

Wrocław, dn. 10 czerwca 2015r.

POLITECHNIKA WROCŁAWSKA

APLIKACJE INTERNETOWE I ROZPROSZONE INES00304P

Obliczanie przybliżenia liczby π .

DOKUMENTACJA KOŃCOWA

Grupa projektowa:

Paweł STERNIK, 200623
Kamil CICHUTA, 200838
Mariusz CEBULA, 20772
Sławomir SYGUT, 200668

Prowadzący:

dr inż. Marek WODA

Termin spotkań:

Środa, godz. 07:30

Spis treści

1	Temat projektu.	2
2	Cel projektu.	2
2.1	Cel dydaktyczny.	2
2.2	Cel merytoryczny.	2
3	Opis zastosowanego algorytmu.	3
3.1	Opis słowny algorytmu.	3
3.2	Obliczanie liczby Pi metodą Monte Carlo.	3
3.3	Implementacja algorytmu - program jednowątkowy.	5
4	Zastosowane technologie.	8
4.1	Framework Django.	8
4.1.1	Opis	8
4.1.2	Instalacja Django	8
4.2	Technologia MPI.	9
4.3	CSS i HTML.	11
4.4	Komunikacja grupy.	12
4.4.1	GitHub.	12
4.4.2	Trello.	14
5	Plan realizacji.	15
6	Implementacja silnika obliczeniowego.	16
6.1	Zrównoleglenie algorytmu	16
6.2	Zastosowanie technologii MPI	18
6.3	Komunikacja z bazą danych aplikacji.	21
7	Aplikacja internetowa.	25
7.1	Tworzenie nowego projektu.	25
7.2	Ustawienia bazy danych.	25
7.3	Tworzenie modeli.	26
7.4	Tworzenie widoków wraz z obsługą żądań	27
7.5	Zrzuty ekranów aplikacji internetowej	29
8	Testy.	31
9	Podsumowanie i wnioski.	33
10	Bibliografia	33

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.

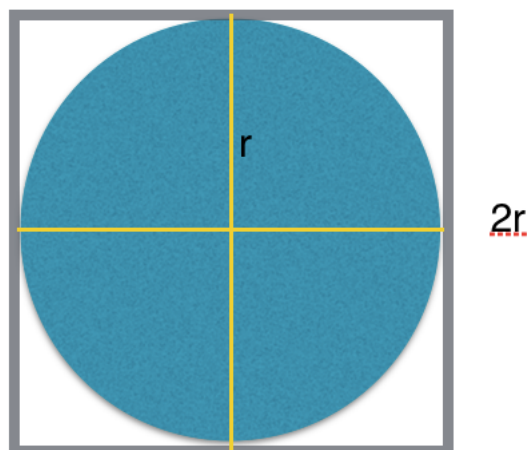
Liczbę Pi można obliczać na wiele różnych sposobów. Jednym z nich jest wykorzystanie metody Monte Carlo. Metoda ta charakteryzuje się przede wszystkim swoją stosunkowo prostą procedurą jak i przystosowaniem do zaimplementowania mechanizmów zrównoleglenia - co jest jednym z głównych celów projektu.

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 $2r$.

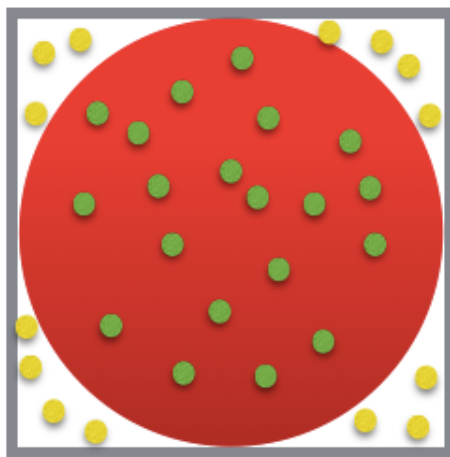
Znając zatem pole koła jak i kwadratu można zauważyć że stosunek pola koła do pola kwadratu wynosi:

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

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 << "Prosze podac ilosc wszystkich punktow: ";
18     cin >> iloscPunktow;
19 }
20
21 int main( int argc , char * argv[] )
22 {
23     // Delaracja 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 wspolrzednych 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     // "WYSWETLENIE 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;
```

```
67     cout << "Moje  Pi = " << mojePi << "\n";
68     cout << "Orginalne Pi = 3.14159265359\n";
69     cout << "
    _____\n";
70
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
Orginalne 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.



Rysunek 4: Logo frameworku 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 projektu wykorzystany został Python 2.7.6, instalacja jednak nie była konieczna, ponieważ w systemie Linux był wgrany domyślnie.

