Implementing the combing method in the dynamic Monte Carlo

Fedde Kuilman

PNR_131_2012_008

Faculteit Technische Natuurwetenschappen

Implementing the combing method in the dynamic Monte Carlo.

Bachelor End Thesis
PNR_131_2012_008

Naam:	Fedde Kuilman		
Studentnummer:	1384643		
Opleiding:	Technische Natuurkunde		
Begeleider:	B. L. Sjenitzer	Sectie:	PNR
2de beoordelaar:	J.E. Hoogenboom		
Start datum:	05-11-2011	Einddatum:	25-07-2012
Is vertrouwelijkheid van toepassing?	Nee		

Abstract

The aim of this report is to increase the stability of a dynamic Monte Carlo simulation of a nuclear reactor. This is done by implementing a population control method called the combing method. The combing method forces the neutron population to remain constant, while redistributing the weight equally among the particles. This method was implemented on a sample problem and compared to the Russian roulette method already in use. The difference in results of the dynamic Monte Carlo code using the combing method and the same dynamic Monte Carlo using Russian roulette lie within the standard deviation, indicating the proper functioning of the combing method.

The combing method was unsuccessful in increasing the stability or Figure of Merit of the dynamic Monte Carlo when compared to Russian roulette. However, it does offer some extra flexibility and shows some promise in it's ability to achieve better performance.

Contents

Chapter 1

Introduction

For nuclear safety reasons it is very important to calculate the power behaviour of a nuclear reactor when a transient is introduced. This transient behaviour happens for instance while moving a control rod or during an accident. Usually these calculations are done by deterministic or hybrid methods; although fast, these methods have the drawback of necessary approximations. Approximations made, make it difficult to precisely estimate the uncertainty calculation result.

Sjenitzer and Hoogenboom of Delft University of Technology introduced a method of calculating this power behaviour using Monte Carlo simulation [1]. Although computationally expensive, this method requires no assumptions. Hence, the uncertainty only relies on the statistical nature of Monte Carlo, this allows for a precise estimation of the uncertainty in power behaviour.

In this report the feasibility of using the **combing method**, as introduced by Booth [2], for increasing stability and efficiency of a dynamic Monte Carlo reactor simulation has been investigated.

1.1 Layout of report

This report the following structure. Chapter 2 discusses the general (dynamic) behaviour of reactors. It explains how and why prompt neutrons and precursors are important for this behaviour and it introduces the mathematics needed to calculate the power behaviour of reactors. Chapter 3 discusses the Monte Carlo technique. It explains how Monte Carlo simulations can be made to run more efficient. It also introduces the mathematics needed to effectively simulate the particles introduced in the previous chapter.

Chapter 4 explains the combing method and its advantages. It explains several different ways the method can be applied and what their respective advantages and disadvantages are. Chapter 5 introduces the sample problem on which the calculations are done, it introduces the findings and results of this research. Chapter 6 discusses the findings in greater detail and draws conclusions regarding the use of the combing technique in dynamic Monte Carlo simulations for reactor simulation.

Chapter 2

The dynamic behaviour of reactors

2.1 Nuclear reactions

There are essentially two types of nuclear reactions of importance in the study of nuclear reactors:

1. **Spontaneous disintegrations of nuclei**
2. **Reactions resulting from the collision between nuclei and/or nuclear particles**

An example of the first would be radioactive decay of unstable fission products. The rate at which naturally occurring nuclei decay is very slow. A neutron colliding with a ^{235}U nucleus is an example of the second type:

$$\text{Neutron} + {}^{235}U \rightarrow \text{fission products} + \text{more neutrons} + \text{energy} \tag{2.1}$$

The products of such a reaction emerge with very large kinetic energy, which is turned into heat as they are slowed down by neighbouring atoms. This heat is used by nuclear reactors to create steam which powers turbines and thus create electricity. The neutrons released by this fission reaction may go on to induce more fission reactions and hence start a *chain* of fission reactions [3].

2.1.1 Nuclear cross section

The probability of a neutron-nuclear reaction for the nucleus is characterized by the nuclear cross section, this is given by:

$$\sigma = \frac{(R/N_a)}{I} \tag{2.2}$$

Where R is the rate of neutron-nuclear reactions in $\frac{\#}{cm^2 s}$, N_a is the number of target atoms per unit area in $\frac{\#}{cm^2}$ and I is the incident neutron beam intensity in $\frac{\#}{cm^2 s}$ A neutron can on collision scatter or get absorbed and when absorbed can either get captured or cause a fission, all these events have their own cross section. The total probability of interaction is σ_t.

For simulations the macroscopic cross section is a more useful quantity; this is the microscopic cross section multiplied by the atomic number density:

$$\Sigma = N\sigma \tag{2.3}$$

Here the term "macroscopic" arises from the recognition that Σ characterizes the probability of neutron interaction in a macroscopic chunk of material (the target); whereas the microscopic cross section characterizes the probability of interaction with only a single nucleus. It should be noted that Σ is not actually a cross section, since its units are inverse in length. A better interpretation would be the probability that the neutron will undergo a reaction with a nucleus in the sample per unit path length travelled [4].

The total macroscopic cross section Σ_t, indicating the chance of interaction, can simply be divided into other macroscopic cross sections, indicating the probability the neutron will undergo a specific type of interaction with a nucleus in the sample per unit path length. Thus we can define other macroscopic cross sections such as Σ_s, the scattering cross section and Σ_f, the fission cross section. These cross sections indicate the probability the neutron will respectively undergo scattering and fission in the sample per unit path length travelled.

2.1.2 Criticality

For nuclear reactions a multiplication factor can be defined; this is the average number of neutrons of one fission event that cause another, or

$$k \equiv \frac{\#\text{neutrons in generation n+1}}{\#\text{neutrons in generation n}} \tag{2.4}$$

A somewhat more practical definition of the multiplication factor k can be given in terms of a neutron balance relation:

$$k \equiv \frac{\text{Rate of neutron production in reactor}}{\text{Rate of neutron loss (absorption plus capture) in reactor}} \equiv \frac{P(t)}{L(t)} \tag{2.5}$$

Here the production and loss rates may change with time (e.g., due to fuel consumption).

Notice that if $k = 1$ the reaction is critical, the number of neutrons in any two consecutive generations will be the same, hence the reaction will be time-independent, the is the stationary or stable situation. If $k > 1$ the population will now grow over each time step and hence be unstable, this is called a supercritical reaction. If $k < 1$ the reaction is subcritical for every generation has less neutrons then the previous one and the reaction will die out.

2.1.3 Prompt and delayed neutrons

In this work two types of neutrons are distinguished: delayed neutrons and prompt neutrons. It should be noted that although these two types of neutrons have a different origin, they are however otherwise completely identical.

Prompt neutrons are released within 10^{-13} s from a fission event. A delayed neutron is emitted after the beta-decay of an unstable nuclei. These nuclei are called precursors. Precursors are formed by fission reactions and decay stochastic. The decay probability of a precursor can be described by an exponential distribution:

$$p(t) = \lambda e^{-\lambda t} \tag{2.6}$$

The average lifetime of a precursor is simply the mean of this distribution; this is $\frac{1}{\lambda}$. The precursors are often divided into different groups, categorised by average lifetime.

A small fraction of all fission reactions produce a precursor; the net production of precursors is [5]:

$$\frac{\partial C_i}{\partial t} = \beta_i \nu \Sigma_f \phi(\mathbf{r}, t) - \lambda_i C_i(\mathbf{r}, t) \tag{2.7}$$

Where C_i is the concentration of precursors of group i, β_i is the fraction of total fission reactions which produce a precursor in group i, ν is the average number of neutrons released per fission event and ϕ is the neutron flux. All β_i can be summed to form the total delayed fraction β likewise all C_i can be summed to form the total precursor concentration C.

2.1.4 Point kinetics

Suppose $N(t)$ is the total neutron population at time t, a neutron lifetime can now be defined as:

$$l \equiv \frac{N(t)}{L(t)} \tag{2.8}$$

This lifetime is useful for studying the dynamic behaviour of reactors. Also a rate of change of $N(t)$ can be given by:

$$\frac{dN}{dt} = P(t) - L(t) \tag{2.9}$$

substituting with Eq. (2.5) gives:

$$\frac{dN}{dt} = (k-1)L(t) \tag{2.10}$$

and using Eq. (2.8) this becomes:

$$\frac{dN}{dt} = \frac{(k-1)}{l} N(t) \tag{2.11}$$

Under the assumption that both k and l are time-independent (which does not hold in general), this becomes an ordinary differential equation which becomes:

$$N(t) = N_0 \mathrm{exp}\left[\frac{(k-1)}{l} t\right] \tag{2.12}$$

where N_0 is the number of neutrons at $t = 0$. This simple expression allows us to evaluate the dynamic behaviour of the neutron population to some extent [3].

This description is too simple for the dynamic case. To described this other important quantities are needed and the delayed neutrons need to be inserted more explicitly. One of which, the **reactivity** is given by:

$$\rho(t) \equiv \frac{k(t) - 1}{k(t)} \tag{2.13}$$

This essentially measures the deviation of core multiplication from its critical value $k = 1$. Note that k and ρ are explicitly indicated as possible functions of time. The **mean generation time** is given by:

$$\Lambda \equiv \frac{l}{k} \equiv \frac{\text{mean generation time between birth of neutron}}{\text{and subsequent absorption inducing fission}} \tag{2.14}$$

These make it possible to write the system of equations describing the neutron flux in a reactor including delayed neutrons in their most conventional form [4]:

$$\frac{dN}{dt} = \left[\frac{p(t) - \beta}{\Lambda}\right] N(t) + \sum_i \lambda_i C_i(t) \tag{2.15}$$

$$\frac{dC_i}{dt} = \frac{\beta_i}{\Lambda} N(t) - \sum_i \lambda_i C_i(t) \tag{2.16}$$

2.2 Delayed lifetime

Although the total delayed fraction β is very small ($\approx 0.70\%$ for thermal reactors), it is very important for reactor control. Nuclear reactors are operated in prompt sub-critical, delayed critical condition: the prompt neutrons alone cannot sustain a chain reaction, the delayed neutrons are required to keep the reaction going. Hence the average lifetime of a neutron used in the chain reaction is lowered by the existence of delayed neutrons.

