Uniwersytet Jagielloński w Krakowie
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Michał Smolis
Nr albumu: 1064556

Zrównoleglenie i optymalizacja algorytmów w pakiecie symulacji wykrywacza materiałów wybuchowych

Praca magisterska na kierunku Informatyka

Praca wykonana pod kierunkiem
Prof. Dr hab. Pawła Moskala
Instytut Fizyki im. Mariana Smoluchowskiego

Kraków, 2015

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami. Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Kraków, dnia Podpis autora pracy

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Kraków, dnia Podpis kierującego pracą

Podziękowania

Składam serdeczne podziękowania mojemu opiekunowi naukowemu Panu Profesorowi Pawłowi Moskalowi za cenne wsparcie merytoryczne, uwagi i sugestie oraz za poświęcony mi czas i pomoc przy opracowaniu niniejszej pracy.

Dziękuję również całemu zespołowi, wraz z którym rozwijaliśmy pakiet symulacyjny, opisany w tej pracy. Dziękuję Panu Doktorowi Michałowi Silarskiemu, Pani Dominice Hunik oraz Panu Sławomirowi Tadeji za cenne wskazówki, wsparcie, miłą współpracę oraz cierpliwość.

Streszczenie

Zrównoleglenie i optymalizacja algorytmów w pakiecie symulacji wykrywacza materiałów wybuchowych

Współczesne metody wykrywania materiałów niebezpiecznych są ograniczone, szczególnie ze względu na brak mobilności. Do materiałów niebezpiecznych zalicza się między innymi materiały wybuchowe i narkotyki, których głównymi składnikami pierwiastkowymi są azot, tlen, węgiel i wodór. Na Wydziale Fizyki, Astronomii i Informatyki Stosowanej, w ramach projektu SABAT (*ang.* Stoichiometry Analysis by Activation Techniques), prowadzone są badania nad atomometrią - metodą pozwalającą zdalnie wykrywać substancje niebezpieczne. Atomometria polega na analizie stechiometrycznej badanej substancji w oparciu o widma energii kwantów gamma emitowanych w wyniku napromieniowania badanego obiektu strumieniem neutronów.

Aby wspomóc badania nad tą metodą, rozwijany jest pakiet oprogramowania do symulacji emisji neutronów, ich oddziaływania w zadanym materiale oraz rejestrowania kwantów gamma. W tej pracy przedstawione zostały podstawowe algorytmy oraz metody statystyczne, które zostały zaimplementowane we wspomnianym pakiecie symulacyjnym SABAT. Praca zawiera opis oraz wyniki testów jednostkowych wykonanych na opisanych metodach. Omówione zostały także problem optymalizacji wybranych metod oraz zrównoleglenia całego programu symulacji. W pracy zawarte są także opis działania symulacji oraz jej przykładowy wynik. Zrównoleglenie działania procedur spowodowało 4-krotne zwiększenie szybkości działania pakietu symulacyjnego SABAT dla takiej samej liczby procesów uruchomionych na czterordzeniowym procesorze.

Abstract

Parallelization and optimization of algorithms for the simulation package of the explosives detector

Modern methods of detection of hazardous materials are limited, especially when mobility is considered. Explosives and drugs can be considered as such hazardous materials. Main elemental components of these materials are nitrogen, oxygen, hydrogen and carbon. Reasearch presented in this thesis, aiming at the development of the method of atometry, is conducted in the framework of the SABAT project (Stoichiometry Analysis by Activation Techniques) at the Faculty of Physics, Astronomy and Applied Computer Science. Atometry is a method of stoichiometric analysis of a given substance by irradiating it with neutron beams and thus causing emmision of gamma quanta. Measurement of energy spectra of these quanta allows to determine chemical composition of an examined substance.

The software package for simulation of neutron emission, their interactions in a given environment and gamma quanta registration is developed in order to support the research on this method. Basic algorithms and statistical methods, implemented in the SABAT simulation package, are introduced in this thesis. Unit tests and their results are also described in this work. Moreover, problems of optimisation of a few selected methods and parallelization of the whole simulation are discussed. The thesis contains description of simulation's basic workflow and example of simulation's result. Final result of parallelization is acceleration by the factor of 4 for the same number of parallel processes run on the quad-core processor.

Spis treści

Wstęp

Współczesne metody wykrywania materiałów niebezpiecznych są ograniczone, szczególnie ze względu na brak mobilności. Do materiałów niebezpiecznych zalicza się między innymi materiały wybuchowe i narkotyki, których głównymi składnikami pierwiastkowymi są azot, tlen, węgiel i wodór. Na Wydziale Fizyki, Astronomii i Informatyki Stosowanej, w ramach projektu SABAT (*ang.* Stoichiometry Analysis by Activation Techniques), prowadzone są badania nad atomometrią - metodą pozwalającą zdalnie wykrywać substancje niebezpieczne. Atomometria polega na analizie stechiometrycznej badanej substancji w oparciu o widma energii kwantów gamma emitowanych w wyniku napromieniowania badanego obiektu strumieniem neutronów [1, 2].

Aby wesprzeć badania nad tą metodą, rozwijany jest pakiet oprogramowania do symulacji emisji neutronów, ich oddziaływania w zadanym materiale oraz rejestrowania kwantów gamma. Podstawowe działanie programu symulacyjnego polega na wygenerowaniu zadanej liczby neutronów i zasymulowanie zachowania neutronów w zadanym środowisku, korzystając z metod Monte Carlo. Program wymaga wcześniejszego sparametryzowania symulacji - zdefiniowania geometrii obiektów, ich składu pierwiastkowego i pozycji w środowisku symulacji. Neutron w trakcie symulacji może zareagować z jądrem atomu obiektu, w którym aktualnie się znajduje. W wyniku niektórych reakcji mogą powstać nowe cząstki, które także są symulowane. Program korzysta z bazy danych zawierającej podstawowe informacje potrzebne do symulacji, tj. skład i gęstość substancji, podstawowe informacje o pierwiastkach i ich izotopach oraz przekroje czynne na reakcję neutronów w funkcji ich energii.

W celu zaprojektowania wykrywacza należy przeprowadzić symulację dla różnych ustawień środowiska, różnych parametrów i ustawień przestrzennych generatorów i detektorów. Współczesne źródła neutronów osiągają aktywności rzędu 600 GBq [3], co odpowiada generowaniu 0.6×10^{12} neutronów na sekundę. Przy takiej liczbie cząstek symulacja może trwać bardzo długo, jako że każdy neutron wchodząc w reakcje z atomami może ulegać wielu rozproszeniom, które z kolei mogą prowadzić do powstania nowych cząstek. Analiza każdej cząstki wymaga zasymulowania jej drogi w zadanym środowisku, aż do momentu, gdy wyleci poza zdefiniowane granice lub zniknie w wyniku reakcji.

Podstawową motywacją do napisania tej pracy jest skrócenie długiego czasu wykonywania takiej symulacji. Celem tej pracy jest poszukiwanie sposobów przyspieszenia symulacji poprzez zrównoleglenie lub zmianę algorytmów, które są często wykorzystywane w programie. Omówione zostały także wybrane metody statystyczne, które są zaimplementowane w pakiecie.

W pierwszym rozdziale zostały zaprezentowane algorytmy generowania liczb pseudolosowych. W tym rozdziale został omówiony także problem generowania liczb losowych z rozkładu jednorodnego oraz dwie metody pozwalające przekształcić wybrany generator liczb losowych w generator liczb z innego rozkładu.

W drugim rozdziale omówione zostały przykłady generowania liczb losowych zaimplementowane w pakiecie SABAT. Na początku omówiony został generator liczb z rozkładu wykładniczego. Następnie został przedstawiony problem generowania liczb losowych z zadanym rozkładem wielomianowym – opartym o wielomiany Legendre'a. Rozdział drugi został zakończony przybliżeniem problemu jednorodnego generowania punktów na sferze. Przedstawione zostały cztery propozycje rozwiązań tego problemu. Porównanie efektywności implementacji wybranych metod znajduje się w piątym rozdziale.

Trzeci rozdział zawiera opis pakietu SABAT. Opisana została parametryzacja symulacji, która pozwala na zmianę liczby neutronów oraz ustawienia obiektów w przestrzeni symulacji. Program zawiera również opis podstawowego algorytmu działania. Rozdział kończy się opisem formatu danych wyjściowych.