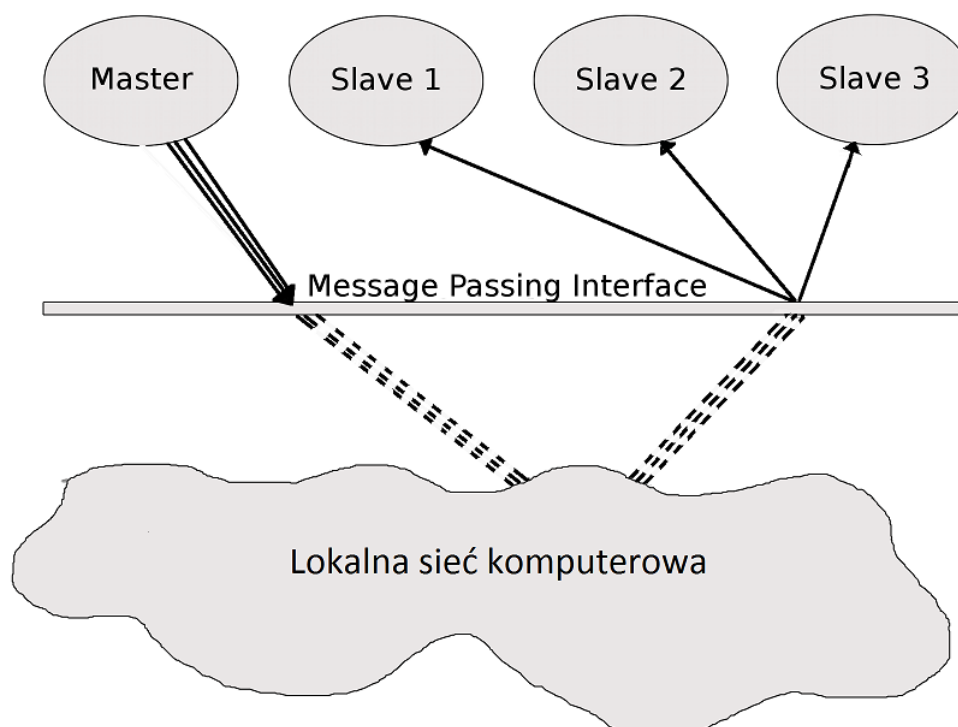
4.1.2 Instalacja Django

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

4.2 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 Rysunek 5.



Rysunek 5: Ogólny schemat działania protokołu MPI.

Do poprawnego uruchomienia klastra obliczeniowego w technologii **MPI** potrzebne jest kilka rzeczy:

- co najmniej dwie maszyny podłączone przez TCP/IP
- wszystkie maszyny mają zainstalowaną system oparty na Linuxie
- każda maszyna używa takiej samej architektury procesora, aby zapewnić poprawne działanie programu wykonywalnego

Aby móc uruchomić projekt, należy na wszystkich komputerach zainstalować biblioteki **OpenMPI**:

```
1 sudo apt-get install openmpi-bin openmpi-common libopenmpi1 libopenmpi-dev
```

Dodatkowo, 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
- **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, ze node jest dwu-
  procesorowy
4 localhost slots=2
5
6 #Definiujemy slave'y, oraz dozwolona liczbe procesorow do uzycia
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.



Rysunek 6: Logo HTML5 i CSS3.

HTML5 – to język wykorzystywany do tworzenia i prezentowania stron internetowych. Jest rozwinięciem języka HTML 4 i jego XML-owej odmiany (XHTML 1). Został opracowany w ramach prac grupy roboczej WHATWG (Web Hypertext Application Technology Working Group) i W3C. Główną różnicą pomiędzy HTML 5 a XHTML 2 polega na tym, że HTML 5 jest pisane z myślą o kompatybilności wstecznej, podczas gdy XHTML 2 porzuciło kompatybilność. HTML5 poza dodaniem nowych elementów, usprawniających tworzenie serwisów i aplikacji internetowych, doprecyzowuje wiele niejasności w specyfikacji HTML 4, dotyczących przede wszystkim sposobu obsługi błędów. Niejasności co do sposobu, w jaki przeglądarki powinny obsługiwać błędy w kodzie HTML są jedną z podstawowych przyczyn, dla której wiele serwisów internetowych, napisanych z naruszeniem specyfikacji, w różnych przeglądarkach działa w inny sposób – w niektórych działając, w innych nie. Dzięki HTML-owi 5 obsługa błędów ma być ta sama we wszystkich przeglądarkach, czyli zły element będzie działać w każdej przeglądarce albo żadnej. Bootstrap to framework CSS i Javascript, mający za zadanie ułatwić zaprojektowanie kompletnej strony internetowej w prosty sposób. Założeniem frameworka Twitter Bootstrap jest udostępnienie gotowych elementów, takich jak listy, slidery czy menu, które umożliwiają zaprojektowanie strony w spójny sposób w możliwie krótkim czasie. Sam framework oprócz bogatej biblioteki stylów CSS zawiera także elementy takie jak „karuzele” które obsługiwane są z użyciem języka JavaScript. Oprócz podstawowych elementów, takich jak przygotowane wstępnie górne menu nawigacyjne strony, czy też elementy list i formularzy dostępne są także różnego rodzaju slidery, boxy na treść wyróżnioną, boxy powiadomień czy też gotowe, uproszczone nagłówki

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 7: 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.

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

1. Instalacja najnowszej wersji gita za pomocą komendy:



Rysunek 8: Logo systemu kontroli wersji- Git.

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.