For the lifetime of a neutron, from birth to collision is in the the order of 10^{-4}s whereas the longest lifetime of a precursor is in the order of 10^{2}s hence the difference in lifetime can be many orders of magnitude.

The average lifetime of a neutron is typically in the order of magnitude of 0.1s, considerably longer than that the lifetime of a prompt neutron.

Suppose the reaction would be sustained by prompt neutrons alone. Using the simple point kinetics model introduced in Section 2.1.4 a neutron lifetime l of 10^{-4}s and a supercritical reaction with k=1.001, it is possible to calculate the time it takes for the population to increase by a factor of 10. Rewriting Eq. (2.12) to

$$t = \frac{l}{(k-1)} \log \frac{N(t)}{N_0} \tag{2.17}$$

and inserting $N(t) = 10N_0$ gives $t \approx 0.23s$. Hence for a reactor operating in prompt critical condition with a multiplication factor k slightly over 1 it only takes $0.23s$ to increase the number of neutrons by a factor of 10. The importance of which can be seen using Eq. (2.12), which shows that the number of neutrons increase by a factor $e^{10} \approx 22.026$ in only one second.

Now examine the same reaction only this time with delayed neutrons needed to sustain the reaction. Now the neutron lifetime would become:

$$l = (1-\beta)l_p + \beta(t_d + l_p) \approx \beta t_d \tag{2.18}$$

where l_p is the neutron lifetime of prompt neutrons and t_d is the average lifetime of delayed neutrons of a certain nucleus. The approximation made is very close for $t_d \gg l_p$. Using ^{235}U for which $\beta = 0.70\%$ and $t_d = 13s$ this gives a neutron lifetime l of $0.091s$. Inserting this in Eq. (2.12) yields a time of almost

$210s$, almost a factor 1000 larger then prompt critical reaction. The number of neutrons now increase by a factor of $e^{\frac{.001}{.091}} \approx 1.011$.

Hence, delayed neutrons substantially slows down the behaviour of a reactor. If only prompt neutrons would exist, the rapid power changes would make reactor control using mechanical methods near impossible. Hence, the delayed neutrons allow for effective control of nuclear reactions making operable nuclear reactors possible [3].

2.3 Difference in time scales

At the end of a prompt neutron's lifetime it can create a new prompt neutron; the probability this happens is given by:

$$P_f = k_{\text{eff}}(1 - \beta) \tag{2.19}$$

If a new prompt neutron is created it also has a chance of P_f to create a new prompt neutron at the end of it's lifetime. Now the average chain length can be found to be:

$$\frac{1}{1 - P_f} \tag{2.20}$$

This value lies typically around 150. Hence, the average lifetime of a prompt neutron chain is in the order of 10^{-2}s, several orders of magnitude smaller than the lifetime of a precursor. The variance in length of these chains is very important, for it is one of the main contributions to the variance in this work.

In a critical system a prompt neutron chain will on average create one precursor. After the chain has ended there is no more power production from this event until the precursor decays and starts a new prompt neutron chain. This is shown in Fig. 2.1. In the simulation this causes long periods without precursor decay and hence no prompt neutrons. This is addressed in Section 3.2.

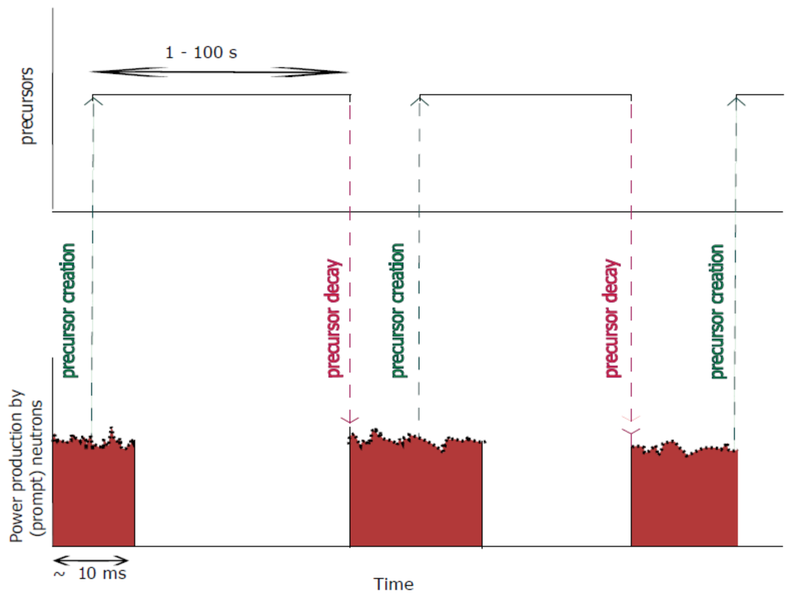


Figure 2.1: A somewhat crude representation of the difference in lifetime between precursors and prompt neutrons chain. Note that the time scale is not exact and that prompt neutron chains tend to diverge.[1]

Chapter 3

The Monte Carlo method

The Monte Carlo method is a stochastic simulation method able to simulate a wide variety of problems, spanning many research areas including finance, reactor physics, oil exploration and complex mathematical problems. The method works by generating a large number of (pseudo-)random numbers to calculate the behaviour of the desired set-up for a large number of possible states. Using statistics, specifically the Central Limit theorem [6], the expectation value and standard deviation can then be calculated for the used set-up.

Monte Carlo has the advantage that for nuclear application, no approximations need to be made. The uncertainty is thus only dependent on the statistics and can hence be precisely estimated. The drawback of using this method is the high computational cost due to the large number of simulations being done to achieve a low variance.

3.1 The general principles

The Monte Carlo method is a stochastic method used to solve deterministic problems. In neutron transport problems such as discussed in this work, it is used to solve the Boltzmann Eq. (3.1), which determines the dynamic behaviour of neutrons.

$$\Omega \cdot \Delta\psi(\mathbf{x},\mathbf{\Omega}) + \sigma_t\psi(\mathbf{x},\mathbf{\Omega}) = \frac{\Sigma_s}{4\pi}\int_{4\pi}\psi(\mathbf{x},\mathbf{\Omega}')d\mathbf{\Omega}' + \frac{Q(x)}{4\pi} \tag{3.1}$$

Monte Carlo solves this equation by tracking individual particles and simulating them throughout their existence. This is analogous to particles interacting with the real world counterpart of the simulated set up and is bottoms up approach of solving this equation.

The simulation method is stochastic for it relies on random variables to create the particles.

3.2 Change in weight due forced decay of neutrons

Particles in Monte Carlo simulation are often given a statistical weight. This weight is not a physical quantity, but is a measure of many relative real world particles they represent; note that this weight does not have to be an integer.

In this simulation the difference in time-scales causes large periods without precursor decay and hence no prompt neutrons. To solve this problem all precursors are forced to decay in each time interval. The weights are then adjusted to compensate. This is described by Legrady and Hoogenboom [7].

The probability that a precursor has a forced decay at time t inside a time interval between t_1 and $t_1 + \Delta t$ is chosen to be uniformly distributed:

$$\bar{p}(t) = \frac{1}{\Delta t} \tag{3.2}$$

The probability of a precursor having natural decay is given by

$$p_i(t) = \lambda_i e^{-\lambda_i(t-t_0)} \tag{3.3}$$

When using multiple precursor groups this becomes:

$$p(t) = \sum \frac{\beta_i}{\beta} \lambda_i e^{-\lambda_i(t-t_0)} \tag{3.4}$$

Note that all precursor groups are now combined into one precursor particle. This particle combines all decay times, hence it does not have a true exponential decay, although all the individual precursor groups which make up this particle do. The weights must be adjusted to make the probability of forced decay multiplied by its weight equal the probability of natural decay. The weight of the delayed neutron particle becomes:

$$w_n(t) = \frac{p(t)}{\bar{p}} = \Delta t \sum \frac{\beta_i}{\beta} \lambda_i e^{-\lambda_i(t-t_0)} \tag{3.5}$$

The average weight resulting from forced precursor decay can be determined by:

$$w_{n,av} =< w_n >= \frac{1}{\Delta t} \int_{t_1}^{t_1+\Delta t} \Delta t \sum \frac{\beta_i}{\beta} \lambda_i e^{-\lambda_i(t-t_0)} dt \tag{3.6}$$

Which is equal to:

$$w_{n,av} = \sum \frac{\beta_i}{\beta} e^{-\lambda_i(-t_0)} (e^{-\lambda_i t_1} - e^{-\lambda_i(t_1+\Delta t)}) \tag{3.7}$$

The precursor weight accounting for the change in weight due to the decay over time is:

$$w(t) = w \sum \lambda_i \beta_i \exp^{-\lambda_i(t-t_0)} \tag{3.8}$$

3.3 Precursors at start-up

When starting the simulation, a criticality calculation is done until the source has reached a stable equilibrium. This equilibrium is the steady state neutron distribution. Using this distribution and Eq. (3.5), the precursor and prompt neutron distributions can be calculated.

When a particle decays exponentially, the age of the particle does not influence it's chance of decay. For the particle has the same probability for decaying on each time step due to the exponential nature of this decay. However, for combined precursor family's, the decay is no longer truly exponential and hence the age of the group does matter for it's chance of decay, see Section 3.2. Different precursors from different groups decay with different constants therefore the ratio between precursor groups change over time. Therefore the make up of a combined precursor particles also changes over time until it has reached steady state. The difference between the precursor family distribution at the creation of a precursor and the family distribution in steady state is shown in Fig. 3.1.

To compensate for this change in precursors, the combined precursor particles should be started with an age between $-\infty$ and 0. This way the ratio between the different precursor groups is correct. Another method is to start precursors that are present at $t = 0$ with the steady state family distribution. This is done in the case of this work.

