

Wrocław, dn. 10 czerwca 2015r.

POLITECHNIKA WROCŁAWSKA

APLIKACJE INTERNETOWE I ROZPROSZONE

---

# Obliczanie przybliżenia liczby $\pi$ .

DOKUMENTACJA KOŃCOWA

---

*Grupa projektowa:*

Paweł STERNIK, 200623

Kamil CICHUTA, ???

Mariusz CEBULA, ???

Sławomir SYGUT, ???

*Prowadzący:*

dr inż. Marek WODA

*Termin spotkań:*

Środa, godz. 07:30

# Spis treści

<b>1</b>	<b>Temat projektu.</b>	<b>2</b>
<b>2</b>	<b>Cel projektu.</b>	<b>2</b>
2.1	Cel dydaktyczny. . . . .	2
2.2	Cel merytoryczny. . . . .	2
<b>3</b>	<b>Opis zastosowanego algorytmu.</b>	<b>3</b>
3.1	Opis słowny algorytmu. . . . .	3
3.2	Obliczanie liczby Pi metodą Monte Carlo. . . . .	3
3.3	Implementacja algorytmu - program jednowątkowy. . . . .	5
<b>4</b>	<b>Zastosowane technologie.</b>	<b>7</b>
4.1	Framework Django. . . . .	7
4.1.1	Opis. . . . .	7
4.1.2	Instalacja Django. . . . .	8
4.1.3	Tworzenie nowego projektu. . . . .	8
4.1.4	Ustawienia bazy danych. . . . .	8
4.1.5	Tworzenie modeli. . . . .	9
4.2	Technologia MPI. . . . .	9
4.3	CSS i HTML. . . . .	11
4.4	Komunikacja grupy. . . . .	11
4.4.1	GitHub. . . . .	11
4.4.2	Trello. . . . .	13
<b>5</b>	<b>Plan realizacji.</b>	<b>14</b>
<b>6</b>	<b>Implementacja silnika obliczeniowego.</b>	<b>15</b>
6.1	Zrównoleglenie algorytmu . . . . .	15
<b>7</b>	<b>Aplikacja internetowa.</b>	<b>17</b>
<b>8</b>	<b>Testy.</b>	<b>17</b>
<b>9</b>	<b>Podsumowanie i wnioski.</b>	<b>17</b>

# 1 Temat projektu.

Tematem projektu realizowanego w ramach kursu Aplikacje Internetowe i Rozproszone było wyznaczanie rozszerzenia liczby Pi z wykorzystaniem algorytmu Monte Carlo. Realizacja wymagała implementacji kilku modułów stanowiących cały system.

## 2 Cel projektu.

### 2.1 Cel dydaktyczny.

Głównym celem dydaktycznym kursu było zapoznanie się z technologiami wykorzystywanymi do tworzenia rozproszonych aplikacji połączonych z internetowymi klientami. Ponadto kurs wymagał zoorganizowania pracy w grupach kilkusobowych co wymuszało zaplanowanie kolejnych etapów pracy i podział poszczególnych zadań między członków grupy. Wykorzystane zostały do tego odpowiednie narzędzia komunikacji, które zostaną opisane dokładniej w dalszej części dokumentu.

### 2.2 Cel merytoryczny.

Implementacja algorytmu Monte Carlo obliczającego rozszerzenie liczby Pi z wykorzystaniem technologii MPI (ang. Message Passing Interface ) czyli protokołu komunikacyjnego służącego do przesyłania komunikatów pomiędzy procesami programów równoległych. Ponadto stworzenie aplikacji internetowej która poprzez stworzoną bazę danych łączy się z silnikiem obliczeniowym działającym na kilku niezależnych komputerach. Strona powinna posiadać elementy zmieniające się dynamicznie podczas realizacji zadania - przykładowo pasek postępu.

## 3 Opis zastosowanego algorytmu.

### 3.1 Opis słowny algorytmu.

Metoda Monte Carlo stosowana jest do problemów, które jest bardzo trudno rozwiązać za pomocą podejścia analitycznego. Najczęściej stosuje się ją do modelowania złożonych problemów takich jak:

1. Obliczanie całek.
2. Obliczanie łańcuchów procesów statystycznych .
3. Obliczanie złożonych symulacji.

Metoda opiera się na losowaniu nazywanym w tym przypadku wyborem przypadkowym. Losowanie dokonywane jest zgodnie z rozkładem, który jest znany. Przykładowo całkowanie metodą Monte-Carlo działa na zasadzie porównywania losowych próbek z wartością funkcji. Dokładność wyniku uzyskanego tą metodą jest zależna od liczby sprawdzeń i jakości użytego generatora liczb pseudolosowych. Zwiększanie liczby prób nie zawsze zwiększa dokładność wyniku, ponieważ generator liczb pseudolosowych ma skończenie wiele liczb losowych w cyklu. Przykładowo całkowanie tą metodą jest używane w przypadkach, kiedy szybkość otrzymania wyniku jest ważniejsza od jego dokładności (np. obliczenia inżynierskie).