W czwartym rozdziale przedstawiony został przykład zastosowania pakietu symulacyjnego. Zostało zaprezentowanych kilka przykładowych wyników różnych symulacji w formie wykresów.

Kolejny, piąty rozdział obejmuje testowanie opisanych metod. Każdy z podrozdziałów dotyczących kolejnych algorytmów rozpoczyna się propozycją testów jednostkowych, potwierdzających poprawność implementacji. Następnie zostały przedstawione opis i wyniki testów jednostkowych. W przypadku generatora liczb jednorodnych, na którym oparte są przedstawione w tej pracy algorytmy, zostało omówione zagadnienie testów statystycznych, w celu sprawdzenia ewentualnych korelacji i innych niepożądanych błędów występujących w generatorach liczb pseudolosowych.

W szóstym rozdziale przedstawione zostały najważniejsze różnice pomiędzy implementacją sekwencyjną, a implementacją równoległą. Rozdział rozpoczyna się opisem wykorzystanych narzędzi. Omówiony został problem ustawienia początkowego dla generatora liczb pseudolosowych oraz struktura działania wersji równoległej programu. Rozdział zakończony jest porównaniem szybkości obu implementacji.

Na zakończenie zostały podsumowane wyniki pracy. Została, tym samym, określona zasadność wdrożenia opracowanych metod do pakietu symulacyjnego SABAT.

Niektóre wykresy, algorytmy i tabele w tej pracy zostały napisane w języku angielskim, ze szczególnym naciskiem na nazwy funkcji i zmiennych opisanych w algorytmach. Spowodowane jest to większą zwięzłością prezentowanego materiału i brakiem podstaw do tłumaczenia na język polski elementów, które już zostały zaimplementowane w języku angielskim w docelowym pakiecie symulacyjnym. Nie powinno to jednak powodować utrudnienia w zrozumieniu tych materiałów.

1 Algorytmy generowania liczb losowych zaimplementowane w pakiecie SABAT

1.1 Omówienie generatora liczb pseudolosowych

1.1.1 Liczby losowe a liczby pseudolosowe

Liczbami losowymi nazywamy liczby otrzymywane w wyniku działania pewnego mechanizmu losowego [4]. Liczby takie można otrzymać m.in. rzucając monetą. Liczby pseudolosowe są szczególnym rodzajem liczb losowych, które zostały wygenerowane za pomocą określonego przepisu. Z tego względu nie są one w pełni losowe, zwykle są obarczone korelacjami pomiędzy kolejnymi wylosowanymi liczbami.

1.1.2 Generatory liczb pseudolosowych

Generatorem liczb pseudolosowych nazywamy program lub jego część odpowiedzialną za generowanie kolejnych liczb pseudolosowych. W tej pracy zostanie omówiona implementacja generatora z rodziny generatorów opartych o wzór:

$$x_n = (ax_{n-1} + c) \bmod m \tag{1}$$

Powyższa formuła jest przykładem liniowego generatora kongruencyjnego (*z ang.* Linear Congruential Generator) [5]. a, c i m to odpowiednio dobrane stałe, takie że:

$$\begin{aligned} &0 < m, \\ &0 < a \ < m, \\ &0 \leq c < m, \\ &0 \leq x_0 < m, \\ &0 < n, \end{aligned}$$

gdzie x_0 to wartość początkowa (*z ang.* seed), wymagana do początkowej konfiguracji generatora.

1.1.3 Generator oparty o komplementarne mnożenie z przeniesieniem

Generator CMWC oparty o komplementarne mnożenie z przeniesieniem (*z ang.* Complementary-Multiply-With-Carry generator) został zaproponowany przez G. Marsaglia [6]. Jest on ulepszeniem koncepcji liniowego generatora kongruencyjnego. Pozwala on na uzyskanie bardzo dużych okresów

i wykorzystanie elementarnej arytmetyki liczb całkowitych używanej przez komputery. Generator ten jest oparty o wzór:

$$x_n = (b-1) - (ax_{n-r} + c_{n-1}) \bmod b, \tag{2}$$

$$c_n = \left\lfloor \frac{ax_{n-r} + c_{n-1}}{b} \right\rfloor, \tag{3}$$

gdzie:

$$\begin{aligned} &0 < r \leq n, \\ &0 < m, \\ &0 < a \ < m, \\ &0 \leq c_{r-1} < m. \end{aligned}$$

W jednym z kolejnych podrozdziałów przedstawione zostaną wyniki testów dla tego generatora porównane z wynikami innych generatorów: *rand* i *rand48* z biblioteki standardowej C i implementacji generatora *mt19937* i *ranlux* w bibliotece *random* będącej częścią standardu języka C++ w wersji z 2011 roku [7].

1.1.4 Algorytm generowania liczb pseudolosowych za pomocą generatora CMWC

Poniższy algorytm generatora CMWC został napisany w języku C++ i posiada okres rzędu 2^{131104}:

```
unsigned long Q[4096];
unsigned long c = 362436;
unsigned long i= 4095;

unsigned long CMWC4096(void)
{
  unsigned long long t, a= 18782LL;
  unsigned long x;
  i = (i+1) & 4095;
  t = a*Q[i] + c;
  c = (t>>32);
  x = t+c;
  if(x<c){
        x++;
```

```
        c++;
    }
    return(Q[i] = 0xfffffffe-x);
}
```

1.2 Losowanie liczb z zadanym rozkładem prawdopodobieństwa

1.2.1 Ogólne metody generowania liczb pseudolosowych z zadanym rozkładem prawdopodobieństwa

W tej części zostaną omówione dwie metody generowania liczb pseudolosowych z zadanym rozkładem prawdopodobieństwa. Są to metoda odwrócenia dystrybuanty i metoda eliminacji [8].

1.2.2 Metoda eliminacji

Jest to uniwersalna metoda pozwalająca na generowanie liczb, dla których znamy funkcję gęstości f. Zakładając, że potrafimy generować liczby losowe z rozkładem o gęstości $y = g(x)$ dla $x \in [x_{min}, x_{max}]$ oraz że dla tego przedziału spełniona jest nierówność $0 \leq f \leq g$ liczby z rozkładu $f(x)$ generujemy stosując następujący algorytm:

```
float DrawFRandomByElimination(float x_min, float x_max)
{
        do {
                float x = DrawGRandom(x_min, x_max);
                float u = DrawUniformRandom01();
        } while (u > f(x)/g(x))

        return x;
}
```

gdzie DrawGRandom() to funkcja generująca liczby zgodnie z rozkładem $g(x)$, a DrawUniformRandom01() to funkcja generująca liczby zgodnie z rozkładem $f(x)$.

1.2.3 Metoda odwrócenia dystrybuanty

Jest to metoda pozwalająca precyzyjnie przekształcić liczby wylosowane z rozkładem jednorodnym U na liczby z dowolnego rozkładu o gęstości f, dla

którego potrafimy obliczyć funkcję odwrotną F^{-1} do dystrybuanty F tego rozkładu. Stosujemy tutaj następujący algorytm:

```
float DrawFRandomByInverseCDF()
{
        float y = DrawUniformRandom01();
        return CalculateInverseCDF(y);
}
```

gdzie CalculateInverseCDF() oznacza funkcję F^{-1}.

2 Przykłady generowania liczb losowych w pakiecie SABAT

2.1 Generowanie liczb z zadanych rozkładów

2.1.1 Rozkład wykładniczy

Rozkład ten opisuje na przykład czas pomiędzy zdarzeniami, występującymi niezależnie ze średnią częstotliwością λ [9]. Opisany jest następującą funkcją gęstości:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{4}$$

Do generowania liczb z tym rozkładem zostanie użyta metoda odwrócenia dystrybuanty. W pakiecie SABAT jest on wykorzystywany głównie do symulowania miejsca reakcji cząstki w materiale.

Implementacja algorytmu w języku C++:

```
float DrawExponential(float lambda)
{
  float x = DrawUniformRandom01();
  return std::log(1.0f - x) / -lambda;
}
```

2.1.2 Rozkład oparty o wielomiany Legendre'a

Wielomiany te można opisać wzorem Rodriguez'a [10]:

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n \quad (n = 0, 1, \ldots) \tag{5}$$

Dla $x \in [-1,1] \implies x = \cos\theta$, gdzie $P_n(\cos\theta)$ są związane z grupą obrotów (poprzez funkcje kuliste). W symulacji SABAT są one wykorzystywane do generowania kierunku ruchu cząstki po reakcji w materiale. Wielomiany te można przedstawić także w postaci:

$$f(x) = \begin{cases} a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0, & -1 \leq x \leq 1 \\ 0, & |x| > 1 \end{cases} \tag{6}$$

W ten sposób można łatwo zastosować metodę eliminacji do generowania kąta wektora pędu cząstki po reakcji.