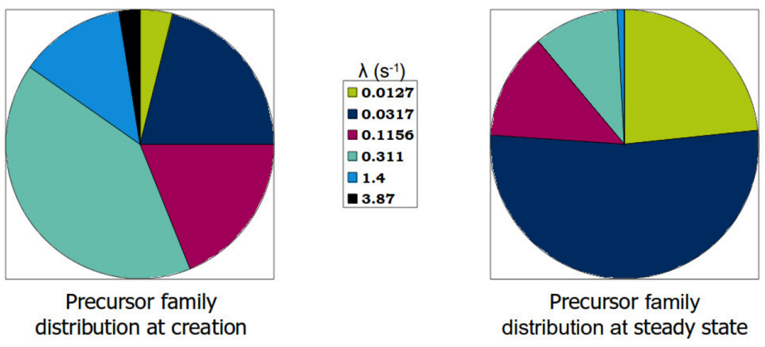


Figure 3.1: The difference in distribution of precursor family at creation an at steady state.[1]

In this work two types of precursors are distinguished, those created at the start and those created at $t > 0$. The precursors created for the start using a steady state calculation will be called Type II precursors in this work. Precursors created during the simulation will be called Type I precursors.

3.4 Prompt neutron fraction

Both prompt and delayed neutrons have a chance to start a neutron chain. The fraction of prompt neutrons is given by

$$\frac{n_0}{n_0 + d_0} \tag{3.9}$$

Here n_0 is the number of prompt neutrons at the start of the time interval and d_0 is the number of delayed neutrons that is created during this time interval.

Using Eq. (3.7) the number of delayed neutrons can be calculated

$$d_0 = \sum_i C_{0,i} \frac{1}{\Delta t} \int_t^{t+\Delta t} \frac{p_i(t)}{\bar{p_i}(t)} dt \tag{3.10}$$

which becomes

$$d_0 = \sum_i C_{0,i} \int_t^{t+\Delta t} \lambda_i e^{-\lambda_i t} dt \tag{3.11}$$

and then

$$d_0 = \sum_i \frac{\beta_i}{\lambda_i} \nu \Sigma_f \phi(\mathbf{r})(1 - \lambda_i e^{-\lambda_i \Delta t}) \tag{3.12}$$

Which can be substituted in Eq. (3.9) to get

$$\frac{n_0}{n_0 + d_0} = \frac{1}{1 + \sum_i \frac{\beta_i}{\lambda_i} \nu \upsilon \Sigma_f (1 - \lambda_i e^{-\lambda_i \Delta t})} \tag{3.13}$$

When using the same system properties described in Section 5.1 and a time interval of 100 ms, around 10% of the neutron chains is started by prompt neutrons of the last interval and 90% is started by delayed neutrons.

3.5 Variance reduction

To increase the computational efficiency of Monte Carlo simulations, variance reduction techniques can be applied. These can be divided into two general classes:

1. **Population control techniques**
2. **Event bias techniques**

Not all particles have the same contribution to the result. In population control techniques the particle distribution and particle weights are adjusted so the particles populate more interesting areas of phase space. In event bias techniques, physical events (such as scattering) and particle weights are adjusted to populate more interesting areas of phase space. [8]

3.5.1 Figure of merit

The Figure of Merit (FoM) is a quantity used to define the computational effectiveness of a simulation. It is defined in (3.14);

$$FoM = \frac{1}{RE^2 T} \tag{3.14}$$

with RE the relative error en T the time the computer needs for the simulation. N.B. this factor T results in a more subjective value for the Figure of Merit; the value for the Figure of Merit differs between computers and even slightly between different runs on the same computer.

3.5.2 Implicit capture

Implicit capture forces all particles to survive the collision and must hence reduce the weight of the surviving particle by the probability it will not get captured. The new weight becomes [9]:

$$W_{\text{new}} = W_{\text{old}}(1 - \Sigma_a/\Sigma_t) \tag{3.15}$$

where Σ is the macroscopic cross section which characterizes the probability of a neutron-nuclear reaction for the nucleus. Σ_a characterizes the probability for absorption while Σ_t characterizes the total probability for interaction.

Implicit capture must also use some form of low-weight cut-off such as Russian roulette introduced in Section 3.5.3 to eliminate particles with a very low weight.

For instance, statistics may require on average 60 particle out of a 100 particles to survive a certain event. Using implicit capture as explained in Section 3.5.2 every particle can be forced to survive however each particle should now have its weight reduced to 60% of its former weight.

3.5.3 Russian roulette

The computational cost of a Monte Carlo simulation using implicit capture can be reduced using Russian roulette. The Russian roulette variance reduction method causes particles with low expected contributions to the final response from being simulated. When the weight of a particle w goes below w_{RR} the particle undergoes Russian roulette. This value of w_{RR} may require some optimising. Due to the dynamic nature of this work it is set as a predefined percentage of the average weight. This value is selected to be 12.5%.

Then a survival weight must be defined, this is usually $w_s = 2w_{RR}$ as in the case of this work. Now the particle is killed with a probability of $1 - \frac{w}{w_s}$. The probability a particle survives thus is $\frac{w}{w_s}$. Surviving particles are given the w_s. Russian roulette always reduces computational cost but increases variance. Note that on average the total weight is conserved [9].

3.5.4 Particle splitting

Particle splitting can be used with weighted particles. It splits the existing particle into two if the particle weight exceeds a predefined amount, this prevents the particle weight from becoming too large. If the particle weight w exceeds a users defined limit w_{high}, the particle is split into N particles, each with weight w/N. Due to the dynamic nature of this work, w_{high} is defined as a set percentage of the average weight. This value is selected to be $w_{high} = 2w_{avg}$. When split, two particles are created of weight $\frac{w}{2}$ both retaining information(position,velocity, etc.) of the original particle.

The method has a lot of similarities with Russian roulette both conserves the total weight and adjust the number of particles depending on probability. Particle splitting however creates extra particles while Russian roulette kills particles. Splitting always decreases variance but increases computational cost [9].

3.5.5 Branchless collision

Prompt chains branch. Branching causes variance in the power production because of the variance in chain length. The branchless collision method is a novel variance reducing method to reduce the variance introduced by this branching, introduced by Sjenitzer and Hoogenboom [10].

The technique sets the number of neutrons to survive a collision to exactly 1 and changes the particle's weight accordingly. This can be a scattering neutron or a fission neutron. The particle retains its probability P_s to have a scattering interaction with an unchanged simulation weight. If the particle does not scatter, it terminates producing a new fission neutron with weight:

$$W_{new} = W_{old} \frac{\nu \Sigma_f}{\Sigma_a} \tag{3.16}$$

with Σ_a the absorption cross section given by

$$\Sigma_a = \Sigma_t - \Sigma_s \tag{3.17}$$

This technique has the disadvantage that the weight of the neutron can increase significantly hence particle splitting is used in combination with this method.

Particles with high weight are split, this reintroduces branching into the simulation. And thus variance due to the variance in chain length is reintroduced [10]. Efficient population control and other techniques might be able to reduce this variance.

3.5.6 The combing method

In this work the combing method is applied to reduce the computational cost for the simulation of reactor behaviour using a dynamic Monte Carlo technique. The combing method is a population control method, which can be used in codes using weighted particles. It distributes the total weight of the old number of particles over the new number of particles while maintaining the physics of the problem.

In the dynamic Monte Carlo simulation the population of neutrons vary over each time step, this causes a difference in computational cost for each time step. For an increased computational stability, i.e. distributing the computational cost roughly equal over each time step, the combing method could be used.

The combing method can be combined with other variance reduction techniques and has the potential for added flexibility when replacing other standard population control methods, such as Russian roulette. It is explained in greater detail in chapter 4.

Chapter 4

The variance reducing combing method

It is the aim of this work to research the feasibility of the combing method for achieving better population control. In the code used in this report Russian roulette is already used for population control over each time step. The combing method can be used to replace the existing Russian roulette method, however, a combination of both Russian roulette and the combing method can also be used.

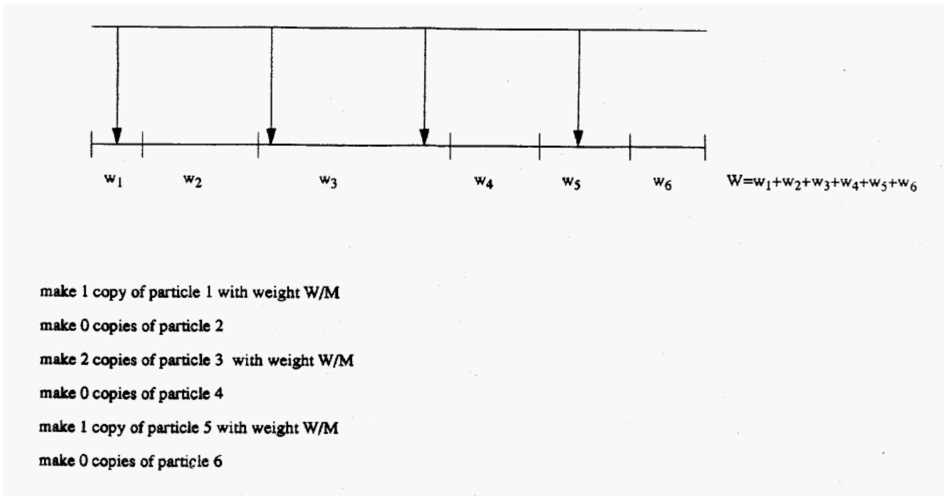
The combing method is a population control technique which can be used to exactly control the particle populations throughout a problem. The technique distributes (part of the) existing particles into the user defined new number of particles while preserving the expected weight of the original particles. This allows the user to determine the number of particles and distribute the particle weight evenly throughout the problem. This could reduce the variance. It also enables the user to spread the computational cost more evenly over each time step thus allowing the computational cost to be more stable and a more even spread of variance throughout the problem.