W projekcie algorytm Monte Carlo zostanie wykorzystany do obliczenia rozszerzenia dziesiętnego liczby Pi. Dokładny opis realizacji tego algorytmu znajdują się w następnym punkcie.

### 3.2 Obliczanie liczby Pi metodą Monte Carlo.

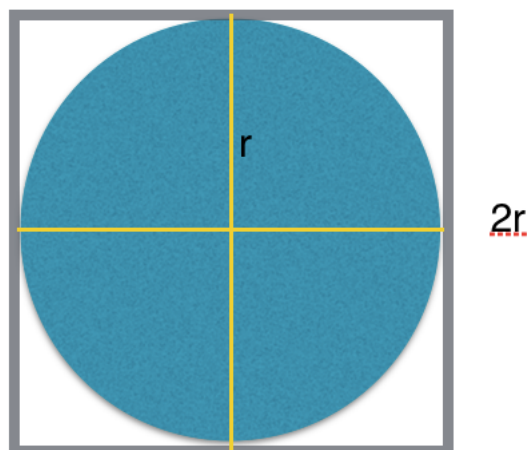
Liczbę Pi można obliczać na wiele różnych sposobów. Jednym z nich jest wykorzystanie metody Monte Carlo. Metoda ta charakteryzuje się przede wszystkim swoją stosunkowo prostą procedurą jak i przystosowaniem do zaimplementowania mechanizmów zrównoleglenia - co jest jednym z głównych celów projektu.

Pole kwadratu przedstawionego na Rysunku 1 wynosi:

$$(4\Pi)^2$$

natomiast koła oczywiście:

$$(\Pi)r^2$$



Rysunek 1: Koło o promieniu  $r$  wpisane w kwadrat - bok  $2r$ .

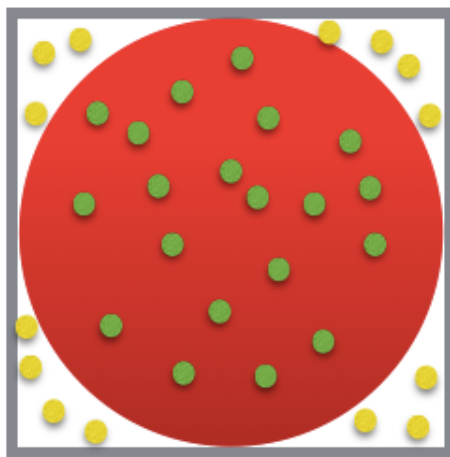
Znając zatem pole koła jak i kwadratu można zauważyć że stosunek pola koła do pola kwadratu wynosi:

$$\frac{PoleKola}{PoleKwadratu} = \frac{\Pi r^2}{4r^2} = \frac{\Pi}{4}$$

Następnie korzystając z tego, że pole koła jak i kwadratu zostały w jakiś sposób obliczone można wyciągnąć wzór na liczbę Pi:

$$\Pi = 4 \frac{PoleKola}{PoleKwadratu}$$

Dojście do tego punktu mogłoby się wydawać zatoczeniem koła i powrotem do punktu początkowego problemu. Tak nie jest. Ponieważ dopiero w tym miejscu pojawia się cała charakterystyka metody Monte Carlo. Abstrakcyjnie należy sobie teraz wyobrazić sytuację, w której rzucamy bardzo dużo razy rzutkami w tarczę wyglądającą jak Rysunek 1. Kończąc grę, plansza będzie pokryta rzutkami znajdującymi się we wnętrzu koła jak i poza nim na obszarze kwadratu. Podsumowując liczba rzutek w i poza kołem będzie równa stosunkowi pola koła do pola kwadratu.



Rysunek 2: Przykład losowania punktów.