Implementacja algorytmu w języku C++ [11]:

```
float DrawLegendre(
        float * polynomial_coeffs,
        int polynomial_size,
        float max_value
)
{
  float x, y, f, powerx;
  do
  {
        x = 2.f * DrawUniformRandom01() - 1.f;
        y = DrawUniformRandom01() * max_value;
        f = polynomial_coeffs[0];
        powerx = 1;
        for(int i = 1; i < polynomial_size; ++i)
        {
                powerx *= x;
                f += polynomial_coeffs[i] * powerx;
        }
  } while(y > f);

  return x;
}
```

2.2 Losowanie punktów rozmieszczonych jednorodnie na sferze

2.2.1 Wprowadzenie

Do przeprowadzenia symulacji w pakiecie SABAT wymagany jest także generator jednorodnego rozkładu punktów na sferze w trójwymiarowej przestrzeni. Jest wiele sposobów na napisanie takiego generatora – w poniższym materiale zostanie przybliżone kilka z nich [12].

2.2.2 Metoda 1: jednorodne losowanie trzech współrzędnych

Pierwsza przedstawiona metoda, polega na losowaniu jednorodnym trzech współrzędnych $x, y, z \in [-1, 1]$ i użycia ich do konstrukcji punktów wewnątrz sześcianu. Następnie punkty, które nie leżą wewnątrz kuli o promieniu $R = 1$ (znajdującej się wewnątrz sześcianu) odrzucamy i do skutku losujemy współrzędne ponownie. Ostatnim krokiem jest rzutowanie punktu na sferę.

Implementacja algorytmu w języku C++:

```
Point Method1()
{
  double x,y,z,r;
  do
  {
    x = DrawUniformRandom01() * 2.0 - 1.0;
    y = DrawUniformRandom01() * 2.0 - 1.0;
    z = DrawUniformRandom01() * 2.0 - 1.0;
    r = sqrt(x*x + y*y + z*z);
  } while (r > 1);

  x = x/r;
  y = y/r;
  z = z/r;
  return Point(x,y,z);
}
```

2.2.3 Metoda 2: losowanie amplitudy biegunowej i współrzędnej „z”

Metoda korzystająca z funkcji trygonometrycznych polegająca na wylosowaniu jednorodnie dwóch wartości: $z \in [-1, 1]$ i $t \in [0, 2\pi)$. Następnie x i y

obliczamy wykonując następujące działania:

$$r = \sqrt{1 - z^2} \tag{7}$$
$$x = r \cos t \tag{8}$$
$$y = r \sin t \tag{9}$$

Tak otrzymane punkty x, y i z wykorzystujemy do utworzenia punktu leżącego na sferze o promieniu $R = 1$.

Implementacja algorytmu w języku C++:

```
Point Method2()
{
  double t = DrawUniformRandom01() * M_PI * 2.0;
  double z = DrawUniformRandom01() * 2.0 - 1.0;
  double r = sqrt(1-z*z);
  double x = r * cos(t);
  double y = r * sin(t);
  return Point(x,y,z);
}
```

gdzie M_PI to stała umieszczona w bibliotece <cmath> przechowująca wartość liczby π

2.2.4 Metoda 3: rzutowanie punktów w kole na sferę

Wariacja metody powyższej polega na wylosowaniu jednorodnie dwóch wartości $u, v \in [-1, 1]$. Następnie obliczamy:

$$s = u^2 + v^2 \tag{10}$$

Następnie sprawdzamy czy $s > 1$, jeżeli tak, to musimy powtórzyć operację losowania u i v, a jeżeli nie kontynuujemy obliczenia:

$$a = 2\sqrt{1 - s} \tag{11}$$
$$(x, y, z\ ) = (au, av, 2s - 1) \tag{12}$$

Tak otrzymane współrzędne x, y i z definiują punkt na sferze o promieniu $R = 1$.

Implementacja algorytmu w języku C++:

```
Point Method3()
{
  double u,v,s;
  do
  {
    u = DrawUniformRandom01() * 2.0 - 1.0;
    v = DrawUniformRandom01() * 2.0 - 1.0;
    s = u*u + v*v;
  } while (s > 1);
  double a = 2 * sqrt(1-s);
  return Point(a*u,a*v,2*s-1);
}
```

2.2.5 Metoda 4: losowanie współrzędnych sferycznych

Jest to metoda polegająca na wylosowaniu jednorodnie wartości kąta ϕ i cosinusa kąta θ:

$$\phi \in [0, 2\pi),$$
$$\cos\theta \in [-1, 1].$$

Następnie obliczamy sinus kąta θ:

$$\sin\theta = \sqrt{1 - \cos^2\theta}. \tag{13}$$

Współrzędne punktu na sferze o promieniu $R = 1$ obliczamy za pomocą układu równań:

$$(x, y, z) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta). \tag{14}$$

Implementacja algorytmu w języku C++:

```
Point Method4()
{
  double phi = DrawUniformRandom01() * M_PI * 2.0;
  double costh = DrawUniformRandom01() * 2.0 - 1.0;
  double sinth = sqrt(1.0 - costh*costh);
  double x = sinth * cos(phi);
  double y = sinth * sin(phi);
  double z = costh;
  return Point(x, y, z);
}
```

3 Opis pakietu SABAT

3.1 Wprowadzenie

Pakiet służy do symulowania emisji neutronów, ich oddziaływania w zadanym materiale oraz rejestrowania kwantów gamma. Ta praca zajmuje się tylko niektórymi elementami oprogramowania, które są używane od momentu uruchomienia symulacji aż do otrzymania wyników.