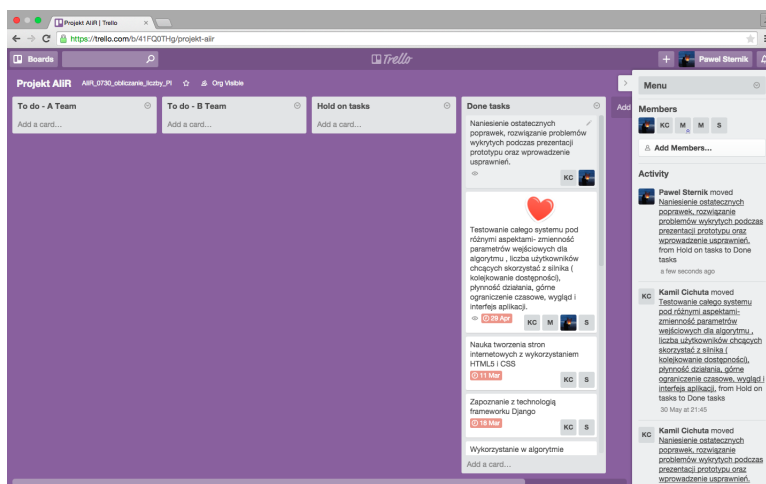
4. **git add** - w przypadku dodania plików należy podłączyć je pod system kontroli wersji metodą `git add nazwaPlikuFolderu`
5. **git status** - metoda listująca zmiany wprowadzone przez użytkownika w wersji lokalnej od ostatniej aktualizacji
6. **git commit** - stworzenie commita zbierającego wprowadzone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

4.4.2 Trello.



Rysunek 9: 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 10: 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<< numerek << 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 << "Originalne 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.

6.2 Zastosowanie technologii MPI

Następnym krokiem podczas realizacji silnika obliczeniowego było zaimplementowanie protokołu komunikacyjnego, czyli technologii MPI. Działa ona analogicznie co do metody współbieżnej przy czym wątki POSIX zostały zastąpione slave'ami MPI. Program

posiada zdefiniowane na sztywno zmienne z biblioteki MPI określające liczbę stacji roboczych, które wczytuje z pliku konfiguracyjnego. Następnie identycznie jak w poprzednim programie w zależności od ilości punktów i maszyn wylicza zakresy dla każdego z wątków. Program przyjmuje w parametrach wywołania ilość punktów. Master po zakończeniu pracy slave'ów zlicza wynik cząstkowe i podstawia do wzoru obliczającego przybliżenie liczby Pi. Poniżej Listing programu:

```
1  /**
2  Calculate Pi with MPI protocol
3  Author : Pawel Sternik
4  Date: 22.05.2015
5  **/
6  #include "mpi.h"
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <time.h>
10 #include <string.h>
11 #include <stdbool.h>
12
13 void srandom (unsigned seed);
14 long double dboard (int darts);
15 #define MASTER 0
16
17 int main (int argc, char *argv[])
18 {
19     srandom(time(NULL));
20     clock_t    start=clock();
21     double     homepi,                /* value of pi calculated by
current task */
22     pisum,                /* sum of tasks' pi values */
23     avepi;                /* average pi value for all
iterations */
24     int        taskid,          /* task ID – also used as seed
number */
25     numtasks,                /* number of tasks */
26     rc,                      /* return code */
27     i,
28     DARTS = 0,
29     ROUNDS = 1;
30     long double    pi;          /* average of pi after "darts"
is thrown */
31     MPI_Status status;
32
33     /* Obtain number of tasks and task ID */
34     MPI_Init(&argc,&argv);
35     MPI_Comm_size(MPLCOMM_WORLD,&numtasks);
36     MPI_Comm_rank(MPLCOMM_WORLD,&taskid);
37
```

```
38     /* Set seed for random number generator equal to task ID */
39     srandom (taskid);
40
41     /* Number of random points */
42     int zakres = atoi(argv[1])/numtasks;
43     DARTS = zakres;
44     avepi = 0;
45
46     for (i = 0; i < ROUNDS; i++)
47     {
48         /* All tasks calculate pi using dartboard algorithm */
49         homepi = dboard(DARTS);
50         rc = MPI.Reduce(&homepi, &pisum, 1, MPLDOUBLE, MPLSUM,
51             MASTER, MPLCOMM_WORLD);
52
53         if (rc != MPLSUCCESS)
54         {
55             printf("%d: failure on mpc_reduce\n",
taskid);
56         }
57
58         /* Master computes average for this iteration and all
iterations */
59         if (taskid == MASTER)
60         {
61             pi = pisum/numtasks;
62             avepi = ((avepi * i) + pi)/(i + 1);
63             printf("%10.16f\n", avepi);
64         }
65     }
66
67
68     /* The finalize timers and display time for running algorithm */
69     clock_t end=clock();
70     float seconds = (float)(end - start) / CLOCKS_PER_SEC;
71     MPI_Finalize();
72     return 0;
73 }
74
75 long double dboard(int darts)
76 {
77     #define sqr(x) ((x)*(x))
78     long random(void);
79     double x_coord, y_coord, pi, r;
80     int score, n;
81     unsigned int cconst; /* must be 4-bytes in size */
82     int counterLoop=0;
83     if (sizeof(cconst) != 4)
```

```
84         {
85             printf("Wrong data size for cconst variable in dboard
routine!\n");
86             exit(1);
87         }
88         /* 2 bit shifted to MAXRAND later used to scale random number
between 0 and 1 */
89         cconst = 2 << (31 - 1);
90         score = 0;
91         /* "throw darts at board" */
92         for (n = 0; n < darts; n++)
93         {
94             if(counterLoop == n)
95             {
96                 counterLoop += darts/10;
97             }
98             /* generate random numbers for x and y coordinates */
99             r = (double)random()/ccconst;
100             x_coord = (2.0 * r) - 1.0;
101             r = (double)random()/ccconst;
102             y_coord = (2.0 * r) - 1.0;
103
104             /* if dart lands in circle , increment score */
105             if ((sqr(x_coord) + sqr(y_coord)) <= 1.0)
106             {
107                 score++;
108             }
109         }
110
111         /* calculate pi */
112         pi = 4.0 * (double)score/(double)darts;
113         return(pi);
114     }
```