Formalnie losowanie będzie odbywać się pośród punktów należących do zbioru pokrytego współrzędnymi należącymi do przedziału  $[-2r, 2r]$ . Stosunek liczby punktów zawierających się w kole o środku w punkcie  $[0,0]$  i promieniu  $r$  do wszystkich wylosowanych punktów będzie dążył w nieskończoności (z pewnym prawdopodobieństwem) do stosunku tego pola koła do koła kwadratu o boku  $2r$ . Co więcej, stosunek ten będzie identyczny również do ćwiartki koła. Jeżeli pole koła podzielimy na cztery i tak samo podzielimy pole kwadratu, to ich stosunek będzie wciąż taki sam. Oznacza to, że wystarczy, jeżeli będziemy losowali punkty o współrzędnych od 0 do  $r$ . Cała metoda sprowadza się więc do tego, by losować punkty, sprawdzać, czy mieszczą się w kole, i następnie podstawiać liczby wylosowanych punktów do wzoru. Losując odpowiednio dużo punktów, powinniśmy otrzymać z pewnym prawdopodobieństwem rozsądne przybliżenie liczby Pi.

### 3.3 Implementacja algorytmu - program jednowątkowy.

Dla lepszego zrozumienia działania algorytmu i sprawdzenia jego rezultatów stworzony został program w języku C++ obliczający przybliżenie liczby Pi wykonujący wszystkie obliczenia szeregowo, czyli po prostu korzystający z jednego wątku. Program przyjmuje od użytkownika zadaną liczbę punktów. W rezultacie wyświetla obliczoną liczbę Pi oraz oryginalne rozwinięcie.

```
1 // Obliczanie_liczby_pi.cpp : wersja zwykła -jednowatkowa
2 // Autor : Pawel Sternik
3 // Data : 17.03.2015
4 // Metoda Monte Carlo wyznaczania liczby Pi
5
6
7 #include <iostream>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <time.h>
11 #include <math.h>
12
13 using namespace std;
14
15 void PobieranieDanych(int &iloscPunktow)
16 {
17     cout << "Proszę podać ilość wszystkich punktów: ";
18     cin >> iloscPunktow;
19 }
20
21 int main( int argc, char * argv[] )
22 {
23     // Deklaracja wszystkich zmiennych
24     int liczbaWszystkichPunktow, bokKwadratu,
25     poleKwadratu, liczbaPunktowKolo = 0;
```

```

26     double promienKola;
27     double **wszystkiePunkty;
28     long double mojePi;
29     srand(time(NULL));
30     // Pobieranie od uzytkownika liczby punktow
31     PobieranieDanych(liczbaWszystkichPunktow);
32     bokKwadratu = 1;
33     poleKwadratu = 1;
34     promienKola = bokKwadratu / 2;
35     // Utworzenie macierzy do przechowywania punktow w kwadracie
36     wszystkiePunkty = new double*[liczbaWszystkichPunktow];
37     for(int i=0; i < liczbaWszystkichPunktow; i++)
38     {
39         wszystkiePunkty[i] = new double[2];
40     }
41
42     // Losowanie wspolrzecznych punktu w kwadracie z zakresu -0,5 do 0,5
43     for(int i=0; i < liczbaWszystkichPunktow; i++)
44     {
45         for(int j=0; j < 2 ; j++)
46         {
47             double a = ( rand() % 10000 ) -
5000;
48             a = a / 10000;
49             wszystkiePunkty[i][j] = a;
50         }
51     }
52     for(int i=0; i < liczbaWszystkichPunktow; i++)
53     {
54         double potega = pow(wszystkiePunkty[i][0], 2) + pow(
wszystkiePunkty[i][1], 2);
55         double odleglosc = sqrt(potega);
56         if(odleglosc <= 0.5)
57         {
58             liczbaPunktowKolo++;
59         }
60     }
61
62     // "WYSWIETLENIE REZULTATOW DZIALANIA PROGRAMU – POROWNANIE"
63     cout << "
64     _____\n";
65     cout << "Liczba punktow w kole: " << liczbaPunktowKolo
66     << " na " << liczbaWszystkichPunktow << "\n";
67     mojePi = ((double)liczbaPunktowKolo/((double)liczbaWszystkichPunktow) *
4.0;
68     cout << "Moje Pi = " << mojePi << "\n";
69     cout << "Originalne Pi = 3.14159265359\n";

```

```
69  cout << "
70  _____\n";
71  // Zwolnienie pamieci zaalokowanej przez macierz
72  for(int i=0; i < liczbaWszystkichPunktow; i++)
73  {
74      delete [] wszystkiePunkty[i];
75  }
76
77  delete [] wszystkiePunkty;
78  return 0;
79 }
```

```
MacBook-Air-Pawe:Kod Pawel$ cd Obliczanie_liczby_Pi/
MacBook-Air-Pawe:Obliczanie_liczby_Pi Pawel$ ls
makefile          obliczanie_pi.o
obliczanie_pi.cpp  pi
MacBook-Air-Pawe:Obliczanie_liczby_Pi Pawel$ ./pi
Prosze podac ilosc wszystkich punktow: 10000000
-----
Liczba punktow w kole: 7855365 na 10000000
Moje Pi = 3.14215
Originalne Pi = 3.14159265359
-----
MacBook-Air-Pawe:Obliczanie_liczby_Pi Pawel$ █
```

Rysunek 3: Przykład działania programu jednowątkowego.