4.1 The simple comb

For K particles being **combed** into M particles, the **length** of the comb can be defined. The length of the comb is the total area on which the comb is applied. Since the comb is applied over the total weight of all particles, the length of the comb here simply means the total weight of all particles:

$$\text{length of comb} = W = \sum_{i=1}^{K} w_i \tag{4.1}$$

The new particles are chosen from the existing particles using an equally spaced weight interval on the total weight. To prevent a bias a random number is needed to select the first particle. Every time a new particle is selected this is called a tooth. This is shown in Fig. 4.1.



w1
w2
w3
w4
w5
w6
W=w1+w2+w3+w4+w5+w6
make 1 copy of particle 1 with weight W/M
make 0 copies of particle 2
make 2 copies of particle 3 with weight W/M
make 0 copies of particle 4
make 1 copy of particle 5 with weight W/M
make 0 copies of particle 6

Figure 4.1: A single comb. The included instructions determine the particles which are to be included in the postcombed system. Note that the spacing before the first selected particle is determined by the random number ρ. This (random) spacing is also used for the selection of the other particles throughout the combing process, for the spacing determines the exact location of the teeth. [2]

With each tooth equally spaced and the position selected as:

$$t_m = \rho \frac{W}{M} + (m-1)\frac{W}{M} \qquad m = 1, ..., M \tag{4.2}$$

Where ρ is the random number needed for selecting the starting position of the comb. Now each new particle is assigned a weight:

$$w_i' = \frac{W}{M} \tag{4.3}$$

Notice that the total weight is conserved because:

$$W_{\text{total}}' = w_i' M = \frac{W}{M} M = W \tag{4.4}$$

To find the exact locations of the teeth the integer j is defined as

$$j < \frac{w_i}{W/M} \le j + 1 \tag{4.5}$$

now depending on random number ρ, either j or $j+1$ teeth of the comb, will hit an interval of length w_i. This can be used to define the probabilities of teeth hitting an interval, as:

$$\begin{aligned} p_{i,j} &= j + 1 - \frac{w_i}{W/M} \quad \text{is the probability of j teeth in interval i} \\ p_{i,j+1} &= \frac{w_i}{W/M} - j \quad \text{is the probability of } j+1 \text{ teeth in interval i} \end{aligned} \tag{4.6}$$

If C_i is the total weight from the i^{th} particle after the combing process is done, the expected weight after combing is given by:

$$E[C_i] = p_{i,j} j \frac{W}{M} + p_{i,j+1}(j+1)\frac{W}{M} = \frac{w_i}{W/M}\frac{W}{M} = w_i \tag{4.7}$$

4.2 The double comb

For verification purposes, added flexibility and more advanced variance reduction, multiple parts of the same population can be combed in several steps. This can be used to investigate or enhance parts of the population of specific interest. This can be used for variance reduction.

The "double comb" uses two combs to comb two different populations, precursors and prompt neutrons in this case, separately into two different new populations while preserving their individual summed weights and the combined summed weights.This method also gives some insight in the flexibility of the combing method, being almost analogous to the single comb.

Suppose there are K precursors and L prompt neutrons, which are to be "combed" into M and N particles respectively. The total weight is now given by:

$$W_{\text{total}} = W_{\text{precursor}} + W_{\text{prompt}} \tag{4.8}$$

or

$$W_{\text{total}} = \sum_{i=1}^{K} w_{\text{precursor}} + \sum_{i=1}^{L} w_{\text{prompt}} \tag{4.9}$$

Now each delayed neutrons is given a weight

$$w'_{\text{precursor}} = \frac{W_{\text{precursor}}}{M} \tag{4.10}$$

and each prompt neutron is given a weight

$$w'_{\text{prompt}} = \frac{W_{\text{prompt}}}{N} \tag{4.11}$$

this gives a total weight of:

$$W'_{\text{total}} = w'_{\text{precursor}} M + w'_{\text{prompt}} N = \frac{W_{\text{precursor}}}{M} M + \frac{W_{\text{prompt}}}{N} N = W_{\text{total}} \tag{4.12}$$

hence all weights are conserved. Note this process can be repeated for more types of particles. Also to select the position of the first teeth, a new random variable should be used for every comb to prevent possible bias. The double comb allows the precursor and prompt neutron populations to be kept constant. This can be done in absolute numbers or as a percentage. These numbers can be be calculated at the end of each time step or they can be kept constant since the start of the dynamic part of the simulation. Also a bias can be introduced preferring one type of particle to the other, which can be useful for populating more interesting parts of phase space and thus increasing the Figure of Merit in this part of phase space.

4.3 Weight thresholds

The resulting number of particles from this combing is M, hence for $K > M$, K-M particles are removed from or 'combed out of' the simulation. Note that particles with a lower weight are much more likely to be 'combed out' of the simulation than particles with a higher weight.

The combing method has no minimum weight boundary for particles to be included in the postcombed system. Particles with a very low weight could be included in the postcombed system. This depends on the random variable ρ. However, the combing technique does have a weight boundary for particles to be excluded from the postcombed system. For a particle with a weight greater than the distance between two consecutive teeth will always be included in the postcombed system. This can be rephrased as: "Particles with $w_i > \frac{W}{M}$ have zero chance of elimination." Hence, the threshold weight for the combing method is $\frac{W}{M}$. Below this value particles are eligible for elimination from the simulation. Hence, this threshold depends on the total weight in the system.

Like the combing method, Russian roulette has no minimum weight for a particle to survive, it also has a weight threshold for particles to undergo this elimination technique. However, Russian roulette has several key differences in comparison.

1. Russian roulette is implemented on individual particles, while the combing method is implemented on the entire population at once.
2. Russian roulette relies heavier on statistics for a random value is needed for every single particle to undergo Russian roulette, where the combing method only uses one for the entire population.
3. Russian roulette conserves weight on average. Surviving particles are given a survival weight and the weight of eliminated particles is removed from the system.
4. Russian roulette allows the user to directly adjust the weight thresholds. The combing technique does **not** directly offer this flexibility.

In the case of this work, Russian roulette uses a percentage of the average weight as its weight thresholds. The maximum weight for a particle to undergo Russian roulette is chosen to be $w_{t\text{-}high} = 2w_{avg}$, surviving particles are also given this weight. It is useful to use a percentage of the average weight, for the average weight is not conserved (i.e. the system is not normalized). This means that naively using a constant would raise or lower the relative threshold per usage/time step. Optimisation of these chosen threshold parameters can be quite some work, for either extensive knowledge of the given system is required or an optimum has to be found using trial and error.

The combing technique does not, as stated previously, directly offer the flexibility of adjusting the weight thresholds. This can only be achieved by adjusting the wanted new number of particles. However the threshold setting appear to be quite optimal, for the total weight is exactly preserved and the weight is spread perfectly even throughout the postcombed system.

4.4 Combing different populations

In this problem the population is divided into the prompt neutrons and precursor particles. These populations require a difference in approach because of their different nature. The combing method is applied to both populations separately. To do this the population is first split into the prompt neutron population and into the precursor population. All the information (e.g. positions) of the particles are stored in vectors containing the information for one of these split populations. Because two variance reducing combs are used (one for each population) a new number of particles must be defined for each population separately.

4.4.1 Combing prompt neutrons

For the combing of prompt neutrons a simple comb can directly be applied. The newly created vector, contains the weight the prompt neutrons have. The new number of prompt neutrons can be selected in several ways. The most logical choice is to determine the percentage of prompt neutrons of the old total particle population and multiply this percentage with the wanted new total number of particles. The desired new total number of particles can be chosen to be constant throughout the problem in attempt to divide computational cost equally throughout the problem. However, other options are available such as returning to the original number of prompt neutrons calculated in the steady state (time independent) solution.

4.4.2 Combing precursors

In the simulation discussed in this report three different types of weight important in combing the precursor population can be distinguished.

1. The **precursor weight**, this is the weight contained in the weight bank for precursors. Determining the weight the precursor particle had on it's moment of creation. This weight, however, does not take into account the decay over time of the precursor.

2. The **precursor timed weight**, this weight takes into account the change in weight due to the decay over time of the precursor groups. The precursor timed weight can be calculated using Eq. (3.8).

3. The **expected delayed neutron weight**, this is the expected weight the neutron resulting from decay of the precursor has. Since this particle is the same as any other neutron the value should optimally be the same as any other (prompt) neutron. This weight is calculated by integrating/averaging over the next time step and hence is not an exact value. The expected delayed neutron weight can be calculated using Eq. (3.7).

To correctly comb the population without introducing any biases, the precursor timed weight should always be conserved. This is the weight the system interacts with. Preserving the expected delayed neutron weight instead of the timed precursor weight would introduce extra variance because of the averaging done over the next time step to calculate this weight.

In this work, the combing method is applied at the end of each time step. However, the postcombed particles will interact with the system in the next time step. Hence, it would be incorrect to use the current time step for the calculation of the precursor timed weight and expected delayed neutron weight. This is solved by calculating these weights for the next time step.

It should be noted that the weight resulting from combing needs to be calculated back to precursor weight when stored. This should be done because the weight banks used in this problem contain the precursor weight.

4.5 A decay-weighted comb for precursors

Let the **inclusion factor** be a user defined quantity indicating the importance of inclusion of a weighted particle in the postcombed system. The comb discussed in this section uses both the weight and inclusion factor of a particle for its inclusion in the postcombed system while preserving the total weight. It is otherwise similar to the simple comb. Hence this comb allows the user to comb the system over another quantity than the particle weight while preserving the total weight.

In the system discussed in this report, the timed precursor weight should be preserved, while combing the system over the expected delayed neutron weight. This is done to create a more even spread of weights as the particles interact with the system, as is discussed in Section 4.4.2. Hence the inclusion factor here is the summed decay factor from Eq. (3.7) needed to calculate the expected delayed neutron weight from the precursor timed weight.