6.3 Komunikacja z bazą danych aplikacji.

Komunikacja z bazą danych aplikacji została zrealizowana poprzez wykorzystanie biblioteki języka C: mysql.h. Biblioteka zawiera metody umożliwiające zdalne połączenie z bazą danych MySQL poprzez adres IP i numer portu. Wymaga to również podanie danych użytkownika danej bazy. Zatem został zaimplementowany osobny program obsługujący akcje dokonywane na bazie danych i nasłuchujący pojawienie się zadań zleconych przez użytkowników aplikacji internetowej. Takie rozwiązanie umożliwiło stworzenie dodatkowej cechy systemu. Użytkownik jest w stanie określić ilość wątków na których ma zostać uruchomiony algorytm .Realizowane jest to poprzez wywołanie programu liczącego z poziomu komendy system z odpowiednim parametrem kompilatora mpi open . Przykładowo

:

```
system("mpirun -np lilosProcesow second iloscPunktow")
```

Podsumowując program działa w procedurze:

1. Nawiązanie połączenia z bazą danych poprzez metodę **mysqlRealConnect**, która wymaga podania adresu bazy danych, nazwy użytkownika i jego hasła.
2. Program oczekuje czy zadanie pojawi się w bazie danych sprawdzając metodą biblioteki mysql pierwszy wiersz tabeli :

```
mysqlQuery(conn, "SELECT * FROM PI2Task limit 1")
```

3. Po pojawieniu się zadania program pobiera dane zleconego zadania metodami : **mysqlUseResult(conn)**, gdzie conn jest uchwyttem połączenia z bazą, **mysqlFetchRow(res)**, gdzie res to wszystkie dane.
4. Do tabeli wyników zostaje dodany wiersz sygnalizujący aplikację internetową o podjęciu zlecenia - włączenie dynamicznego progresu. Wiersz nie posiada wówczas uzupełnionej kolumny wyniku.
5. Natomiast w tabeli zleconych zadań zostaje usunięty wiersz pobranego zadania. Dzięki temu tabela działa jak kolejka FIFO - First Input First Output.
6. Następuje wywołanie programu liczącego z pobranymi wcześniej parametrami zadanymi przez użytkownika aplikacji internetowej.
7. Program po zwróceniu wyniku przez program liczący aktualizuje wiersz w tabeli wyników. Następnie jeżeli coś znajduje się w kolejce od razu przystępuje do realizacji zadania. W innym przypadku oczekuje na zlecenie zadania.

