FRACTAL
GENERATOR

Politechnika Wrocławska

Aplikacje Internetowe i Rozproszone

Generowanie fraktali z użyciem klastra obliczeniowego

Jakub Burzała
Krzysztof Cabała
Bartosz Cieśla
Adrian Frydmański
Dawid Gracek
Bartosz Kardas

pod przewodnictwem
dr hab. inż. Henryka Maciejewskiego

2 czerwca 2016

Spis treści

1 Cel projektu **2**

2 Wymagania **2**

2.1 Prototyp – opis ogólny 3

2.2 Prototyp – opis szczegółowy 3

3 System w wersji finalnej **5**

3.1 Baza danych 5

3.2 MPI 6

3.2.1 Master 6

3.2.2 Slave 8

3.2.3 Mandelbrot – generowanie fraktala 9

3.3 Django 10

3.4 Strona internetowa – interfejs użytkownika 13

4 Uruchamianie **19**

4.1 Kompilacja 19

5 Testy wydajności **19**

6 Podsumowanie **30**

1 Cel projektu

Stworzenie aplikacji do generowania fraktali z użyciem technologii internetowych i zrównoleglonych obliczeń na klastrze MPI.

2 Wymagania

Użytkownik aplikacji powinien móc:

1. Wpisać dowolną funkcję zespoloną generującą fraktal
2. Zobaczyć wygenrowany fraktal
3. Przybliżyć obraz w pewnym punkcie płaszczyzny
    - klikając
    - suwakiem
    - zaznaczając pewien obszar
4. Określić:
    - czas trwania animacji
    - liczba klatek na sekundę
    - rozdzielczość
5. Zatrzymać, przewinąć, wznowić animację
6. Wybrać rodzaj animacji. Animowanie względem:
    - przybliżenia – wartość początkowa i końcowa
    - punktu centralnego (ścieżki) – wybranie dwóch punktów na płaszczyźnie (początek i koniec) oraz zdefiniowanie ścieżki pomiędzy nimi, która wskazuje w jaki sposób będzie poruszać się kamera
    - parametrów zespolonych – dowoloność we wprowadzaniu parametrów w równaniu zespolonym. Każdy z nich może być zmieniany z zadanym krokiem podczas animacji
    - kolorystyki – wybór kolorystyki fraktala, definiowanie własnych zasad kolorowania
    - kroków zbieżności – ilość maksymalnej ilość kroków zbieżności, potrzebnych do obliczenia koloru danego piksela
7. Łączyć powyższe animacje:
    - szeregowo – animacje odtwarzają się po sobie
    - równolegle – animacje wykonują się w tym samym momencie
8. Zobaczyć postęp prac przy generacji animacji w postaci paska postępu oraz przewidywany czas zakończenia
9. Zobaczyć skalę przybliżenia w postaci łatwej do wyobrażenia jednostki. Przykładowo punkt początkowy: 100 px odpowiada 1 km. W miarę prybliżania aktualizowanie jednostki do m, cm, mm itd.
10. Zapisać opis animiacji (funkcja, parametry itp) na swoje urządzenie

11. Odczytać wcześniej zapisany opis animacji
12. Zapisać animację na swoje urządzenie
13. Podzielić się wynikiem na Facebooku :)
14. Przygotowanie zadania, bez edytora tylko z pliku konfiguracyjnego
15. Logować się na serwer

Administrator powinien móc:

1. Płynnie przełączać się pomiędzy typem prostym double, a klasą mpf implementującą liczby zmiennoprzecinkowe dowolnej dokładności
2. Wybierać metodę zrównoleglania
    - piksele
    - linie
    - klatki
    - części animacji
3. Wybierać liczbę jednostek wykonawczych

Aplikacja powinna:

1. Zarządzać zadaniami, kolejkować je
2. Sekwencyjnie wykonywać zadania dla wielu użytkowników (z priorytetowaniem)

2.1 Prototyp – opis ogólny

1. Przynajmniej 2 liczące slave'y
2. Może być jednoużytkownikowy system – zlecamy zadanie i trzymamy na serwerze, żeby można było wrócić do niego po jakimś czasie.
3. Wynikiem może być wykonanie fraktala (1 klatki) lub film – dowolność.
4. Zlecenie zadania z np. wczytywaną konfiguracją z pliku, przekazanie do serwera, zakończenie zadania i zwrócenie wyniku użytkownikowi.