Jak widać na zamieszczonym zdjęciu ekranu program jednowątkowy obliczający rozwinięcie dziesiętne liczby Pi korzystając z metody Monte Carlo obliczył liczbę Pi równą 3.14215. Otrzymana dokładność sięga jedynie dwóch miejsc po przecinku przy stosunkowo już dużej ilości losowanych punktów - 10 mln. Program wyświetlił informację o tym, że 7855365 punktów znalazło się w kole.

## 4 Zastosowane technologie.

### 4.1 Framework Django.

#### 4.1.1 Opis.

Django – wysokopoziomowy, opensource’owy framework przeznaczony do tworzenia aplikacji internetowych, napisany w Pythonie. Powstał pod koniec 2003 roku jako ewolucyjne rozwinięcie aplikacji internetowych, tworzonych przez grupę programistów związanych z Lawrence Journal-World. Nazwa frameworku pochodzi od imienia gitarzysty Django Reinhardta. Framework wymaga instalacji interpretera Pythona. W ramach pro-



jektu wykorzystany został Python 2.7.6, instalacja jednak nie była konieczna, ponieważ w systemie Linux był wgrany domyślnie.

#### 4.1.2 Instalacja Django.

W projekcie zostało użyte Django w wersji 1.7.5, jednak od 01.04.2015 dostępna jest już wersja 1.8 LTS:

```
pip install Django==1.7.5
```

#### 4.1.3 Tworzenie nowego projektu.

W linii poleceń należy przejść do katalogu, w którym będzie znajdował się kod, następnie wpisując:

```
django-admin.py startproject PI
```

Utworzony zostanie katalog z projektem: PI/

init.py

manage.py

settings.py

urls.py

- **init.py:** Pusty plik informujący Pythona o tym, że katalog nadrzędny powinien być traktowany jako pakiet Pythona
- **manage.py:** Działające z linii poleceń narzędzie, które pozwala na interakcję z projektem Django na różne sposoby.
- **settings.py** Ustawienia/konfiguracja dla tego projektu Django.
- **urls.py** Deklaracja adresów URL dla tego projektu Django; “mapa serwisu” Twojej strony zbudowanej w oparciu o Django.

Przechodząc do katalogu PI, można sprawdzić czy server deweloperski działa, w tym celu należy wpisać:

```
python manage.py runserver
```

#### 4.1.4 Ustawienia bazy danych.

Django można używać bez bazy danych, jednak dostarczony jest wraz z systemem ORM (odzworowanie relacyjno-obiektowe), za pomocą, którego można definiować tabele w bazie danych oraz pracować na nich pisząc i operując na klasach w Pythonie. Django współpracuje z następującymi serwerami bazodanowymi: PostgreSQL, MySQL, Oracle i SQLite. Ze względu na łatwość obsługi wybrany został MySQL, przez co należało doinstalować MySQLdb:

**sudo apt-get install python2.7-mysqldb**

Co wymagało również zmiany silnika bazy danych w ustawieniach aplikacji na „django.db.backends.mysql” oraz z wnętrza powłoki bazy danych utworzenia potrzebnej bazy danych. Ze względu, że w pliku settings.py INSTALLED\_APPS zawarte są domyślne aplikacje, należy użyć komendy:

**python manage.py syncdb**, która dla w/w aplikacji tworzy tabele w bazie.

**4.1.5 Tworzenie modeli.**

W katalogu PI, aby zacząć pracę nad projektem należy wprowadzić następujące polecenie:

**python manage.py startapp PI2**

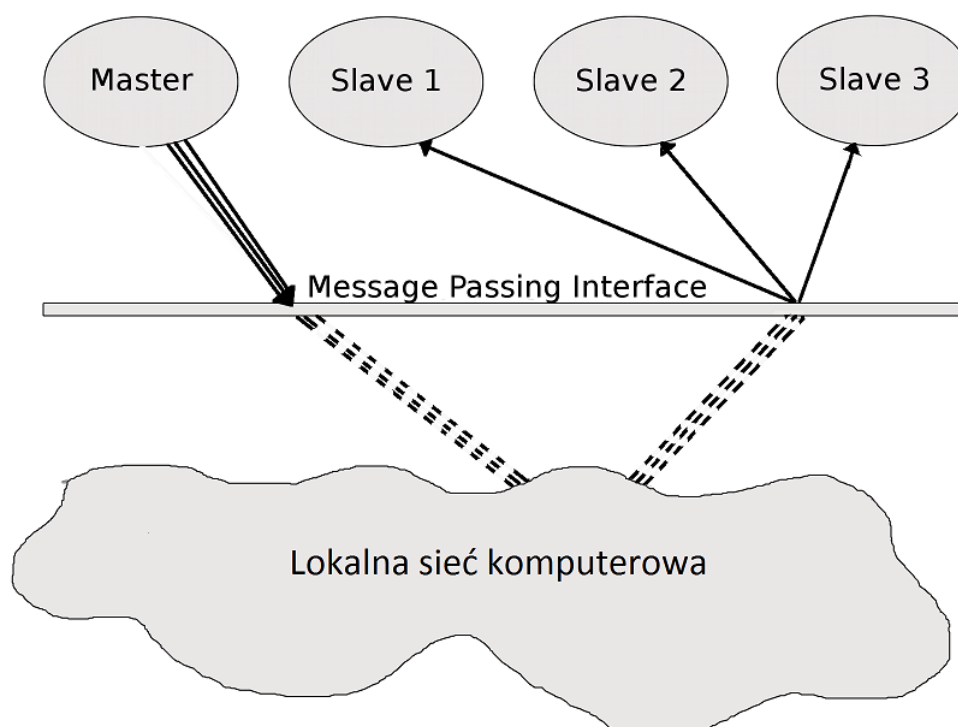
Co utworzy podkatalog PI2, w którym będzie się znajdować struktura docelowej aplikacji.

**4.2 Technologia MPI.**

MPI, czyli Message Passing Interface jest protokołem komunikacyjnym będącym standardem przesyłania komunikatów pomiędzy procesami programów równoległych działających na jednym, lub wielu komputerach. Ze względu na dominację wykorzystywania tego modelu w klastrach komputerowych oraz superkomputerach, posłużył on jako główny protokół komunikacyjny w projekcie. MPI jest specyfikacją biblioteki funkcji opartych na modelu wymiany komunikatów dla potrzeb programowania równoległego. Transfer danych pomiędzy poszczególnymi procesami programu wykonywanymi na procesorach maszyn będących węzłami klastra odbywa się za pośrednictwem sieci. Opisany schemat działania ilustruje poniższy rysunek.

Aby zapewnić sprawną komunikację pomiędzy komputerem Master, a Slave’ami należy skonfigurować bez kluczowe połączenia. Do tego celu najlepszym rozwiązaniem jest użycie SSH, które umożliwi logowanie bez użycia hasła, wykorzystując do tego klucz publiczny, na przykład RSA. Podstawowy schemat działania wygląda następująco:

- **sudo apt-get install openssh-server** - instalacja serwera ssh na komputerze Master
- **sudo apt-get install openssh-client** - instalacja klientów na komputerach Slave
- **ssh-keygen -t rsa** - wygenerowanie kluczy na komputerze Master
- **scp .ssh/id\_rsa.pub username@IP\_addr:** - skopiowanie klucza publicznego do Slave’a



Rysunek 4: Ogólny schemat działania protokołu MPI.

- **cat id\_rsa.pub » .ssh/authorized\_keys** - z poziomu Slave'a dodanie otrzymanych kluczy publicznych do kluczy autoryzowanych

Mając skonfigurowane połączenie pomiędzy hostami, można skompilować program wydając polecenie:

- **mpicc pi.c -o pi.out**

Po poprawnym i bezbłędnym skompilowaniu kodu, powinien pojawić się plik wynikowy z rozszerzeniem **.out**, który należy skopiować do wszystkich hostów:

- **scp pi.out username@ip\_addr:**

Jeśli wszystkie hosty posiadają ten sam program, przed uruchomieniem klastra należy stworzyć i skonfigurować plik, zawierający konfigurację maszyn w klastrze. Tworzymy plik, np. **.mpi\_hostfile**, który należy uzupełnić według podanego schematu:

```

1 #Hostfile dla Open MPI
2
3 #Master node, parametr 'slots=2' ustawiony, dlatego, że node jest dwu-
  procesorowy
4 localhost slots=2
5
6 #Definiujemy slave'y, oraz dozwolona liczba procesorów do użycia
7 cichy@192.168.1.118 slots=2 max_slots=4
```

```
8 slawek@192.168.1.119 slots=8 max_slots=8
```