Listing programu łączącego aplikację internetową z programem liczącym:

```
1  /**
2   Author : Pawel Sternik
3   Date : 22.05.2015
4   **/
5
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <time.h>
9  #include <mysql/mysql.h>
10 #include <string.h>
11 #include <stdbool.h>
12
13 int main(int argc, char* argv[]) {
14     FILE *fp;
15     /* Variables for database connection*/
16     MYSQL *conn;
```

```
17  MYSQLRES *res;
18  MYSQLROW row = NULL;
19  char *server = "192.168.1.103";
20  char *user = "admin";
21  char *password = "admin";
22  char *database = "django_db";
23  conn = mysql_init(NULL);
24  int figurePoints, numberOfThreads;
25  char taskID[11], doneTaskID[11], dateTask[25],
26      userID[10], zapytanie[512], napisek[25],
27      polecenie[200], wynik[30];
28  /* Connect to database */
29  if (!mysql_real_connect(conn, server,
30                          user, password,
31                          database, 0, NULL, 0)) {
32      fprintf(stderr, "%s\n", mysql_error(conn));
33      exit(1);
34  }
35  while(1) {
36      /* Waiting for task */
37      while(row == NULL) {
38          if (mysql_query(conn, "SELECT * FROM PI2_task limit 1")) {
39              fprintf(stderr, "%s\n", mysql_error(conn));
40              exit(1);
41          }
42          res = mysql_use_result(conn);
43          row = mysql_fetch_row(res);
44          printf("Waiting for task...\n");
45          sleep(1);
46      }
47
48      /* Output table name */
49      printf("Downloaded data:\n");
50      printf("TaskID: %s FigurePoints: %s DataTask: %s UserID %s\n",
51            row[0], row[1], row[2], row[3], row[4]);
52      /* Save data from database */
53      strcpy(taskID, row[0]);
54      figurePoints = atoi(row[1]);
55      numberOfThreads = atoi(row[4]);
56      strcpy(dateTask, row[2]);
57      strcpy(userID, row[3]);
58      strcpy(napisek, "Zadanie w trakcie");
59
60      /* Number of random points */
61      int zakres = figurePoints/numberOfThreads;
62      DARTS = zakres;
63      snprintf(zapytanie, 512, "DELETE FROM PI2_task WHERE ID=%s", row
64      [0]);
```



```
63     mysql_free_result(res);
64     printf ("Nasz answer %s \n", zapytanie);
65     if (mysql_query(conn, zapytanie)) {
66         fprintf(stderr, "%s\n", mysql_error(conn));
67         exit(1);
68     }
69     printf("Deleted first task.\n");
70
71     /* Answer about start works algorithm */
72     snprintf(zapytanie, 512, "INSERT INTO PI2_donetask (userID_id ,
numberOfPoints, add_date, value) VALUES(%s, %d, '%s', '%s')", userID,
figurePoints, dateTask, napisek);
73     if (mysql_query(conn, zapytanie)) {
74         fprintf(stderr, "%s\n", mysql_error(conn));
75         exit(1);
76     }
77     snprintf(polecenie, 200, "mpirun -np %d second %d >> wynik.txt",
numberOfThreads, figurePoints);
78     // Wywołanie drugiego programu
79     system(polecenie);
80     fp = fopen("wynik.txt", "r");
81     fscanf(fp, "%s", wynik);
82     printf ("WYNIK: %s\n", wynik);
83     fclose(fp);
84     remove("wynik.txt");
85     snprintf(zapytanie, 512, "SELECT * FROM PI2_donetask ORDER BY
add_date desc limit 1");
86     if (mysql_query(conn, zapytanie))
87     {
88         fprintf(stderr, "%s\n", mysql_error(conn));
89         exit(1);
90     }
91     res = mysql_use_result(conn);
92     row = mysql_fetch_row(res);
93     strcpy(doneTaskID, row[0]);
94     mysql_free_result(res);
95     snprintf(zapytanie, 512, "UPDATE PI2_donetask SET value=%s WHERE id
=%s", wynik, doneTaskID);
96     printf ("Nasz answer %s \n", zapytanie);
97     if (mysql_query(conn, zapytanie))
98     {
99         fprintf(stderr, "%s\n", mysql_error(conn));
100         exit(1);
101     }
102     printf("Dodanie do donetask\n");
103     row = NULL;
104 }
105 mysql_close(conn);
```

```
106     return 0;  
107 }
```

7 Aplikacja internetowa.

7.1 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

7.2 Ustawienia bazy danych.

Django można używać bez bazy danych, jednak dostarczony jest wraz z systemem ORM (odzworowanie relacyjno-objektowe), 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.

7.3 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. W pliku models.py zostały stworzone następujące klasy:

```
1 class Task(models.Model):
2     userID = models.ForeignKey('auth.User')
3     numberOfPoints = models.BigIntegerField(default=0)
4     numberOfThreads = models.IntegerField(default=1)
5     add_date = models.DateTimeField(default=timezone.now)
6
7
8 class DoneTask(models.Model):
9     userID = models.ForeignKey('auth.User')
10    numberOfPoints = models.BigIntegerField(default=0)
11    add_date = models.DateTimeField(default=timezone.now)
12    value = models.TextField();
13
14
15
16 class UserProfile(models.Model):
17     user = models.OneToOneField(User)
18
19     def __unicode__(self):
20         return self.user.username
```

W tabeli Task znajdują się zadania, zlecone przez użytkowników, które oczekują w kolejce do pobrania i obliczenia przez silnik.

- userID – ID usera zgłaszającego zadanie
- numberOfPoints – liczba punktów
- numberOfThreads – liczba wątków, na których zostaną wykonane obliczenia

W pliku forms.py zostały dodane klasy obsługi rejestracji oraz zlecenia zadania:

```

1 class CommisionTask( forms .ModelForm ) :
2     numberOfPoints = forms .IntegerField (label ="" , min_value=1)
3     numberOfThreads = forms .IntegerField (label ="" , max_value=12, min_value
    =1)
4
5     class Meta:
6         model = Task
7         fields = ( 'numberOfPoints' , 'numberOfThreads' )

```

Formy pobierające dane zawierają następujące ograniczenia:

- Liczba punktów nie może być mniejsza od 1
- Liczba wątków musi się znajdować w przedziale [1-12]

W pliku admin.py została dodana możliwość przeglądania przez administratora profilów użytkowników, zleconych zadań oraz skończonych zadań.



Rysunek 11: Panel administratora.