3.2 Opis symulacji

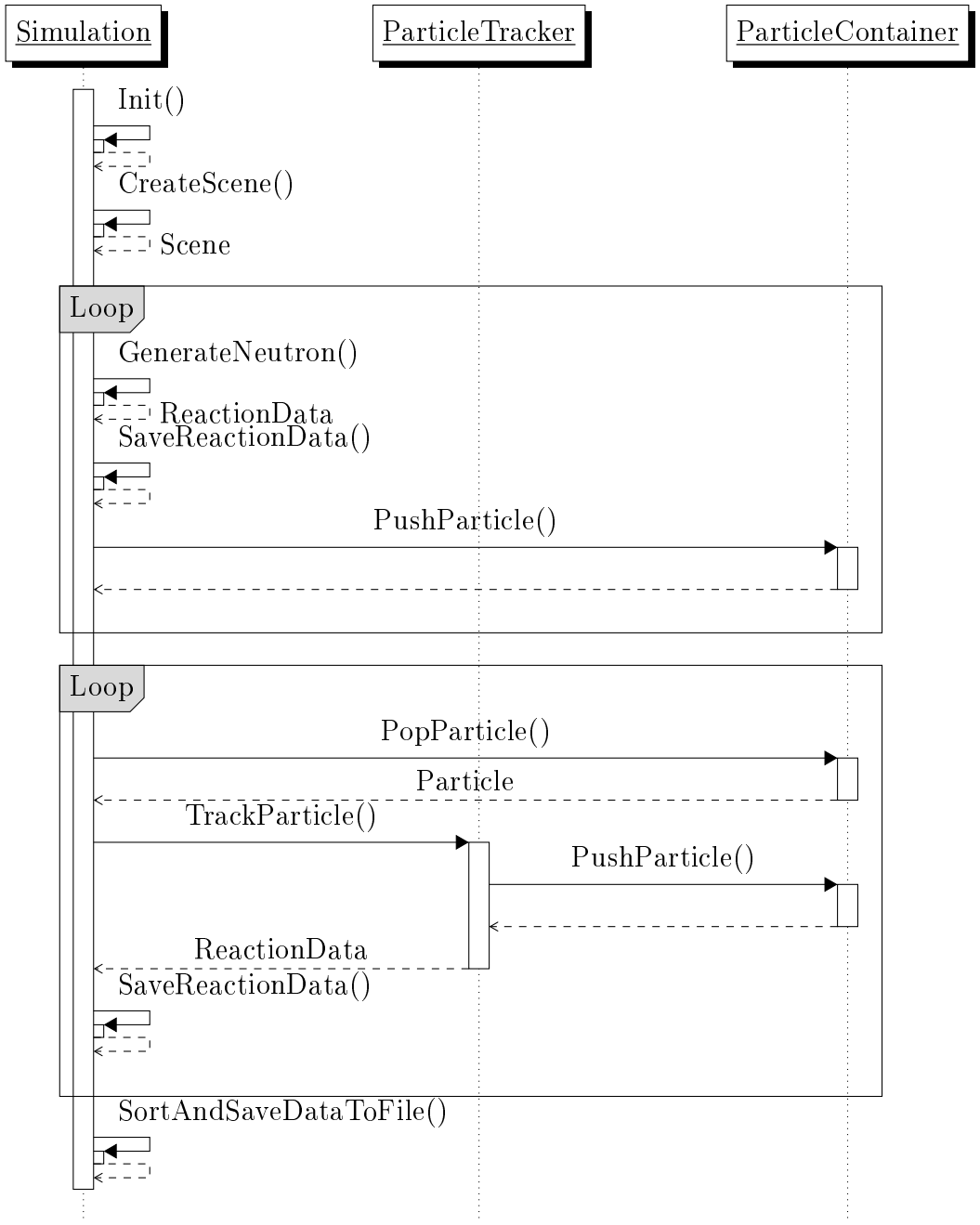
3.2.1 Parametry początkowe

Inicjalizacja symulacji polega na ustawieniu liczby wiązek, które opuszczają źródło promieniowania i liczby neutronów w każdej z wiązek. Ponadto należy podać opis sceny symulacji, czyli kształt i położenie obiektów na scenie oraz substancje z jakich składają się wspomniane obiekty. Program wymaga także dostępu do bazy danych, która zawiera informację o przekrojach czynnych i składzie pierwiastkowym substancji używanych w programie. Domyślna baza danych jest załączona razem z pakietem symulacyjnym.

3.2.2 Podstawowy algorytm działania symulacji

Symulacja rozpoczyna się inicjalizacją parametrów początkowych i utworzeniem sceny, tj. wszystkich obiektów w przestrzeni trójwymiarowej oraz zapisanie informacji o ich składzie chemicznym. Następnie program tworzy zadaną liczbę neutronów, zapisuje informację o powstaniu cząstki do pliku i umieszcza ją w kontenerze cząstek. Po zakończeniu tworzenia neutronów program przechodzi do głównej pętli algorytmu. Na początku pobierana jest pierwsza cząstka z kontenera. Program losuje liczbę z rozkładu wykładniczego z odpowiednio dobranym parametrem lambda w zależności od materiału obiektu, w którym cząstka aktualnie się znajduje. Jeśli liczba ta jest mniejsza od maksymalnej drogi swobodnej cząstki w aktualnym obiekcie dochodzi do reakcji, w przeciwnym przypadku cząstka opuszcza aktualny obiekt w miejscu zależnym od jej wektora pędu i po opuszczeniu obiektu zostaje przeniesiona z powrotem do kontenera. Jeśli jednak dojdzie do reakcji w materiale program losuje jej rodzaj w zależności od rodzaju cząstki (aktualna wersja programu rozróżnia dwa rodzaje cząstek - neutrony i kwanty gamma). Nowy wektor pędu w wyniku reakcji zostaje wylosowany z rozkładu opartego o wielomiany Legendre'a. Informacja o reakcji zostaje zapisana do pliku, a następnie cząstka zostaje dodana z powrotem do kontenera. Cząstki mogą zniknąć w wyniku reakcji, opuszczenia badanego układu lub osiągnięcia

zaniedbywalnie małej energii. Gdy kontener jest pusty pętla kończy pracę. Program zbiera dane zapisane w pliku, sortuje je i zapisuje w odpowiednim formacie. Algorytm w formie diagramu przedstawiony jest na rysunku 1.



Rysunek 1: Diagram sekwencji symulacji.

3.2.3 Format danych wyjściowych

Symulacja zapisuje dane o wszystkich reakcjach cząstek do plików tekstowych z dokładnymi danymi miejsca reakcji, materiału w jakim zareagowała, kierunku, w którym się porusza, liczby cząstek jakie w wyniku reakcji powstały, itp. Zapis następuje w trakcie działania programu, co pozwala na odzyskanie danych w przypadku przerwania symulacji.

4 Przykład zastosowania pakietu symulacyjnego SABAT

4.1 Opis projektu wykrywacza materiałów niebezpiecznych w wodzie

Projekt ma za zadanie zbadać podstawność wykorzystania analizy stechiometrycznej za pomocą generatora neutronów i detektora w celu wykrycia materiałów niebezpiecznych takich jak miny morskie, zbiorniki z iperytem itp. Wykrywanie materiałów niebezpiecznych w wodzie jest trudne ze względu na wysokie prawdopodobieństwo reakcji neutronów ze składnikami wody - wodorem i tlenem. W celu zredukowania tła pochodzącego od reakcji neutronów w wodzie opracowana została nowa koncepcja urządzenia, w którym zastosowano rury wypełnione powietrzem do prowadzenia zarówno neutronów, jak i kwantów gamma [13]. Pierwszym celem symulacji jest przetestowanie wykorzystania rury wypełnionej powietrzem. Kolejnym krokiem jest zastosowanie analogicznego rozwiązania dla detektora. Rura podłączona do detektora jest skierowana w badane podłoże mogące zawierać materiał niebezpieczny. Oba kroki mają prowadzić do wydłużenia drogi swobodnej neutronów i kwantów gamma, aby mogły dotrzeć do badanego materiału mając odpowiednio wysoką energię.

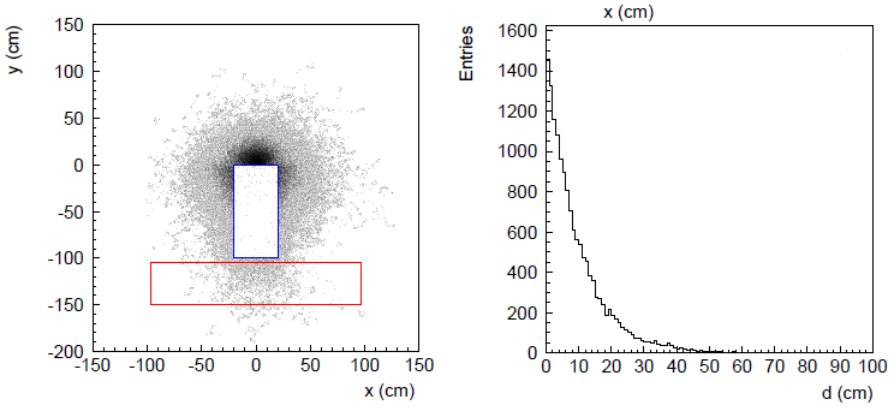
4.2 Prezentacja przykładowego wyniku

Przestrzeń symulacji, wypełniona wodą morską, zawiera trzy obiekty: punktowe źródło promieniowania w środku układu, prowadnicę w kształcie prostopadłościanu wypełnioną powietrzem oraz prostopadłościan wypełniony gazem musztardowym. Źródło promieniowania oraz zbiornik z iperytem znajdują się na przeciwległych końcach prowadnicy. Symulację przeprowadzono dla 10^5 neutronów.

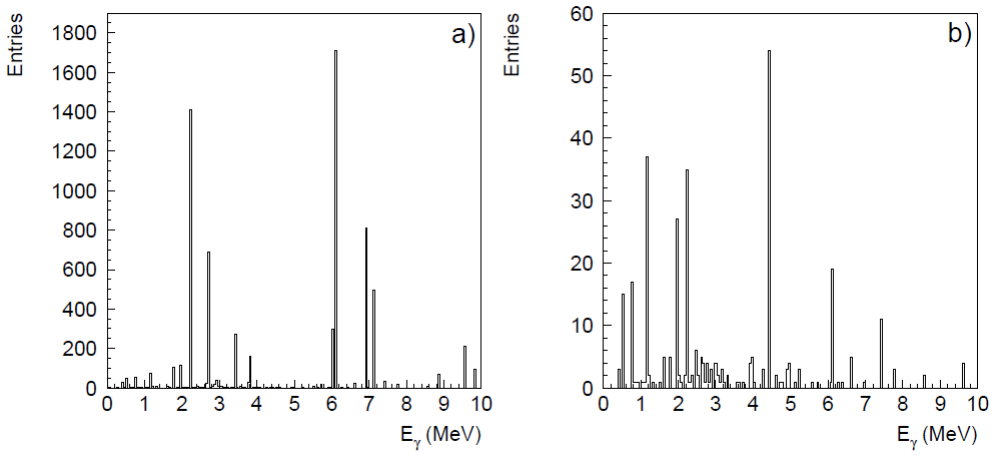
Na rysunkach 2 i 3 znajduje się graficzna reprezentacja danych wynikowych. Rysunki zostały zaczerpnięte z pracy "Project of the underwater system for chemical threat detection" [14, 15].

