the framework. Steps such as grouping source files in sub-folder stricture that follows inheritance hierarchy, may seem trivial but applied consistently can have significant impact on the readability of the code. Furthermore, the object oriented architecture can reduce mental burden when making changes to the code by reliance on the abstractions for the major code components. The discussion of the tallies architecture demonstrates that it is possible to concentrate relevant code in a number of classes (tallyClerks), while hiding the confusing but necessary operations in others (tallyAdmin and scoreMemory). By comparison with Serpent it was shown that SCONE in its current version should be accurate enough for the use in teaching and proof-of-concept studies of new Monte Carlo approaches.

Current work is focused on parallelising the code using OpenMP, and preparing for an open source release by removing gaps in the unit test coverage and documentation. SCONE is likely to be released under MIT licence.

Author Contributions: Conceptualisation, M.A.K.; Methodology, M.A.K.; Software, M.A.K., P.C. and J.B.; Writing-Original Draft Preparation, M.A.K.; Writing-Review Editing P.C. and E.S.; Supervision, E.S.; Funding Acquisition, E.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the UK Engineering and Physical Sciences Research Council and David and Susan Hibbit Scholarship from Jesus Collage.

Data Availability Statement: SCONE is to available under MIT Licence at: https://bitbucket.org/Mikolaj_Adam_Kowalski/scone accessed on 2 March 2021.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. *Information Technology—Programming Languages—Fortran—Part 1: Base Language*; Technical Report ISO/IEC 1539-1:2010; International Organisation for Standardization: Geneva, Switzerland, 2010.
2. Leppänen, J.; Pusa, M.; Viitanen, T.; Valtavirta, V.; Kaltiaisenaho, T. The Serpent Monte Carlo code: Status, development and applications in 2013. *Ann. Nucl. Energy* **2015**, *82*, 142–150. [CrossRef]
3. Brown, F.B. *Fundamentals of Monte Carlo Particle Transport*; Technical Report LA-UR-05-4983; Los Alamos National Labratory: Los Alamos, NM, USA, 2020.
4. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1995.
5. Lux, I.; Koblinger, L. *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*; CRC Press: Boca Raton, FL, USA, 1991.
6. Serpent Wiki. 2020. Available online: http://serpent.vtt.fi/mediawiki/index.php/Main_Page (accessed on 25 December 2019).
7. *Benchmark Calculations of Power Distribution within Fuel Assemblies, Phase II: Comparison of Data Reduction and Power Reconstruction Methods in Production Codes*; Technical Report NEA/NSC/DOC(2000)3; Nuclear Energy Agency: Paris, France, 2000.