7.4 Tworzenie widoków wraz z obsługą żądań

W django do generowania widoków wykorzystywane są pliki z rozszerzeniem ".html". W aplikacji wykorzystano strony napisane w języku html w wersji 5 oparte o technologie twitter bootstrap. Aby w django utworzyć stronę internetową z utworzonymi plikami ".html", należy utworzyć folder w katalogu projektu o nazwie "templates". W folderze tym umieszcza się wszystkie pliki statyczne tzn. dodatkowe biblioteki javascript (z rozszerzeniem: jar, js) pliki stylów "css" oraz wszystkie pliki ".html" tworzonego portalu. Następnie tworzy się plik urls.py w którym umieszcza się ścieżki do plików html oraz podaje obsługujące je metody języka python zdefiniowane w pliku views.py: Listening pliku urls.py który wykorzystano w projekcie:

```

1 from django.conf.urls import patterns , include , url
2 from django.contrib import admin
3

```

```

4 urlpatterns = patterns('',
5     # Examples:
6     # url(r'^$', 'PI.views.home', name='home'),
7     # url(r'^blog/', include('blog.urls')),
8     url(r'^$', 'PI2.views.index'),
9     url(r'^odswiez/$', 'PI2.views.odswiez'),
10    url(r'^odswiez2/$', 'PI2.views.odswiez2'),
11    url(r'^admin/', include(admin.site.urls)),
12    url(r'^register/$', 'PI2.views.register', name='
    register'),
13    url(r'^login/$', 'PI2.views.user_login', name='login
    '),
14    url(r'^logout/$', 'PI2.views.user_logout', name='
    logout'),
15    url(r'^wyczysc/$', 'PI2.views.wyczysc', name='
    wyczysc'),
16    )

```

W pliku views.py zdefiniowane są wszystkie kontrolery obsługujące ścieżki url oraz request'y, gdyż korzystając z technologii ajax można wywołać dany kontroler requestem POST lub GET. Listing prezentujący fragment pliku views.py który wykorzystano w projekcie:

```

1 from PI2.models import Task, DoneTask
2 import PI2
3 from django.template import Context, loader
4 from django.shortcuts import render
5 from django.shortcuts import render_to_response
6 from django.utils import timezone
7 from django.template import RequestContext
8 from PI2.forms import UserForm, CommisionTask
9 from django.contrib.auth import authenticate, login
10 from django.http import HttpResponseRedirect, HttpResponse
11 from django.contrib.auth.decorators import login_required
12 from django.contrib.auth import logout
13 from .forms import CommisionTask
14     from django.http import HttpResponse
15 from datetime import date
16 import parser
17 from django.forms.fields import DateTimeField
18
19 def index(request):
20
21     form = CommisionTask(request.POST)
22     if form.is_valid():
23         Task = form.save(commit=False)
24         Task.userID = request.user
25         Task.save()
26

```

```

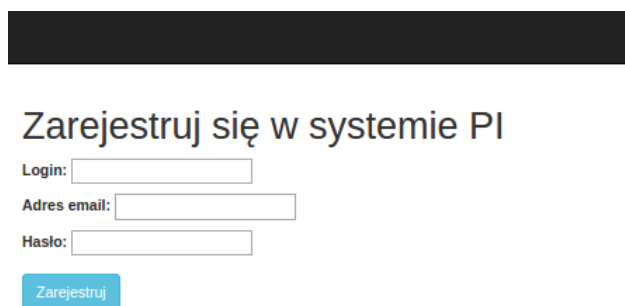
27         #Resetuje formularz po pobraniu poprawnym
28         form =CommisionTask(0);
29
30         return render_to_response('index.html', {'form': form },
31         context_instance=RequestContext(request))
32
33 def wyczyszc(request):
34
35     Task.objects.all().delete()
36     DoneTask.objects.all().delete()
37
38     return HttpResponseRedirect('Czyszczenie bazy wykonane!')
39
40 def odswiez(request):
41
42     """Tutaj mamy pobieranie danych z tabeli i przekazywanie do strony.
43     Pobieramy wszystkie dane z tabeli task czyli .all()
44     filtrujemy po userID = ID oraz sortujemy po dacie dodania
45     malejaco od najmlodszeo czyli descending a odpowiada za to:
46     minus myslnik – przed fieldem czyli rekordem add_date"""
47     form2 = PI2.models.DoneTask.objects.all().filter(userID= request.user.
48     id).order_by('-add_date')
49
50     return render_to_response('odswiez.html', {'form2': form2},
51     context_instance=RequestContext(request))
52
53 ...

```

7.5 Zrzuty ekranów aplikacji internetowej

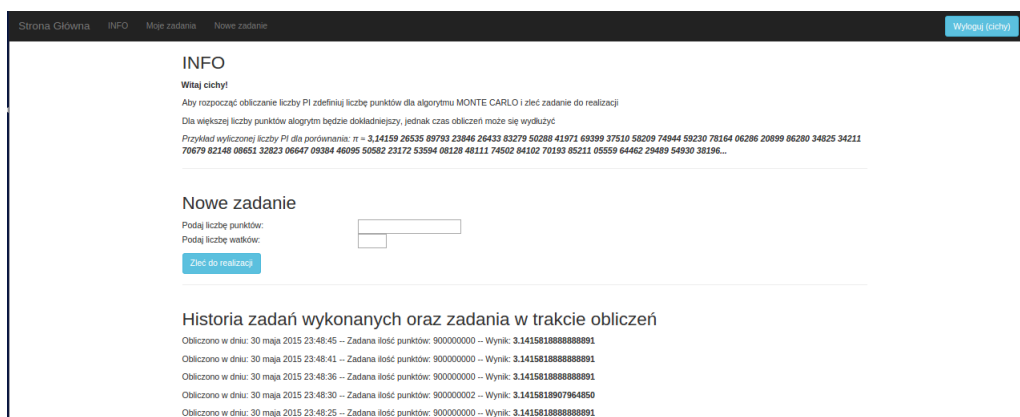


Rysunek 12: Główna strona aplikacji internetowej.