If once again the starting number of particles is taken to be K which are combed into M particles, u_i is the expected delayed neutron weight and w_i is the timed precursor weight (which should be preserved), the inclusion factor can be defined as

$$I_i = \frac{u_i}{w_i} \tag{4.13}$$

this is taken to simply be the factor between the the expected delayed neutron weight and the timed precursor weight.

The total expected delayed neutron weight is

$$U = \sum_{i=1}^{K} u_i \tag{4.14}$$

this quantity is used to determine the new location of the tooth, which now should take into account both inclusion factor and weight.

$$t_m = \rho\frac{U}{M} + (m-1)\frac{U}{M} \qquad m = 1, ..., M \tag{4.15}$$

To find the exact locations of the tooth the integer j is defined as

$$j < \frac{u_i}{U/M} \leq j+1 \tag{4.16}$$

depending on random number ρ, either j or $j+1$ teeth of the comb, will hit an interval of length u_i. This can be used to define the probabilities of teeth hitting an interval, as

$$\begin{aligned} p_{i,j} &= j+1-\frac{u_i}{U/M} \quad \text{is the probability of j teeth in interval i} \\ p_{i,j+1} &= \frac{u_i}{U/M} - j \quad \text{is the probability of } j+1 \text{ teeth in interval i} \end{aligned} \tag{4.17}$$

The new weights are obtained by noting that the average number of particles of type i after combing is $M(u_i/U)$. Thus to conserve the expected total timed precursor weight requires:

$$\text{precomb weight} = w_i = w_i m_i M(u_i/U) = \text{expected postcomb weight} \tag{4.18}$$

where m_i is the weight multiplication for combed particles derived from particle i. Solving Eq. (4.18) for m_i gives:

$$m_i = U/(Mu_i) \tag{4.19}$$

The postcombed weight of all particles derived from the i^{th} precombed particle becomes:

$$w_i' = m_i w_i = w_i U/(Mu_i) = U/(MI_i) \tag{4.20}$$

This comb is constructed to conserve the total weight. This can also be seen by:

$$n_i = \text{number of 'hits' of the } i^{th} \text{ interval} \tag{4.21}$$

then using Eq. (4.20)

$$W' = \sum_{i=1}^{K} n_i \frac{U}{MI_i} \tag{4.22}$$

Taking the expectation of Eq. (4.21) and using Eq. (4.17) yields

$$\begin{aligned} E[W'] &= E\left[\sum_{i=1}^{K} n_i \frac{U}{MI_i}\right] = \sum_{i=1}^{K} \frac{U}{MI_i} \left(j p_{i,j} + (j+1) p_{i,j+1}\right) \\ &= \sum_{i=1}^{K} \frac{U}{MI_i} \left(j(j+1-\frac{u_i}{U/M}) + (j+1)(\frac{u_i}{U/M} - j) \right) \\ &= \sum_{i=1}^{K} \frac{U}{MI_i} \frac{u_i}{U/M} = \sum_{i=1}^{K} w_i = W \end{aligned} \tag{4.23}$$

This selection of particles is shown in Fig. 4.2.

Note that unlike the simple comb the total weight is not exactly conserved. Only the expected value of the weight is conserved. Also note that since u_i and w_i are known quantities, it is not necessary to calculate I_i explicitly.

4.6 Using a single comb for the total population

Another way to comb the population would be to comb the prompt neutron and delayed neutron population at the same time. This method recognizes the fact that both the delayed neutron as the prompt neutron have no real differences hence the same weight could be given to these particles. This method is simpler in application but has several drawbacks.

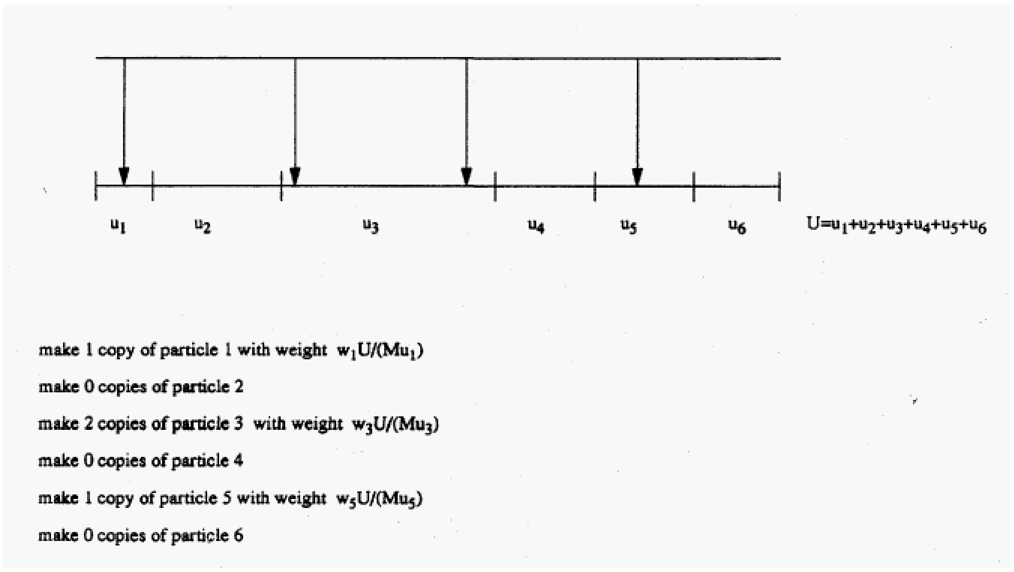

u_1
u_2
u_3
u_4
u_5
u_6
$U=u_1+u_2+u_3+u_4+u_5+u_6$
make 1 copy of particle 1 with weight $w_1U/(Mu_1)$
make 0 copies of particle 2
make 2 copies of particle 3 with weight $w_3U/(Mu_3)$
make 0 copies of particle 4
make 1 copy of particle 5 with weight $w_5U/(Mu_5)$
make 0 copies of particle 6

Figure 4.2: An inclusion factor weighted comb. The instructions included once again determine which particles are included in the postcombed system. [2]

1. The system has less flexibility; by using two combs it is possible to adjust the two different populations in respect of each other.
2. The expected delayed neutron weight is conserved instead of the timed precursor weight. Due to the method of calculation this introduces extra variance.

The second drawback could be solved by using inclusion factor combing for this comb, defining the inclusion factor for prompt neutrons as 1 (for the normalized case), defining the inclusion factor as the factor between timed precursor weight and expected delayed neutron weight (see Eq. (4.13)). This however increases complexity of the system again and once again requires inclusion factor combing to be used.

Using a single comb has the additional advantage that it does not (drastically) change the order of particles, whereas using two combs split the population into two separate parts; one part prompt neutrons and one part precursors. This could increase the computational cost.

Chapter 5

Calculations on a sample problem

5.1 The sample problem

To investigate the functioning of the simulation techniques, a sample problem has been set up. In this sample problem a small rectangular box is simulated. The box is 10 by 12 by 24 centimetre and is surrounded by a vacuum. The system properties are given in Table 5.1. The system has one energy group but there are six families. These are given in Table 5.2.

The size of a time step used in this set-up is 100 ms and the standard number of particles is 10^6. A reactivity is inserted in the system at $t = 10$. This is done by decreasing the Σ_a of the system from 0.5882 cm^{-1} to 0.5870 cm^{-1}.

To verify the results produced by the Dynamic Monte Carlo code, the same problem has been calculated using a point kinetics code. This is shown in Fig. 5.1. Note that the point-kinetics model is an accurate model for such a simple geometry.

The code used in this work already implemented the Russian roulette method for population control, as explained in Section 3.5.3. It is not necessarily the goal to replace this Russian roulette however it is a logical benchmark for investigating the effectiveness of the combing method. Note that some form of population control is required due to the use of other variance technique's, e.g.

Table 5.1: The material properties used in the sample problem. The box is made out of a homogeneous material.

Material Properties
$\Sigma_t = 1$ cm^{-1}
$\Sigma_f = 0.25$ cm^{-1}
$\Sigma_s = 0.4118$ cm^{-1}
$\nu = 2.5$
$\beta = 0.00685$
$\upsilon = 2.2 \times 10^4$ cm

Table 5.2: The precursors are divided in six families. Here the fractions and decay constants per precursor family i are given. Also the total delayed fraction and average decay constant are shown.

Group	$\lambda(s^-1)$	β
1	0.0127	0.00026
2	0.0317	0.001459
3	0.1156	0.001288
4	0.311	0.002788
5	1.4	0.000877
6	3.87	0.000178
av/tot	0.0784	0.00685

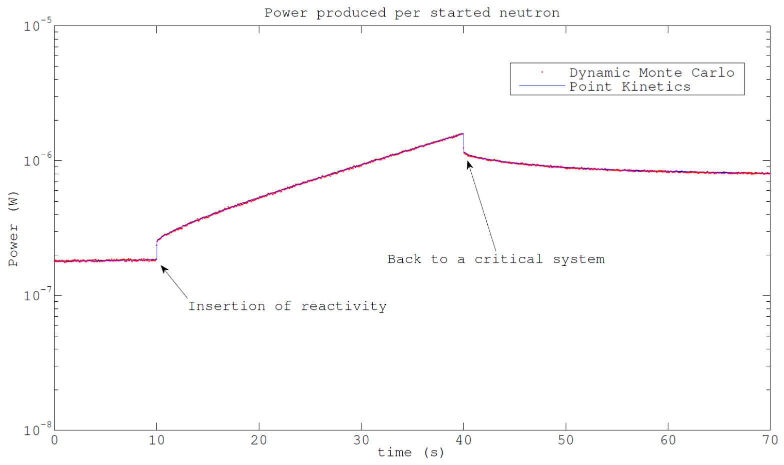


Figure 5.1: A critical system with a reactivity insertion at $t = 10$, the reactivity is set back to 0 at $t = 40$. Now the system returns to a new stable state. The dynamic Monte Carlo simulation agrees with the point-kinetics analysis of the system.[1]

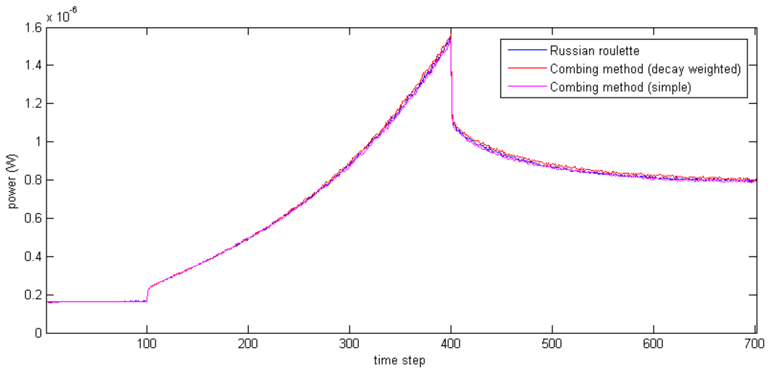


Figure 5.2: The power produced per started neutron for both the combing method and Russian roulette.

branchless collision, hence a simulation without any population control cannot be chosen as a benchmark.