Jak łatwo można zauważyć na rysunku 2 woda powoduje bardzo dużo reakcji neutronów. Neutrony tracą energię, zmieniają kierunek ruchu i tym samym, w większości, nie są w stanie dotrzeć do materiału niebezpiecznego, aby wywołać w nim reakcję. Z tego powodu potrzebna jest prowadnica wypełniona np. powietrzem, dzięki której większa liczba neutronów będzie mo-

gła dotrzeć do badanego materiału. Na rysunku 2, rzut na płaszczyznę XY pokazuje, że prowadnica (prostokąt na środku wykresu) powoduje wydłużenie drogi neutronów i tym samym pozwala im dotrzeć do badanego materiału (prostokąt w dolnej części wykresu) i wywołać w nim reakcje.

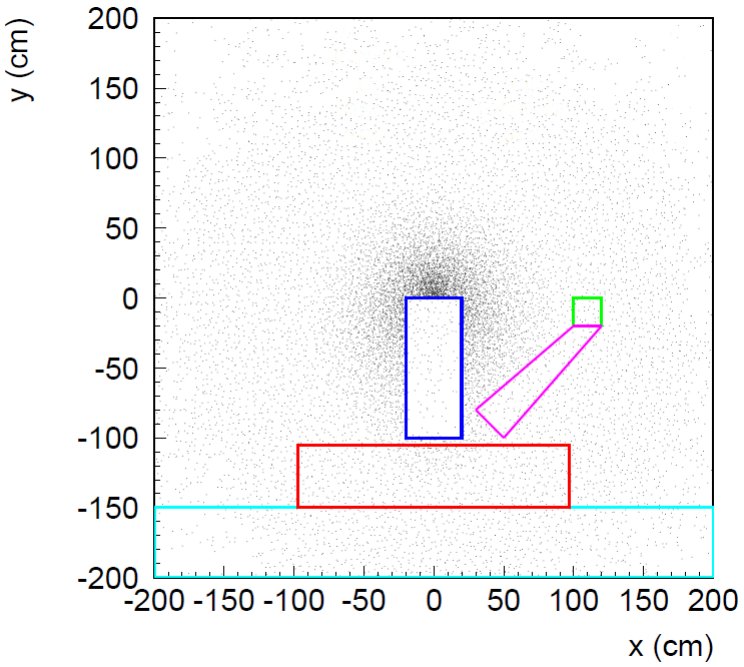


Rysunek 2: Po lewej znajduje się rzut na płaszczyznę XY miejsc reakcji neutronów. Po prawej przedstawiony jest histogram dystansu miejsca pierwszej reakcji neutronów od źródła promieniowania (dla $y \geq 0$ cm).



Rysunek 3: Histogram energii kwantów gamma pochodzących z reakcji neutronów w wodzie (a) i w gazie musztardowym (b).

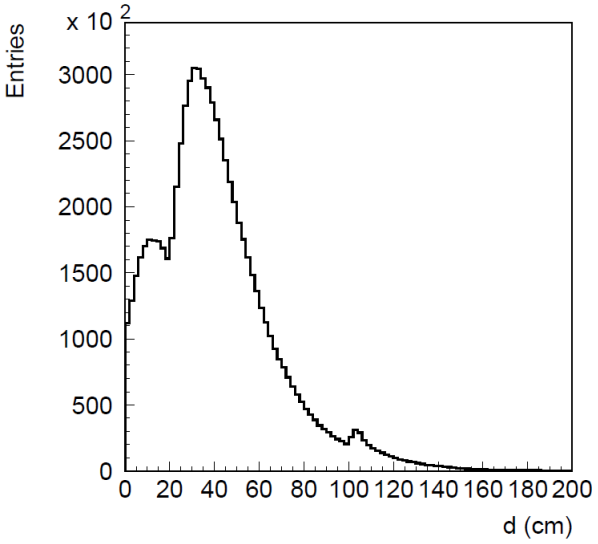
Analiza histogramów energii kwantów gamma z rysunku 3 pozwala zauważyć duże różnice pomiędzy kwantami gamma pochodzącymi z reakcji w wodzie, a tymi które powstały w wyniku reakcji w gazie musztardowym. Przykładem może być duża liczba kwantów o energii ≈4,5 MeV, w przypadku gazu musztardowego, pochodzącymi z reakcji neutronów z atomami węgla.



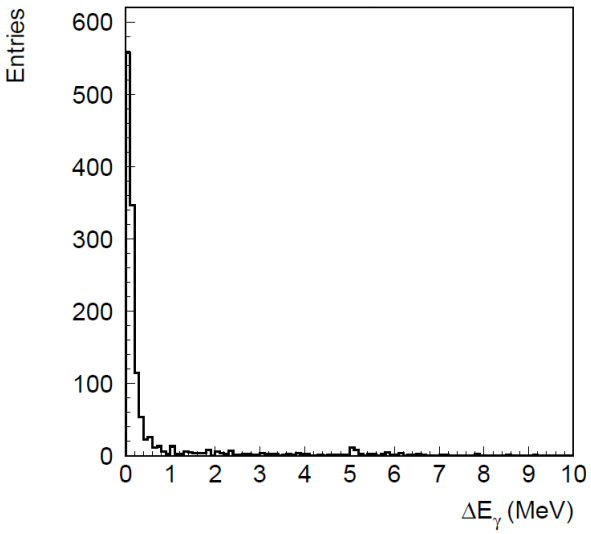
Rysunek 4: Rzut na płaszczyznę XY miejsc reakcji kwantów gamma.

W kolejnym kroku do przestrzeni dodano dno z piasku, detektor oraz prowadnicę detektora przedstawione na rysunku 4. Analizując rysunek 5 można łatwo zauważyć zgodność drogi swobodnej neutronów z ustawieniem obiektów w przestrzeni symulacji. W szczególności małe maksimum lokalne dla $d \approx 100$ cm jest spowodowane reakcją neutronów, które dotarły do końca rury i zareagowały w iperycie lub małej przestrzeni pomiędzy gazem musztardowym a prowadnicą.

Rysunek 6 przedstawia straty energii kwantów gamma w detektorze. Pomijając duża liczbę sygnałów dla energii do 1 MeV można zauważyć bardzo interesującą grupę dużych strat energii wokół wartości 5 MeV.



Rysunek 5: Histogram drogi swobodnej neutronów.



Rysunek 6: Histogram strat energii kwantów gamma pochodzących z reakcji w detektorze.

Zaprezentowane wyniki są tylko częścią otrzymanych wyników końcowych. Całość wymaga głębszej analizy i interpretacji, które nie zostaną podjęte w tej pracy ze względu na jej informatyczny charakter.

5 Testowanie metod generowania liczb losowych

5.1 Test generatora CMWC

Testowanie wybranego generatora liczb losowych z rozkładu jednorodnego zostało wykonane poprzez sprawdzenie kilku podstawowych właściwości generatora liczb losowych, opisanych w książce „Beautiful Testing" [16] oraz porównanie wyników testów dostępnych w pakiecie Dieharder [17]. Do testów wybrano, wspomniany wcześniej, generator CMWC, generatory rand i rand48 z biblioteki stdlib.h oraz generatory mt19937, ranlux48 znajdujących się w bibliotece standardowej języka C++ [22] według standardu C++11 [7].