A registration form titled "Zarejestruj się w systemie PI". It features three input fields: "Login:", "Adres email:", and "Hasło:". Below the fields is a blue button labeled "Zarejestruj".

Rysunek 13: Widok rejestracji.

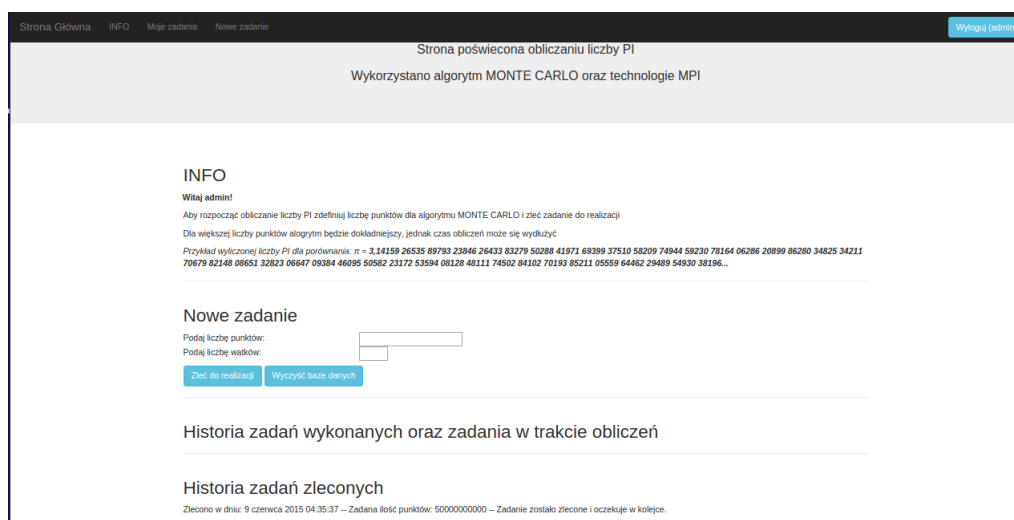


A user dashboard titled "Witaj cichy!". It includes a navigation bar with links: "Strona Główna", "INFO", "Moje zadania", "Nowe zadanie", and a "Wyloguj (cichy)" button. The main content area is divided into three sections:

- INFO**: A welcome message and a list of example numbers for the Monte Carlo algorithm.
- Nowe zadanie**: A form to submit a new task, with fields for "Podaj liczbę punktów:" and "Podaj liczbę ważyków:", and a "Złóż do realizacji" button.
- Historia zadań wykonanych oraz zadania w trakcie obliczeń**: A list of completed tasks with their results.

Obliczono w dniu:	Zadana ilość punktów:	Wynik:
30 maja 2015 23:48:45	900000000	3.1415818888888891
30 maja 2015 23:48:41	900000000	3.1415818888888891
30 maja 2015 23:48:36	900000000	3.1415818888888891
30 maja 2015 23:48:30	900000002	3.1415818907964850
30 maja 2015 23:48:25	900000000	3.1415818888888891

Rysunek 14: Widok zalogowanego użytkownika.



Rysunek 15: Widok zalogowanego użytkownika.

8 Testy.

Aplikację testowaliśmy pod kątem złożoności czasowej w stosunku do liczby punktów. Testy zostały przeprowadzone na 3 stacjach roboczych o podanej konfiguracji:

1. Stacja robocza 1

- Procesor: Core2Quad Q9400 (6M Cache, 2.66 GHz, 1333 MHz FSB)
- Pamięć: DDR3 4GB 1333MHz
- System: Linux Ubuntu 14.04

2. Stacja robocza 2

- Procesor: Core2Duo T6500 (2M Cache, 2.10 GHz, 800 MHz FSB)
- Pamięć: DDR3 4GB 1333MHz
- System: Linux Ubuntu 14.04