5.2 Calculations done one using multiple precursor families

For this simulation precursor families of Table 5.2 was used. The rest of the problem on which the calculations are done, is the same as discussed in Section 5.1. The starting number of particles used is 10^6. The calculations for these simulations were done using six CPU's for each run.

5.2.1 Power behaviour

First of all the proper functioning of the combing method needs to be demonstrated. This can be done by inspecting the power behaviour resulting from both methods. This is shown in Fig. 5.2.

The power behaviour for all methods appear to agree, which implies the proper function of the combing method. To ensure the combing method does not introduce a bias the errorbars are included in Fig. 5.3.

Unfortunately, Fig. 5.3 still doesn't clearly show if the result is unbiased. Therefore a close up is made, this is shown in Fig. 5.4. As can be seen in this picture, the errorbars (standard deviation) of the different combing methods are within the range of the errorbars of the Russian roulette method, hence the result can be deemed to be unbiased.

5.2.2 Figure of Merit

The Figure of Merit of the different techniques is shown in Fig. 5.5.

It is important to know what causes this decrease in the Figure of Merit. Using the definition given by Eq. (3.14), this can be caused by a increase in relative error or by a increase in calculation time, i.e. higher computational expenses.

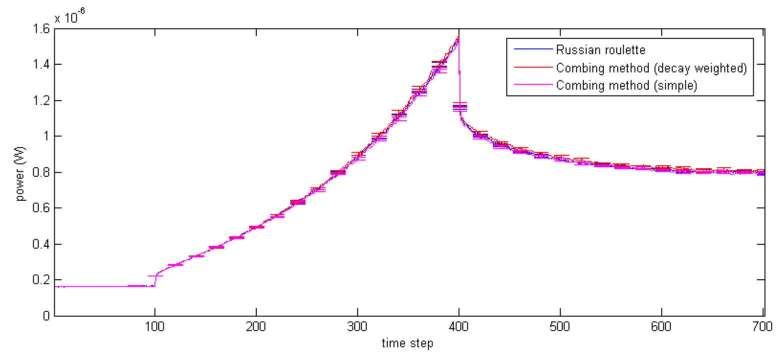


Figure 5.3: The power produced per started neutron for both the combing method and Russian roulette, here the errorbars are included at every 20 time steps.

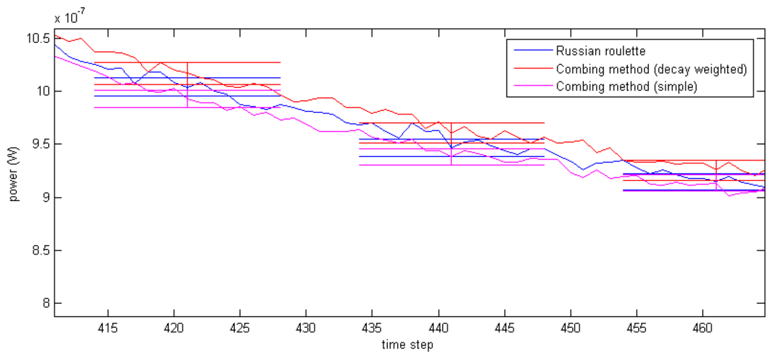


Figure 5.4: A close up of the power produced per started neutron for both the combing method and Russian roulette, here the errorbars are included at every 20 time steps.

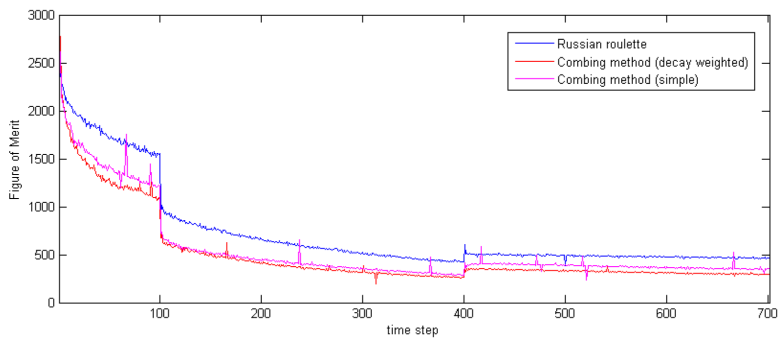


Figure 5.5: The total Figure of Merit.

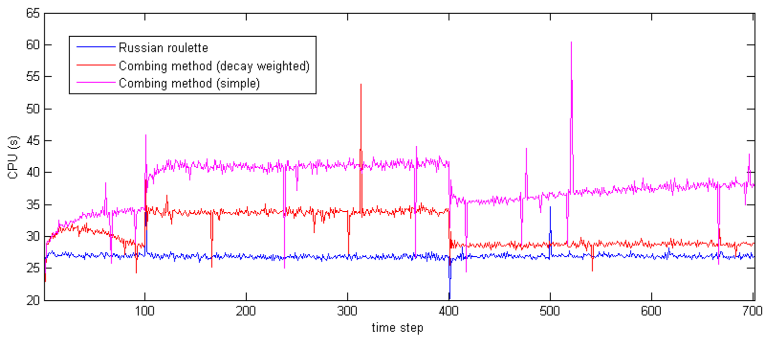


Figure 5.6: The computer cost per time step in seconds.

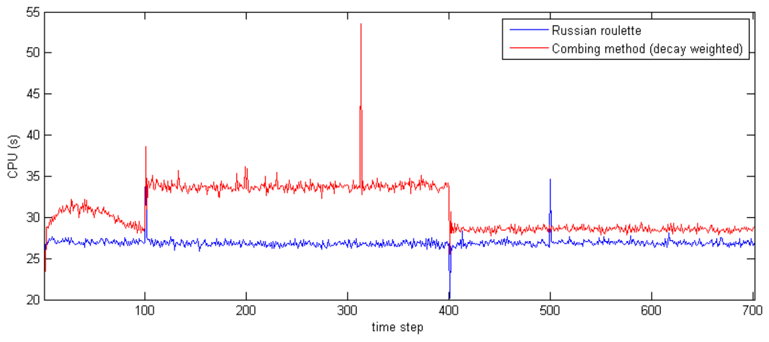


Figure 5.7: The computer cost per time step in seconds.

The computer cost per time step is shown in Fig. 5.6 and the standard deviation is given in Fig. 5.8. These figures clearly show that the decrease in Figure of Merit can be contributed to the increase in computational cost, since the standard deviation roughly stays constant while the computational cost increases and becomes erratic. The erratic behaviour of the computer cost could be due to the fact that computer cost is a subjective measurement, where the specific computer and conditions measure. To check this influence another simulation was done. The computer cost of that run is shown in Fig. 5.7. As can been seen from this graph, the behaviour is less erratic. Hence, this erratic behaviour is likely due to unpredictable behaviour of the computer the simulation was run on.

5.2.3 Difference in CPU load

To understand what causes this increase in computational cost it is important to inspect the make-up of this computational cost.

For an arbitrary time step, taken to be 401 the difference in distribution for the non-master CPU's is shown in Tables 5.3, 5.4 and 5.5. Here the weight, is the weight as contained in the weight banks.

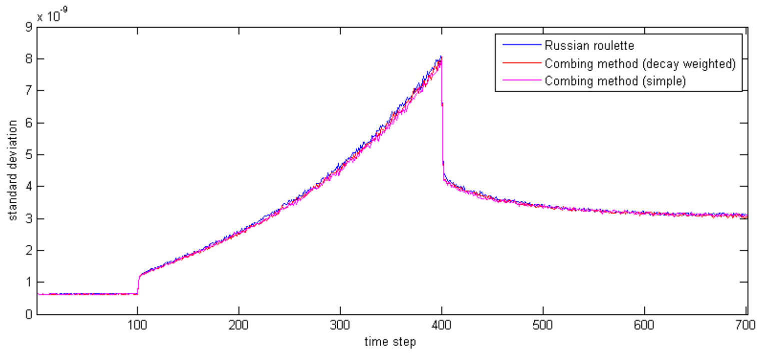


Figure 5.8: The standard deviation on power produced per started neutron for each time step for both the combing method and Russian roulette.

Table 5.3: Distribution in CPU load for the decay weighted combing method for time step 401.

CPU number	# particles in	weight in	# particles out
1	170334	383090505.672068	184393
2	170334	298691339.822376	184089
3	170334	207295190.654775	183992
4	170334	120766689.512432	184378
5	170334	32540994.4865478	108438

Table 5.4: Distribution in CPU load for the simple combing method for time step 401.

CPU number	# particles in	weight in	# particles out
1	170334	322518612.493628	183014
2	170334	286814284.785877	182885
3	170334	229463719.042564	184090
4	170334	173475609.210797	185874
5	170334	66879549.7075076	113132

Table 5.5: Distribution in CPU load for the Russian roulette for time step 401.