Pierwsza właściwość, jaka została sprawdzona to przedział liczb x wylosowanych przez generator, $x \in [0, 1]$. Test ten nie dopuszcza błędu, tj. żadna wartość nie może pochodzić spoza zadanego przedziału. Wszystkie inne przedstawione tu testy dopuszczają pewną granicę błędu, której przekroczenie powoduje negatywny wynik testu. Następny test polegał na sprawdzeniu zgodności wartości średniej i wariancji z ich odpowiednikami teoretycznymi. W tym przypadku dopuszczalny błąd został wybrany arbitralnie i wynosił 1% wartości teoretycznej. Test ten jest jedynie wprowadzeniem do kolejnego testu, stąd arbitralność dopuszczalnego błędu. Test zgodności histogramu polega na wykonaniu losowania dużej liczby wartości zmiennej losowej, a następnie porównania histogramu z rozkładem tych wartości ze średnimi wartościami funkcji gęstości prawdopodobieństwa. Wynik porównania zostaje wykorzystany w teście zgodności χ^2, który został opisany w [16].

Każdy ze wspomnianych wcześniej generatorów pomyślnie przeszedł powyższe testy.

Kolejnym krokiem było wykonanie porównania za pomocą testów z pakietu Dieharder. Składa się on na 117 wariantów testów. Część testów jest w całości oryginalna, a reszta testów została zaczerpnięta z innych pakietów: Diehard [18] i STS [19], ale jest rozbudowana przez autora pakietu. Testy pobierają określoną liczbę wyników i szukają pewnych regularności w ciągach danych. Przykładami takich testów jest test bazujący na prawdopodobieństwie znalezienia dwóch osób w pewnej grupie ludzi, które mają taki sam dzień i miesiąc urodzin oraz testy prostych regularności w danych w różnym formacie (np. wielowymiarowe macierze bitów czy łańcuchy DNA). Pakiet zawiera także test Kołmogorowa-Smirnowa, który będzie opisany w kolejnym rozdziale dotyczącym testowania pozostałych generatorów. Wyniki ilościowe dla poszczególnych generatorów przedstawia tabela 1.

rand	rand48	mt19937	ranlux48	CMWC
62%	97%	100%	100%	100%

Tablica 1: Stabelaryzowane wyniki testowania generatorów jako procent zaliczonych testów.

Generator CMWC osiąga zadowalające wyniki w powyższych testach. Ponadto jest on wystarczająco szybki w stosunku do swoich kontrkandydatów, a w szczególności tych, które zaliczyły wszystkie poprzednie testy. Można to łatwo zauważyć analizując wyniki porównania szybkości generowania liczb zamieszczone w tabeli 2, które zostały otrzymane za pomocą omówionego wcześniej pakietu Dieharder. Z tego powodu generator ten został zastosowany w pakiecie symulacyjnym.

rand	rand48	mt19937	ranlux48	CMWC
242	271	35.5	6.35	125

Tablica 2: Stabelaryzowane wyniki porównania szybkości generatorów jako liczba wyników na sekundę $[10^5/s]$.

5.2 Test pozostałych generatorów

Generator liczb z rozkładu wykładniczego oraz generator liczb z rozkładów sparametryzowanych za pomocą wielomianów Legendre'a zostały przetestowane analogicznymi sposobami co wcześniej za wyjątkiem testu zgodności histogramu. Brak tego testu jest uzasadniony jego zastępowalnością przez test Kołmogorowa-Smirnowa. W przypadku obu genaratorów można łatwo policzyć dystrybuantę z ich funkcji gęstości. Dodatkowo pozwala to na uniknięcie problemu dotyczącego dziedziny funkcji gęstości rozkładu wykładniczego określonej jako przedział nieskończony $[0, \infty)$. Test Kołmogorowa-Smirnowa polega na porównaniu dystrybuanty otrzymanej z pewnej dużej liczby wyników losowania $F_n(x)$ z dystrybuantą teoretyczną rozkładu $F(x)$. Obliczane są tutaj dwie wartości:

$$K^+ = \sqrt{n} \max_{\infty}(F_n(x) - F(x)) \tag{15}$$

$$K^- = \sqrt{n} \max_{\infty}(F(x) - F_n(x)) \tag{16}$$

Wartości współczynnika K powinny znaleźć się w przedziale $[0.07089, 1.5174]$ w przypadku około 98% prób testowych, dla próbek o liczności większej od

1000 [16]. Autor jednak zdecydował się na obniżenie limitu zaliczenia testu do 95% prób testowych z powodu możliwości wystąpienia dużego błędu dla wartości bliskich zeru w przypadku dużych wartości λ. W przypadku wielomianów Legendre'a również występują wartości bliskie zeru co przekłada się na małą liczbę próbek w tych przedziałach, a tym samym dużych wartości K^+ i K^-. Oba generatory przetestowano wykonując 10^3 prób testowych na zestawach próbek o liczności 10^4.

Zaproponowane generatory przechodzą wszystkie testy. Generator liczb z rozkładu wykładniczego przetestowano dla wartości λ ze zbioru:

$$\lambda \in \{0.2, 0.5, 1, 2, 3, 10, 1000, 100000\}$$

Natomiast generator oparty o wielomiany Legendre'a przetestowano w dwóch przypadkach wielomianów:

$$\begin{aligned} f_1(x) =& -2.09444 \times 10^{-15}x^6 - 3.36007 \times 10^{-16}x^5 + 3.30407 \times 10^{-15}x^4 \\ &+ 5.60012 \times 10^{-16}x^3 - 1.12802 \times 10^{-15}x^2 - 2.42319 \times 10^{-14}x + 0.5 \\ f_2(x) =& 0.0195364x^6 - 0.0092849x^5 - 0.661385x^4 - 0.108053x^3 \\ &+ 1.55978x^2 + 0.0265623x + 0.109558 \end{aligned}$$

5.3 Testy wydajnościowe metod losowania punktów jednorodnie na sferze

Testy przeprowadzono generując próbki o liczności $N = 10^8$ dla każdej metody oraz mierząc dla każdej z nich czas generacji w milisekundach. Następnie otrzymany pomiar czasu generacji zadanej próbki Δt wykorzystano do obliczenia liczby zdarzeń wygenerowanych w ciągu sekundy $N_{1s}[10^6/s]$:

$$N_{1s} = \frac{N}{1000\Delta t}$$

Najszybsza okazała się metoda nr 3, której szybkość polega na sprowadzeniu problemu trójwymiarowego do dwóch wymiarów - losowanie punktu w kole - a następnie zrzutowaniu tego punktu na sferę. Wyniki porównania przedstawia tabela 3.

Nr metody	1	2	3	4
Wynik	2.50	4.94	5.21	5.10

Tablica 3: Stabelaryzowane wyniki porównania szybkości metod jako liczba wyników na sekundę $[10^6/s]$.

6 Porównanie implementacji sekwencyjnej i równoległej

6.1 Wykorzystane narzędzia

Kod programu do symulacji został napisany w języku C++ [22] i jest zgodny ze standardem C++11 [7]. Do kompilacji użyto kompilatora z kolekcji GCC [23]. Wersja równoległa została zaimplementowana za pomocą biblioteki Open MPI [20], opartej o standard interfejsu komunikacji MPI [21]. Porównanie zostało przeprowadzone na komputerze z systemem Ubuntu w wersji 14.04 [24].

6.2 Ustawienie początkowe generatora liczb pseudolosowych

W przypadku generowania liczb pseudolosowych w wersji sekwencyjnej wystarczy jeden stan generatora, który ustawiamy na początku programu. Natomiast w przypadku generatora w wersji równoległej, aby uniknąć błędu generowania identycznej sekwencji liczb dla każdego z procesów, w każdym z nich musimy utworzyć osobny stan generatora i zainicjować go innym parametrem. W ten sposób otrzymamy różne ciągi liczb pseudolosowych.