Jeśli wszystko przebiegło pomyślnie możliwe jest uruchomienie programu rozproszonego:

- `mpirun -np liczbaProcesorow -hostfile nazwaHostfile /sciezka/pi.out`

### 4.3 CSS i HTML.

### 4.4 Komunikacja grupy.

Podczas realizacji projektu wykorzystane zostały narzędzia ułatwiające pracę w grupie. Stworzone zostało repozytorium, w którym umieszczane były kody źródłowe programu obliczającego liczbę Pi czy też aplikacji internetowej napisanej w frameworku Django. Ponadto na portalu Trello istniała tablica na której zamieszczane były zadania do zrealizowania. Umożliwiło to jasny podział zadań oraz bez problemową serializację efektów pracy niwelując liczbę konfliktów podczas równoległego tworzenia kodu przez różne osoby. Podczas realizacji projektu każdy członek z grupy dzięki dobrze zoorganizowanej tablicy Trello znał swoje zadania oraz terminy realizacji, które stanowiły motywację do pracy.

#### 4.4.1 GitHub.



Rysunek 5: Logo portalu GitHub.

GitHub to serwis internetowy wykorzystywany masowo przy projektach programistycznych, korzystający z systemu kontroli wersji Git. Stworzony został już w 2008 roku przy wykorzystaniu języków Ruby on Rails oraz Erlang. Dzisiaj serwis obsługuje już kilka milionów repozytoriów, a wśród nich darmowe publiczne repozytoria oraz płatne prywatne. Darmowe repozytoria utrzymywane są na licencji open source. Podczas realizacji

projektu został stworzony bardzo prosty tutorial korzystania z Git'a z poziomu konsoli poleceń. Tutorial opisuje korzystanie z podstawowych funkcjonalności systemu kontroli wersji, wyłączając bardziej zaawansowane techniki branch'owania i merge'owania projektów.



Rysunek 6: Logo systemu kontroli wersji- Git.

Instrukcja do skonfigurowania i uruchomienia repozytorium opartego o system kontroli wersji Git na systemie Linux:

1. Instalacja najnowszej wersji gita za pomocą komendy:  
**sudo apt-get install git**
2. Drugi krok to ustawienie globalnych danych konta założonego na stronie GitHub:  
**sudo git config --global user.name "nazwaKonta"**
3. Następnie podajemy analogicznie maila na którego mamy założone konto na GitHub:  
**sudo git config --global user.email "email"**
4. Przed tym krokiem zostały już skonfigurowane dane konta. Teraz należy je potwierdzić i połączyć się z interesującym nas repozytorium. Do nawiązania połączenia wykorzystywane są dwa sposoby: protokół HTTPS lub SSH. Tym razem wykorzystany będzie protokół HTTPS.
5. Prawdopodobnie po skonfigurowaniu danych konta Git pobierze automatycznie repozytoria, do których dane konto miało dostęp i umieści je w folderze Dokumenty.
6. W innym przypadku należy skopiować adres repozytorium ze strony portalu. Przykładowo:  
**<https://github.com/syguts/AiIR0730obliczanieliczbyPI.git>**
7. Nawiązanie połączenia poprzez komendę:  
**git clone https://github.com/syguts/AiIR0730obliczanieliczbyPI.git AIIR**  
Gdzie "AIIR" to nazwa folderu, do którego zostanie pobrane repozytorium (wersja lokalna).

Natomiast same edytowanie repozytorium sprowadza się do sekwencji:

1. **git fetch** - funkcja sprawdza wersje zdalnego repozytorium (jeżeli zdalna wersja jest identyczna co do lokalnej wówczas wyświetlany jest stosowny komunikat)