CPU number	# particles in	weight in	# particles out
1	89967	233382515.583194	90660
2	89967	235646368.740038	91169
3	89967	231291946.041189	90570
4	89967	233745258.155254	90200
5	89967	236883249.928050	90468

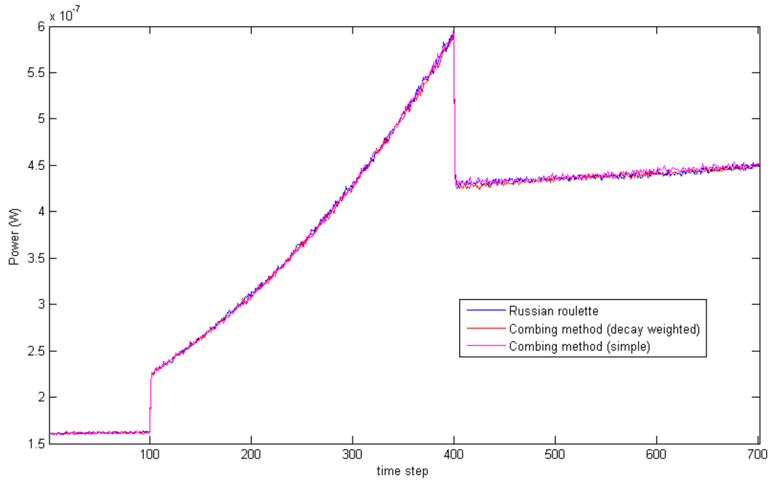


Figure 5.9: The power produced per started neutron for both the combing method and Russian roulette, here the errorbars are included at every 20 time steps.

5.3 Calculations done using one precursors group

The combining of different type of precursors in this rapport complicate the simulation, for it requires some mathematical techniques to efficiently simulate. This is explained in Chapter 2. However this may compromise the efficiency of the combing method. To see how the combing method functions without these technique, simulations were done using only one precursor group. For this simulation the average of Table 5.2 was used. The starting number of particles used is 10^6. The rest of the problem on which the calculations are done, is the same as discussed in section 5.1. These simulations were done using two CPU's for each run.

5.3.1 Power behaviour

The power behaviour of this simulation is shown in Fig. 5.9.

To see if this result is unbiased the errorbars are added in Fig. 5.10. A close up inspection gives Fig. 5.11. Here it can clearly be seen that the errorbars of both combing methods overlap and hence the results can be assumed to be unbiased.

5.3.2 Figure of Merit

The effectiveness of both population control methods, Russian roulette and the combing technique can best be demonstrated by calculating their respective Figure of Merit's for each time step. This is shown in Fig. 5.12.

The Figure of Merit can once again be decomposed into both the computational cost and the relative error. The computer cost per time step is shown in Fig. 5.13 and the standard deviation is given in Fig. 5.14. These graphs

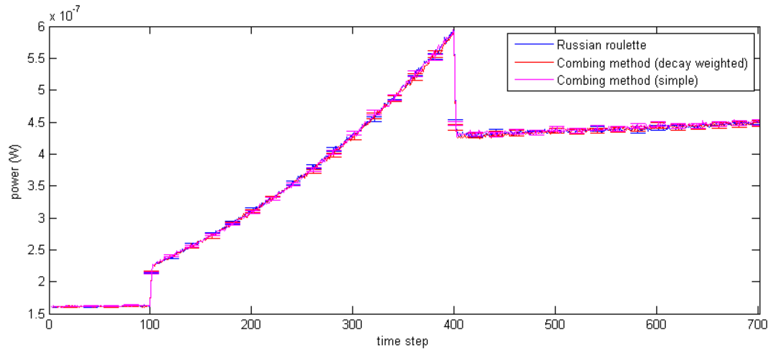


Figure 5.10: The power produced per started neutron for both the combing method and Russian roulette, here the errorbars are included at every 20 time steps.

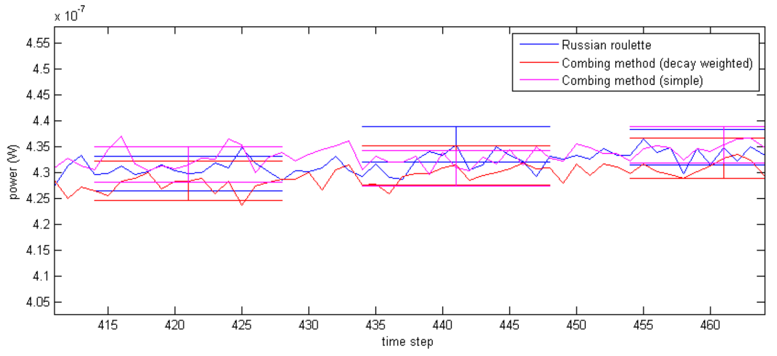


Figure 5.11: A close up of the power produced per started neutron for both the combing method and Russian roulette, here the errorbars are included at every 20 time steps.

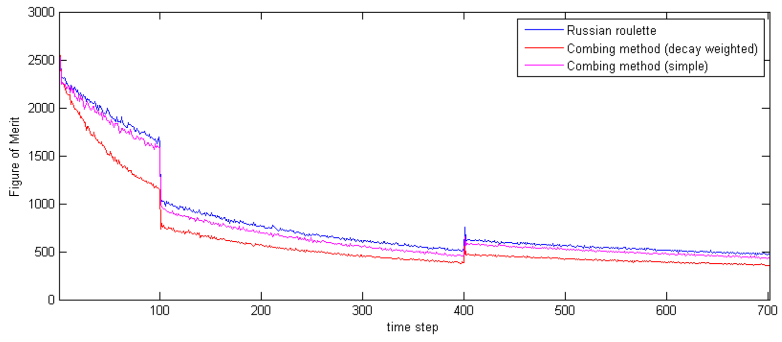


Figure 5.12: The total Figure of Merit.

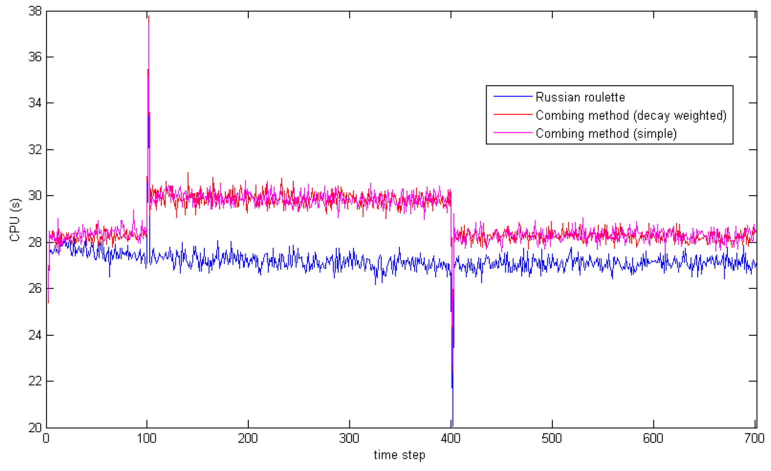


Figure 5.13: The computer cost per time step in seconds.

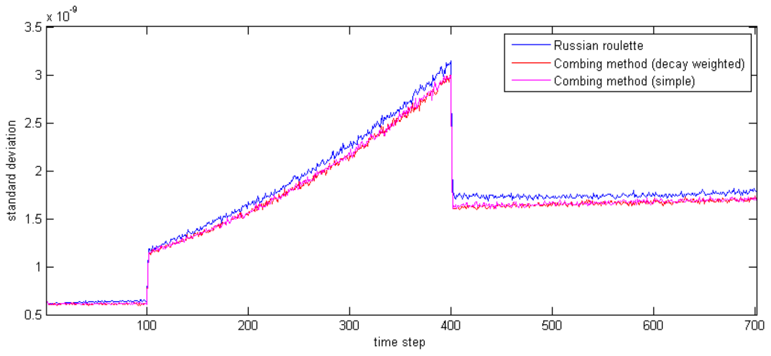


Figure 5.14: The standard deviation on power produced per started neutron per time step.

show that the decrease in the Figure of Merit for the combing method can be contributed to the increase in computational cost, as the standard deviation decreases slightly while the computational cost increases.

5.3.3 Population

An important part of the functioning of both techniques is their impact on the composition of the population. There are several aspects of the population on which can be focused. Most obvious is the total number of particles in the system, this is shown in Fig. 5.15. The measurements were done at the beginning of each time step.

It can be seen that the system using the combing method has a constant total number of particles while the system using Russian roulette sees a sharp decrease in number of particles throughout the system. To find the cause of these differences, the population is split in it's respective composition. This is

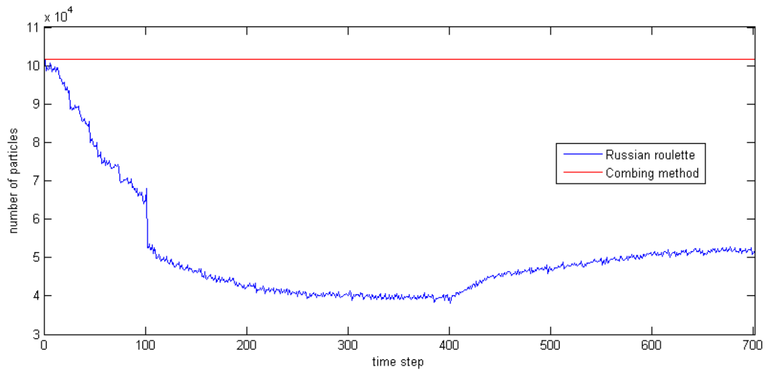


Figure 5.15: The total number of particles at the beginning of each time step.