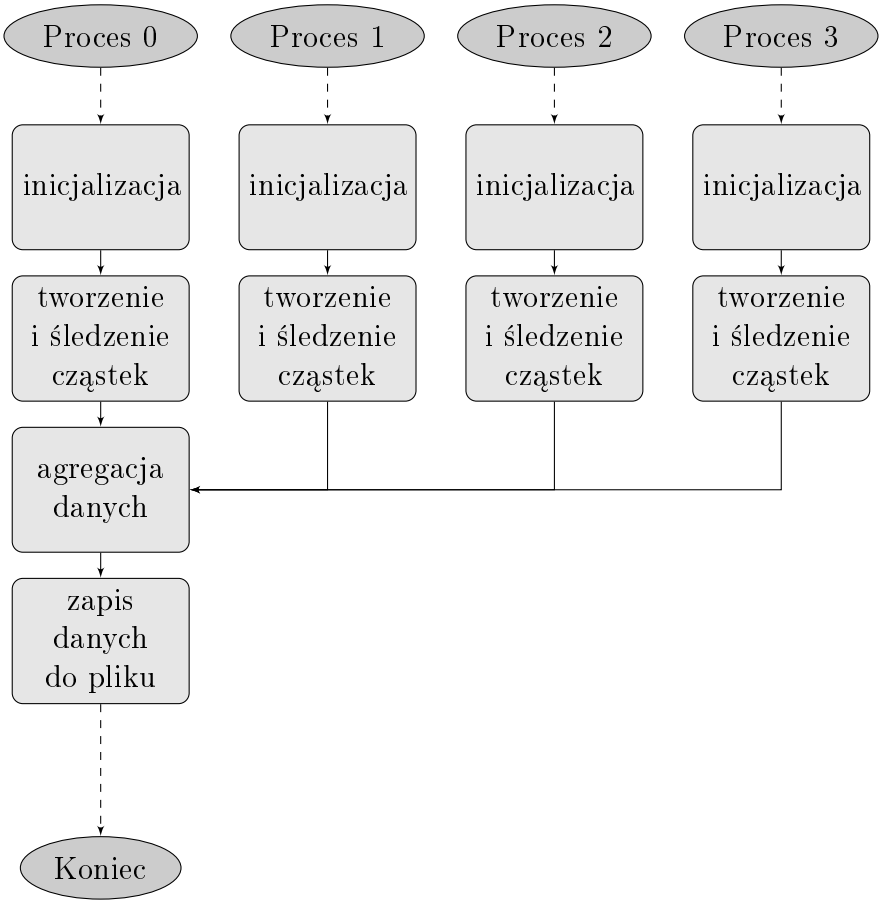
6.3 Struktura działania wersji równoległej

Algorytm dla wersji równoległej opiera się na koncepcji podstawowego algorytmu, w którym cząstki poruszają się w układzie niezależnie od siebie. Z tego powodu najprostszym i najwydajniejszym rozwiązaniem wydaje się być uruchomienie osobnej symulacji na każdym z dostępnych procesorów, a następnie dokonanie agregacji wyników z poszczególnych symulacji do jednego pliku wynikowego.

Algorytm rozpoczyna się uruchomieniem wybranej liczby procesów. Na każdym z nich następuje kolejno inicjalizacja parametrów początkowych, tworzenie i śledzenie cząstek. Po zakończeniu głównej pętli programu proces

o identyfikatorze 0 rozpoczyna zbieranie danych od innych procesów. W tym samym czasie reszta procesów przesyła wszystkie dane do procesu głównego, a po zakończeniu przesyłania - kończą pracę. Proces główny po zebraniu wszystkich danych zapisuje je do pliku wynikowego i kończy pracę.

Diagram działania programu w wersji równoległej został przedstawiony na rysunku 7.



Rysunek 7: Diagram działania programu w wersji równoległej.

6.4 Wyniki porównania

Porównanie przeprowadzono na komputerze wyposażonym w czterordzeniowy procesor z systemem Ubuntu 14.04 [24]. Wynikiem pierwszego etapu

porównania jest dobranie optymalnej liczby procesów. Program można uruchomić dla większej liczby procesów niż liczba rdzeni w procesorze - zarządzaniem procesami w trakcie ich pracy zajmuje się system operacyjny. Pozwala to na lepsze wykorzystanie procesora i tym samym krótszy czas oczekiwania na wyniki symulacji.

Drugi etap polega na analizie wykresu czasu pracy programu, dla optymalnej liczby procesów, jako funkcji liczby neutronów na początku symulacji. Wykres takiej funkcji umożliwi wykrycie ewentualnych spadków wydajności zaprezentowanego rozwiązania przy dużej liczbie danych początkowych.

6.4.1 Pierwszy etap

Wyniki porównania przedstawia tabela 1. Ostatnia kolumna zawiera osiągnięte przyspieszenie $\psi_c(p)$, które zostało obliczone za pomocą wzoru:

$$\psi_c(p) = \frac{t_1}{t_p}$$

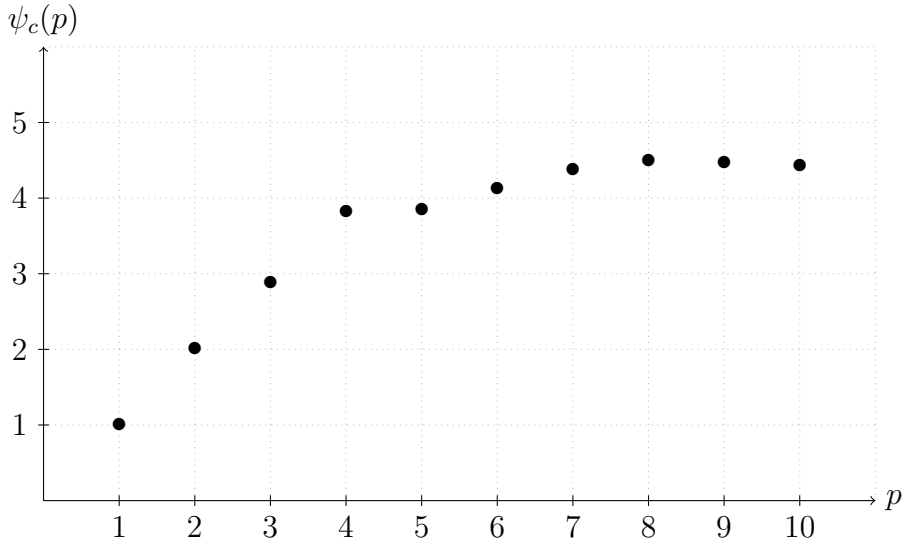
W powyższym wzorze parametr p to liczba procesów wykonywanych równolegle, a t_i to zmierzony czas pracy programu.

p	t[s]	$\psi_c(p)$
1	2042	1
2	1021	2
3	708	2.88
4	534	3.82
5	532	3.84
6	496	4.12
7	467	4.37
8	455	4.49
9	457	4.47
10	460	4.43

Tablica 4: Wartości czasu pracy programu t uzyskane dla p procesów wykonywanych równolegle na komputerze z czterordzeniowym procesorem.

Analiza wykresu przyspieszenia $\psi_c(p)$ na rysunku 5 pozwala zauważyć dwie ważne właściwości: liniowy wzrost funkcji dla $p \in \{1, 2, 3, 4\}$, a następnie spowolnienie wzrostu i zatrzymanie na stałym poziomie od $p = 8$. Jest to

zrozumiałe - pierwsza właściwość wynika z przypisania każdego procesu do osobnego rdzenia i pełną równoległą pracę, a druga wynika z działania planisty krótkoterminowego (*def.* program systemowy, zajmuje się wybieraniem procesów gotowych do wykonania i przydzielaniem im procesora[25]). Niewyjaśniony pozostaje brak zmiany wartości przyspieszenia przy zmianie czterech procesów na pięć. Wyniki dla wybranego rozwiązania są zadowalające.



Rysunek 8: Przyspieszenie jako dyskretna funkcja liczby procesów wykonywanych równolegle na komputerze z czterordzeniowym procesorem.

6.4.2 Drugi etap

W tym etapie przedstawione zostaną wyniki porównawcze dla czterech i ośmiu procesów wykonywanych równolegle jako funkcja liczby neutronów początkowych. Te dwa przypadki zostały wybrane ze względu na liczbę rdzeni dostępnych na komputerze testowym (cztery procesy) i maksymalne przyspieszenie jakie udało się osiągnąć (osiem procesów).