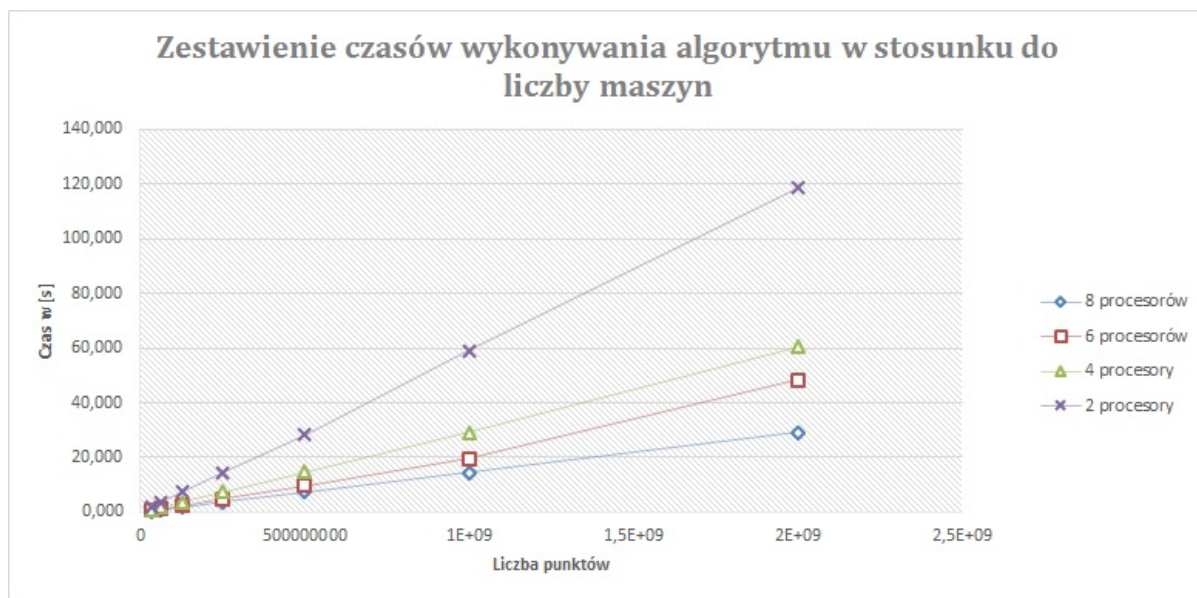
3. Stacja robocza 3

- Procesor: AMD E-450 (1M Cache, 1.65 GHz)
- Pamięć: DDR3 4GB 1333MHz
- System: Linux Ubuntu 14.04

Jak widać z przedstawionych wyników, złożoność obliczeniowa zaimplementowanego algorytmu jest liniowa w stosunku do liczby punktów oraz liczby procesorów. Ze względu

$\begin{matrix} n \\ p \end{matrix}$	n	31250000	62500000	125000000	250000000	500000000	1000000000	2000000000
2	2	1,895774	3,821147	7,422189	14,51247	28,41732	59,147885	118,410756
4	4	0,954118	1,903962	3,710053	7,369633	14,638783	29,25034	60,306934
6	6	0,717063	1,26622	2,472929	4,903922	9,792858	19,457127	48,198792
8	8	0,506648	0,969714	1,865627	3,708315	7,324175	14,615824	29,180241

Rysunek 16: Tabliczyczne zestawienie wyników.



Rysunek 17: Graficzne przedstawienie wyników.

na ograniczenia typu **unsigned int** zakres wpisywanych punktów mieści się w przedziale od 0 do około 2 100 000 000 punktów.

9 Podsumowanie i wnioski.

W ramach kursu Aplikacje Internetowe i Rozproszone został zrealizowany projekt systemu rozproszonego odpowiedzialnego za obliczanie przybliżenia dziesiątego liczby π algorytmem Monte Carlo. System ten składa się z kilku modułów takich jak: silnik obliczeniowy napisany w języku C, aplikacja internetowa z użyciem frameworku języka Python - Django, baza danych MySQL, komunikacja między stacjami roboczymi zrealizowana przy pomocy technologii MPI. Ostatecznie zbudowany system spełnia założenia projektowe i realizuje cele całego kursu.

Podczas rozwoju projektu grupa zdobyła wiedzę z zakresu pracy nad złożonym problemem inżynierskim w środowisku zespołu programistycznego. Do ułatwienia tego zadania wykorzystane zostały narzędzia do komunikacji i współdzielenia zasobów projektu. Mowa tutaj o zdalnym repozytorium opartym o system kontroli wersji Git na platformie GitHub. Planowanie pracy jak i przydział zadań odbywał się natomiast przy wykorzystaniu multimedialnej tablicy na serwisie Trello.

System działa poprawnie i spełnia wszystkie założone funkcjonalności przy respektowaniu pewnych ograniczeń, które zostały zaimplementowane. Użytkownik może maksymalnie wykorzystać 12 wątków w obecnej konfiguracji maszyn roboczych. Ograniczenie uległoby zmianie przy jakimkolwiek zmianie konfiguracji. Innym ograniczeniem jest maksymalna liczba punktów rozpatrywanych podczas pracy algorytmu. Wynika to z architektury programu - zakresu zmiennej przechowującej dane.

W ramach możliwego rozwoju aplikacji należałoby przystosować system, a w szczególności silnik obliczeniowy do przyjmowania większego zakresu punktów oraz optymalizacji wymiany informacji między stacjami roboczymi. Inaczej mówiąc zwiększenie automatyzacji wgrywania oprogramowania silnika na każdą z maszyn.

Komunikacja w zespole odbywała się bez zarzutów o czym może świadczyć uporządkowana tablica na portalu Trello oraz systematyczna praca, którą można odtworzyć na zdalnym repozytorium.

10 Bibliografia

1. Bootstrap Tworzenie Interfejsów Stron WWW - Syed Fazle Rahman
2. Parallel Programming with MPI - Peter Pacheco
3. Documentation of Django - <https://docs.djangoproject.com/en/1.8/>
4. <http://www.zsk.ict.pwr.wroc.pl/zsk/pracownicy/hmaciejewski>
5. Tango with Django Book - https://github.com/leifos/tango-with-django_book