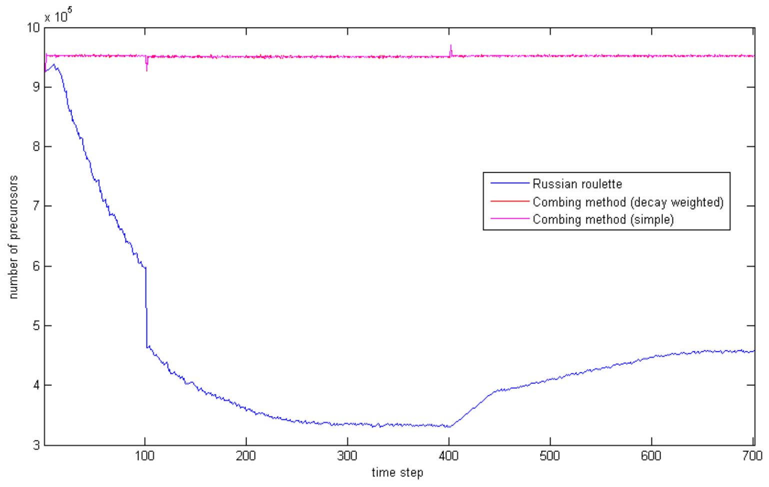


Figure 5.16: The total number of precursors at the beginning of each time step.

shown in Fig. 5.16 and Fig. 5.17 which show the precursor and prompt neutron population over time.

5.3.4 Particle splitting

Another aspect of both methods worth investigating is the number of times particle splitting occurs during the simulation. This is a defining feature of both methods, and can be interesting for further variance reduction. The results are shown in Fig. 5.18.

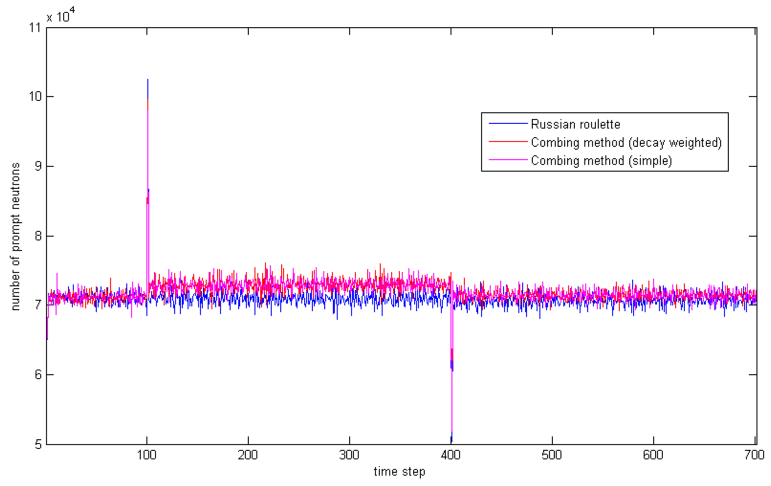


Figure 5.17: The total number of prompt neutrons at the beginning of each time step.

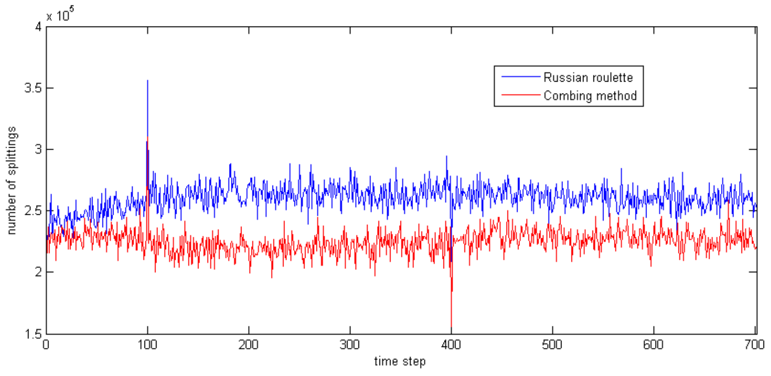


Figure 5.18: The number of times a particle undergoes particle splitting for each time step.

////# Chapter 6

Conclusions and recommendations

6.1 Figure of Merit

As seen in Fig. 5.5, the Figure of Merit is higher for Russian roulette then for the combing method. This is caused by the increase in computational cost. This increase in computational cost is higher during the transient (between $t = 10s$ and $t = 40s$). This implies a higher impact on computational cost during transient behaviour for the combing method than Russian roulette.

The CPU behaviour of the combing method also shows large peaks at times with no important event attached. This means that the combing method is less reliable in terms of constant behaviour.

Note that the decay weighted combing method does an overall better job, both in Figure of Merit as computational cost, than the simple combing method. Also the peaks in computational cost are lower on average for the decay weighted combing method than for simple method, implying the extra cost and erratic behaviour are dependant on the decay weight of the precursors.

For the simulation using only one precursor family, Russian roulette still performs better then the combing method when inspecting the Figure of Merit. These computations had a lower overall computational cost and are less representative of the system on which the the variance technique is used. The relative error is slightly lower, which compensates for the increase in computational cost. Also the computational behaviour of the combing method seems more stable than the simulation using multiple precursor families, however this is likely due to the specific computer run. Both the decay weighted combing method and the simple combing method seem to perform equal in these simulations. Note that, since the difference between the two combing methods has disappeared, the dependence on decay weight is increased by using multiple precursor families.

6.2 Stability of the population

As can be seen from Fig. 5.15 the total number of particles for the combing method is more stable than when using Russian roulette. This is due to the number of precursors remaining constant throughout the problem as seen in Fig. 5.16. This is an inherent advantage of using the combing method. Also the combing method can be implemented in several ways adding to the flexibility of the technique. For instance it can keep the percentage of precursor constant or the total number of particles, whereas Russian roulette does not offer this flexibility and relies heavier on statistics. The combing method used in this rapport is currently unable to handle starting populations much larger then 10^6 particles. This is a serious disadvantage, this could however be solved using some smart coding.

6.3 Interaction with particle splitting

The combing method sets all timed precursor weight and prompt neutron weight to the average of their population. This fact should lead to a lower Figure of Merit and better system stability. This is especially true for a system where particle splitting is used. For in a system with higher variance in weight distribution, particles reach the threshold weight for particle splitting more often than the system with the lower variance in weight distribution. This leads to lower branching in the system.

This particle splitting or (slight) branching of a prompt chain should lead to a lower Figure of Merit and stability of the system. More computer time is required as more particles need to be traced while the size of the population varies more over each time step.

It can be seen from Fig. 5.18 that particle splitting occurs less often when the combing method is used in stead of Russian roulette.

The decrease in particle splitting has the more complex advantage that GPU's cannot handle branching [11]. The possibility of further optimisation by implementing GPU's instead of CPU's require little to no branching. However the decrease in the amount of branching is in the order of several per cent and does not allow for drastic changes in the architecture of programming.

6.4 Order of particles

Because of the way the combing method functions, the order of particles is drastically changed. The banks containing the information of particles are now ordered by type. First the information of one type of particle is stored (precursor in this case) then the second type of particles is stored (prompt neutrons). This could explain some of the increase in computational cost, for a more even (some could say random) distribution of the different particles throughout the system would cause for a more even spread of computational cost. This also could cause a severe difference in computational load per used processor, which would cause the combing method to perform severely poorer, due to the fact that processors assigned with a relative light task have to wait until the processors relative heavy task are finished.

It can be seen in Tables 5.3, 5.4 and 5.5 that the spread of weight and simulated particles is indeed uneven for the combing methods whereas this is more or less even for the Russian roulette method.

This difference in computational load per processor increases for a higher difference between computational cost between the two types of particles, as is the case between using multiple precursor families, which increase the computational cost of calculating a precursor relative to that of a prompt neutron. This would explain the combing method performing worse than Russian roulette during regular simulation while during the simulation using only one precursor family it performs slightly better.

The difference in computational load also causes the system (total time) to react in an non-linear fashion between the number of CPU's used in the set-up. This also could be an explanation for the comparatively better results of the system using only one precursor family since this system used less CPU.

6.5 Future work

Usage of the combing method leads to a worse and less predictable performance of the system. Hence a direct implementation of the combing method would not make sense. However significant gain could be achieved by more efficient coding. For instance a subroutine could be added to return the particles in the post combed system to the order of the pre combed system, this could lead to the combing method achieving an overall better and more predictable performance, this could be the aim of future work. Also some specific research requiring added flexibility in terms of specific population control could make advantage of the combing technique for population more interesting parts of phase space could make advantage of the combing method.

Bibliography

[1] Bart L Sjenitzer and J Eduard Hoogenboom. *Possibilities and efficiency of long-time kinetic and dynamic Monte Carlo calculations.* 2011.

[2] Thomas E Booth. *A Weight (Charge) Conserving Importance-Weighted Comb For Monte Carlo.* 1996.

[3] Weston M Stacey. *Nuclear Reactor Physics.* WILEY-VCH Verlag GmbH & Co. KGaA, 2004, 2004.

[4] James J Duderstad and Louis J Hamilton. *Nuclear Reactor Analysis.* John Wiley and Sons, Inc., 1976.

[5] THJJ van der Hagen H van Dam and JE Hoogenboom. *Nuclear reactor physics, lecture notes ap3341.* http://www.janleenkloosterman.nl/reports/ap3341.pdf.

[6] H P Lopuhaä L E Meester F M Dekking, C Kraaikamp. *A Modern Introduction to Probability and Statistics: Understanding Why and How.* Springer, 2010.

[7] David Legrady and J Eduard Hoogenboom. *Scouting the feasibility of monte carlo reactor dynamics simulations.* 2008,.

[8] Thomas J T Kwan Mark G Gray, Thomas E Booth and Charles M Snell. *A Multicombd Variance Reduction Scheme for Monte Carlo Semiconductor Simulators.* 1998.

[9] Forrest B Brown. *Fundamentals of Monte Carlo Particle Transport, lecture slides on Variance Reduction, Los Alamos National.* http://mcnp-green.lanl.gov/publication/pdf/LA-UR-05-4983_Monte_Carlo_Lectures.pdf.

[10] Bart L Sjenitzer and J Eduard Hoogenboom. *Variance reduction for fixed-source Monte Carlo calculations in multiplying systems by improving chain-length statistics.* 2011.

[11] Mark Harris and Ian Buck. *GPU gems 2: programming techniques for high-performance graphics and general-purpose computation.* Addison Wesley Professional, 2005.