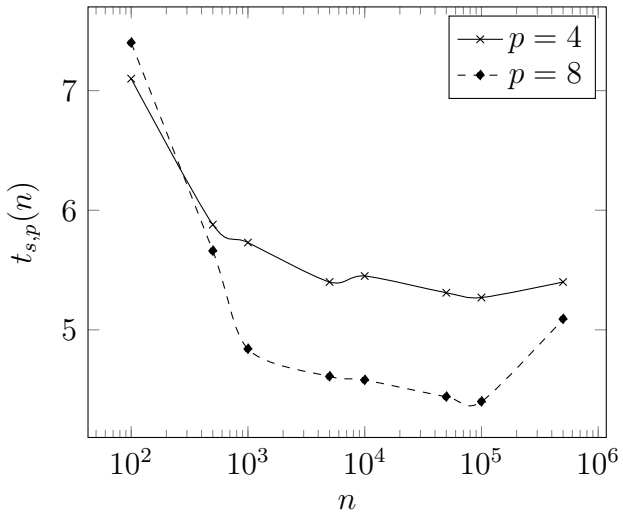
Aby porównać jakość osiąganych wyników czasowych, należy sprowadzić je do konkretnego współczynnika. Zaproponowany niżej współczynnik, za pomocą którego wykonane zostało porównanie to średni czas przypadający na obliczenie wszystkich reakcji dla stu neutronów:

$$t_{s,p}(n) = \frac{t_{c,p}(n)}{n}$$

W powyższym wzorze p oznacza liczbę procesów wykonywanych równolegle, n liczbę neutronów na początku symulacji, natomiast t_s i t_c oznaczają odpowiednio średni czas przypadający na obliczenia dla stu neutronów i całkowity czas działania programu.

n	$t_{c,p=4}$	$t_{c,p=8}$	$t_{s,p=4}$	$t_{s,p=8}$
100	7.1	7.4	7.10	7.40
500	29.4	28.3	5.88	5.66
1 000	57.3	48.4	5.73	4.84
5 000	269.8	230.2	5.40	4.61
10 000	544.5	458.0	5.45	4.58
50 000	2655.0	2220.1	5.31	4.44
100 000	5267.9	4397.4	5.27	4.40
500 000	27019.1	25454.1	5.40	5.09

Tablica 5: Porównanie całkowitego czasu $t_{c,p}$[s] i średniego czasu $t_{s,p}$[s/100 neutronów] uzyskanych dla zmiennej liczby neutronów.



Rysunek 9: Średni czas generowania wszystkich reakcji jako funkcja liczby neutronów początkowych $t_{s,p}(n)$[s/100 neutronów].

Wyniki porównania przedstawione w tabeli 2 i na rysunku 6 pokazują spodziewany spadek w przypadku zwiększania liczby danych zarówno dla czterech jak i ośmiu procesów. Dla czterech procesów następuje stabilizacja od $n = 5000$. W przypadku ośmiu procesów stabilizacja zostaje przerwana

spadkiem wydajności - wskazuje na to skok wartości t_s dla ostatniego pomiaru $n = 500000$.

6.4.3 Omówienie wyników porównania

Wyniki zrównoleglenia są zadowalające. W przypadku, gdy chcemy symulować liczbę neutronów $n \leq 5 \times 10^5$ możliwe jest uruchomienie programu dla większej liczba procesów niż liczba dostępnych rdzeni. Dodatkowe przyspieszenie jest uzyskane ze względu na działanie planisty - gdy jakiś proces oczekuje na pewne zdarzenie (np. komunikacja z bazą danych), planista przydziela rdzeń procesora do innego procesu. W przeciwnym przypadku (dla większych n) zalecane jest uruchomienie liczby procesów zbliżonej do, lub dokładnie odpowiadającej liczbie rdzeni.

Podsumowanie

Celem tej pracy było przedstawienie i uzasadnienie zastosowanych metod statystycznych oraz wykorzystania zrównoleglenia w celu przyspieszenia pracy programu do symulacji działania wykrywacza substancji niebezpiecznych. Omówiono kilka przypadków optymalizacji pracy programu SABAT służącego do symulacji reakcji neutronów w badanym materiale i rejestrowania kwantów gamma emitowanych z tego materiału. Głównym elementem jest wykorzystanie wielu procesorów do przeprowadzenia symulacji. Zrównoleglenie działania programu spowodowało 4-krotne zwiększenie szybkości działania pakietu symulacyjnego SABAT dla takiej samej liczby procesów uruchomionych na czterordzeniowym procesorze. Przedstawione zostały także wyniki porównawcze szybkości dla metod generowania punktów jednorodnie na sferze oraz generatorów liczb pseudolosowych z rozkładu jednorodnego. Praca zawiera także przybliżenie dwóch generatorów liczb losowych, które także występują w pakiecie symulacyjnym. Losują one liczby z rozkładu wykładniczego oraz rozkładu opartego o wielomiany Legendre'a.

Wszystkie zaprezentowane metody zostały przetestowane. Poza prostymi testami, krótko omówione zostały także test Kołmogorowa-Smirnowa oraz pakiet testowy Dieharder. Są one istotnymi testami, pozwalającymi sprawdzić prawidłowe działanie generatorów liczb pseudolosowych.

Praca ta przedstawia tylko część problemów optymalizacyjnych, które dotyczą pakietu SABAT. Bardzo istotne, a nieomówione pozostały kwestie implementacyjne komunikacji z bazą danych oraz przestrzeni symulacji, np. pobieranie informacji o przekrojach czynnych z bazy oraz wykrywanie kolejnych obiektów przez które przelatuje cząstka w przestrzeni symulacji. Te problemy pozostają do omówienia w przyszłych pracach uczestników eksperymentu SABAT.

Literatura

[1] P. Moskal, *Nuclear physics in medicine, minefield and kitchen*, Annales UMCS Physica 66 (2012) 71

[2] B.C. Maglich, *Birth of 'Atometry' – Particle Physics Applied To Saving Human Lives*, prezentacja z konferencji LEAP 2005, Bonn–Jülich, Germany, http://www.fz-juelich.de/leap05/talks

[3] Eckhert & Ziegler, *Industrial Radiation Sources - Product Information*, 2008, http://www.ezag.com/fileadmin/ezag/user-uploads/isotopes/isotopes/5_industrial_sources.pdf

[4] Strona encyklopedii PWN, http://encyklopedia.pwn.pl

[5] S.K. Park, K.W. Miller, *Random Number Generators: Good Ones Are Hard To Find*, Communications of the ACM: Vol. 31: Iss. 10, 1988, str. 1192–1201

[6] G. Marsaglia, *Random Number Generators*, Journal of Modern Applied Statistical Methods: Vol. 2: Iss. 1, Article 2, 2003

[7] Strona zawierająca informację o oficjalnych standardach języka C++, https://isocpp.org/std/the-standard

[8] R.J. Nowak, *Statystyka matematyczna*, Wydział Fizyki Uniwersytetu Warszawskiego, 1999, wyd. trzecie, str. 44-45, str. 52-53

[9] R.J Nowak, *Statystyka dla fizyków*, PWN, Warszawa 2002, ISBN 83-01-13702-9, str. 216-218

[10] G.M. Fichtenholz, *Rachunek różniczkowy i całkowy. Tom 3*, PWN, Warszawa 1999, ISBN 83-01-02177-2, str. 354

[11] Implementacja algorytmu wykonana przez Panią Dominikę Hunik

[12] E.W. Weisstein, *Sphere Point Picking*, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/SpherePointPicking.html

[13] M. Silarski, P. Moskal, *Zgłoszenie patentowe Nr P409388 (2014)*

[14] Wykresy zawarte w tym rozdziale zostały wykonane przez Dr Michała Silarskiego

[15] M. Silarski, D. Hunik, P. Moskal, M. Smolis, S. Tadeja, *Project of the underwater system for chemical threat detection*, Acta Physica Polonica A127 (2015) 1543

[16] T. Riley (red.), A. Goucher (red.), *Beautiful Testing: Leading Professionals Reveal How They Improve Software*, O'Reilly, Sebastopol 2009, ISBN 978-0-596-15981-8

[17] R.G. Brown, *Pakiet testów Dieharder*, http://www.phy.duke.edu/ rgb/General/dieharder.php

[18] G. Marsaglia, *Pakiet testów Diehard*, http://stat.fsu.edu/pub/diehard/

[19] NIST, *Statistical Test Suite*, http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html

[20] Strona projektu Open MPI, http://www.open-mpi.org/

[21] J.Q. Michael, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill Inc., 2004, ISBN 0-07-058201-7

[22] S. Meyers, *Skuteczny nowoczesny C++*, M. Chaniewska, APN PROMISE, Warszawa 2015, ISBN 978-83-7541-155-3

[23] Strona kolekcji kompilatorów GNU, https://gcc.gnu.org/

[24] Strona systemu operacyjnego Ubuntu, http://www.ubuntu.com/

[25] W. Płaczek, Materiały do przedmiotu Systemy Operacyjne, rok akademicki 2010/2011, WFAIS UJ