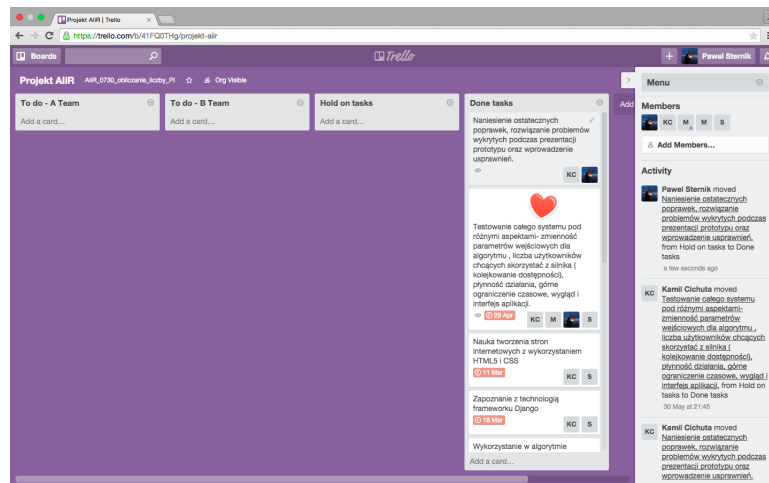
2. **git log** - metoda wyświetla listę wprowadzonych zmian
3. **git pull** - pobranie wprowadzonych zmian - zaaktualizowanie lokalnego repozytorium.
4. **git add** - w przypadku dodania plików należy podłączyć je pod system kontroli wersji metodą `git add nazwaPlikuFolderu`
5. **git status** - metoda listująca zmiany wprowadzone przez użytkownika w wersji lokalnej od ostatniej aktualizacji
6. **git commit** - stworzenie commita zbierającego wprowadzone zmiany
7. **git push** - wypchanie wprowadzonych zmian na zewnątrz lokalnego repo, czyli dodanie do zdalnego repozytorium.

#### 4.4.2 Trello.



Rysunek 7: Logo portalu Trello.

Trello to aplikacja do zarządzania zadaniami. Opiera się ona na paradygmacie Kanban stworzonym w Japoni, w firmie Toyota, należącym do technik Agile. Trello to także produkt horyzontalny. Co to znaczy?. Oznacza to, że nie jest to produkt skierowany konkretnie do jednej grupy odbiorców, a wszystkich, którzy potrzebują danego narzędzia oraz chcących skorzystać z jego funkcji. Projekty w Trello zbudowane są w oparciu o tablice, na których to znajdują się listy zadań. Listy te pozwalają tworzyć karty (zadania), które w bardzo prosty sposób możemy sortować, czy przenosić pomiędzy różnymi listami za pomocą techniki drag-and-drop. Poszczególni użytkownicy mogą być przypisywani do kart oraz budować grupy, a grupy i konkretne projekty tworzą organizacje. Bardzo przydatną funkcjonalnością Trello są także powiadomienia, które spływają w formie wiadomości na adres email lub powiadomień (push) na urządzenia mobilne. Każdy użytkownik ma oczywiście kontrolę nad nimi oraz ich częstotliwością. Dzięki powiadomieniom można być informowanym o dyskusjach, w których pojawia się nazwa użytkownika, obserwowanych projektach, lub aktywnościach innych użytkowników. W ten sposób komunikacja następuje w błyskawicznym tempie.



Rysunek 8: Przykładowy widok tablicy na portalu Trello.

## 5 Plan realizacji.

1. Dokładna analiza algorytmu matematycznego na obliczanie liczby pi - Monte Carlo.
2. Zebranie informacji o metodach współbieżnych tego algorytmu lub też próba własnej implementacji.
3. Podział pracy na dwa zespoły:

Drużyna A - Cebula i Sternik:

- Implementacja algorytmu współbieżnego w języku C++
- Nauka i zapoznanie z technologią MPI
- Implementację technologii MPI
- Testy połączenia między komputerami
- Wykorzystanie w algorytmie współbieżnym
- Testy silnika obliczeniowego - liczba pi

Drużyna B - Sygut i Cichuta:

- Nauka tworzenia stron internetowych z wykorzystaniem HTML5 i CSS
- Zapoznanie z technologią frameworku Django
- Utworzenie konta administracyjnego i użytkowników
- Stworzenie strony internetowej
- Testy- dodawanie użytkowników, administracja, wygląd na różnych przeglądarkach, pobieranie danych z formularzy
- Postawienie aplikacji na serwerze i próba uruchomienia zdalnego

4. Połączenie silnika obliczeniowego z frontendem i serwerem aplikacji.
5. Testowanie całego systemu pod różnymi aspektami- zmienność parametrów wejściowych dla algorytmu , liczba użytkowników chcących skorzystać z silnika ( kolejkiwanie dostępności), płynność działania, górne ograniczenie czasowe, wygląd i interfejs aplikacji.
6. Tworzenie dokumentacji z podziałem prac według opracowanych obszarów systemu.
7. Przygotowanie prezentacji końcowej przedstawiającej efekty pracy oraz wnioski. Podsumowanie całego semestru prac - trudności i problemy oraz ich rozwiązania.

## 6 Implementacja silnika obliczeniowego.

### 6.1 Zrównoleglenie algorytmu

Pierwszym krokiem podczas implementacji silnika obliczeniowego - programu obliczającego przybliżenie liczby Pi w oparciu o algorytm Monte Carlo była implementacja programu współbieżnego. Wykorzystana została w tym celu biblioteka systemu Linux - POSIX Threads. Biblioteka udostępnia jednolite API dla język C do tworzenia i zarządzania wątkami, jest dostępna w systemach m.in. Linux, FreeBSD, Windows. Implementacja miała na celu przygotowanie merytoryczne do implementacji algorytmu z wykorzystaniem technologii MPI. W tym celu zalecane było zapoznanie się z metodą zrównoleglenia tego algorytmu. Poniżej Listing programu z wykorzystaniem POSIX Threads.

```
1 #include <pthread.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <stdio.h>
5 #include <iostream>
6 #include <time.h>
7 #include <math.h>
8 using namespace std;
9
10 pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
11 int *counter;
12 int przydzielPunktow;
13 int idWatku=0;
14
15 void *MojaFunkcjaDlaWatku(void *arg)
16 {
17     double x, y, potega, odleglosc;
18     pthread_mutex_lock(&mutex1);
19     int numerek = idWatku;
20     idWatku++;
```



```
21 cout << endl<< numererek << endl;
22 pthread_mutex_unlock(&mutex1);
23 for(int i=0; i < przydzialPunktow; i++)
24 {
25     x = ( ( (double)rand() / (RAND_MAX) ) * 2) - 1;
26     y = ( ( (double)rand() / (RAND_MAX) ) * 2) - 1;
27     potega = pow(x, 2) + pow(y, 2);
28     odleglosc = sqrt(potega);
29     if(odleglosc <= 1)
30     {
31         counter[numerek]+= 1;
32     }
33 }
34
35 return NULL;
36 }
37
38 void PobieranieDanych(int &iloscPunktow, int &watki)
39 {
40     cout << "Prosze podac ilosc wszystkich punktow: ";
41     cin >> iloscPunktow;
42     cout << "Prosze podac ilosc watekow: ";
43     cin >> watki;
44 }
45
46 int main(void)
47 {
48     int punkty, liczbaWatkow, *tablicaPrzydzialu;
49     pthread_t *mojwatek;
50     PobieranieDanych(punkty, liczbaWatkow);
51     mojwatek = new pthread_t[liczbaWatkow];
52     srand(time(NULL));
53     tablicaPrzydzialu = new int[liczbaWatkow];
54     int sredniaPunktow = punkty / liczbaWatkow;
55     counter = new int[liczbaWatkow];
56
57     for(int i=0; i < liczbaWatkow -1; i++)
58     {
59         tablicaPrzydzialu[i] = sredniaPunktow;
60     }
61
62     tablicaPrzydzialu[liczbaWatkow-1] = punkty - (sredniaPunktow * (
        liczbaWatkow-1));
63
64     for(int i=0; i < liczbaWatkow; i++)
65     {
66         przydzialPunktow= tablicaPrzydzialu[i];
67         if ( pthread_create( &mojwatek[i], NULL, MojaFunkcjaDlaWatku, NULL))
```

```
68     {
69         cout << "blad przy tworzeniu watku\n";
70         abort();
71     }
72 }
73
74 for(int i=0; i < 4; i++)
75 {
76     if ( pthread_join ( mojwatek[i], NULL ) )
77     {
78         cout << "blad w konczeniu watku\n";
79         abort();
80     }
81 }
82
83 int suma=0;
84
85 for(int i=0; i < liczbaWatkow ; i++)
86 {
87     suma += counter[i];
88 }
89
90 // "WYSWIETLENIE REZULTATOW DZIALANIA PROGRAMU – POROWNANIE Z ORGINALEM"
91 cout << "
92     _____\n";
93 cout << "Liczba punktow w kole: " << suma << " na " << punkty << "\n";
94     long double mojePi = ((double)suma/((double)punkty) * 4.0;
95     cout << "Moje Pi = " << mojePi << "\n";
96     cout << "Orginalne Pi = 3.14159265359\n";
97     cout << "
98     _____\n";
99
100     exit(0);
101 }
```

Program pobiera od użytkownika ilość punktów biorących udział w losowaniu oraz ilość wątków. Następnie w zależności od liczby wątków obliczany jest zakres - przydział punktów dla każdego z wątków. Każdy z wątków zlicza ilość punktów znajdujących się w kole. Po wykonaniu się funkcji wszystkich wątków, wątek macierzysty sumuje poszczególne liczniki i oblicza wartość przybliżenia Pi.

**7 Aplikacja internetowa.**

**8 Testy.**

**9 Podsumowanie i wnioski.**