2.2 Prototyp – opis szczegółowy

1. Klaster złożony z co najmniej dwóch maszyn, na każdej wykorzystywane wszystkie wątki procesora
2. Interfejs webowy umożliwiający zapamiętanie kolejki zadań
3. Możliwość pobrania wygenerowanego filmu (ew. obrazka)
4. Możliwość wpisania dowolnej funkcji zespolonej
5. Określenie czasu trwania animacji
6. Generowanie animacji na podstawie klatek kluczowych
7. Każda klatka kluczowa opisana współrzędnymi dwóch rogów obrazka

8. Wybór rozdzielczości animacji
9. Wybór między odcieniami szarości a kolorowym obrazkiem
10. Opis sceny wczytany z pliku tekstowego przygotowanego w edytorze tekstu
11. Interfejs webowy bez podziału na użytkowników uruchomiony na komputerze z aplikacją master
12. Kolejka może być obsługiwana przez aplikację odpowiadającą za interfejs webowy

3 System w wersji finalnej

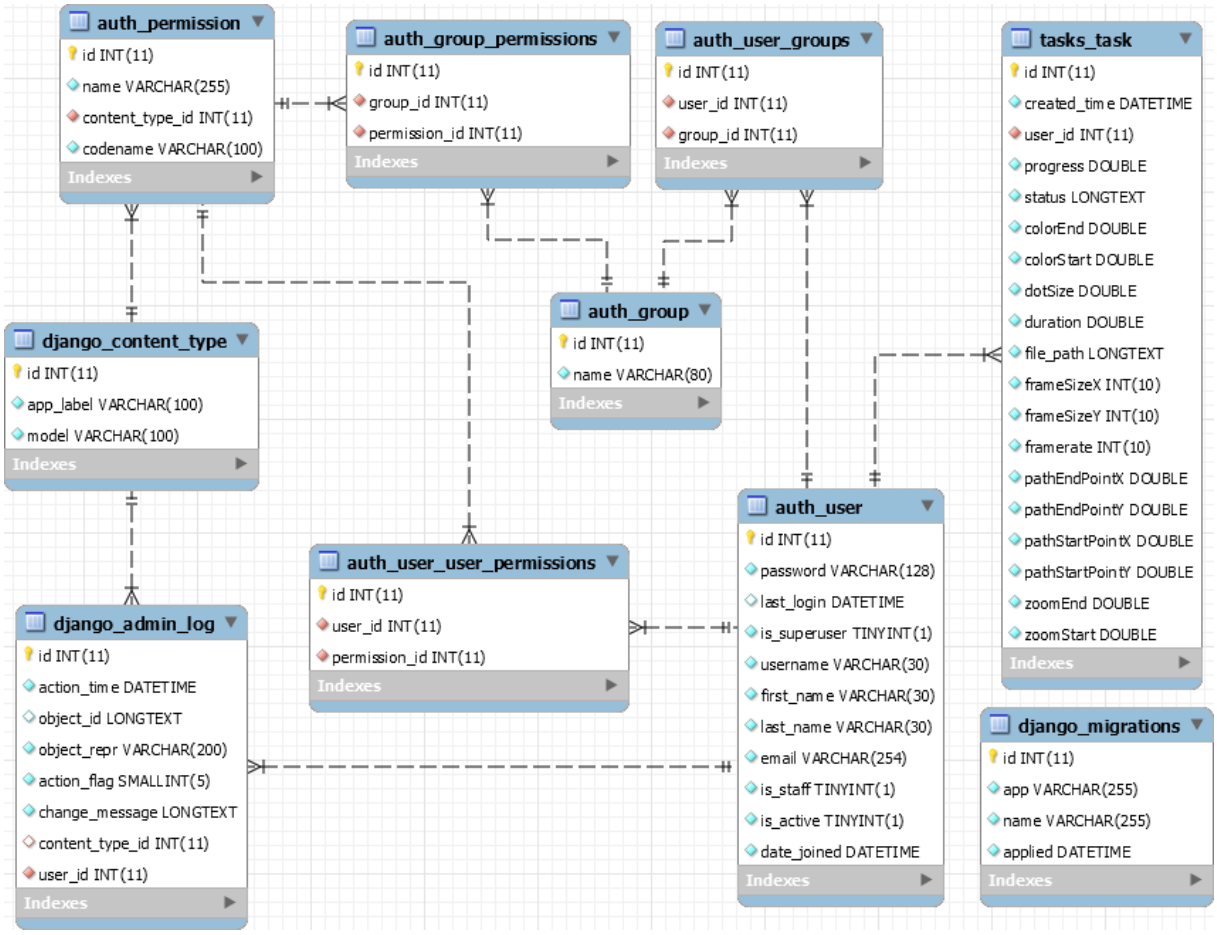
Aplikacja spełnia wymagania. Posiada czytelny interface użytkownika umożliwiający wprowadzanie nowych danych celem wykonania obliczeń. Wynikiem tychże jest animacja, której elementem jest fraktal.

Na działający serwis składa się kilka elementów. Są to:

- Serwer bazodanowy – MySQL Server
- Klaster MPI generujący animacje
- Serwer internetowy

3.1 Baza danych

Baza danych służy do przesyłania zadań pomiędzy aplikacją w Django, a aplikacją rozproszoną oraz do zarządzania użytkownikami systemu. Umieszczona jest na serwerze MySQL. Poniżej widoczny jest jej model fizyczny.



Rysunek 1: Model bazy danych

Kluczową dla zapewnienia funkcjonalności serwisu jest tabela tasks_task. Zawiera ona informację o zadaniach (zarówno parametry i status wykonania, jak i informację o dacie dodania i autorze).
Tabela auth_user zawiera użytkowników. Te i pozostałe tabele obsługuje serwer Django, między innymi na nich opierając swoje działanie.

3.2 MPI

Program co ustalony kwant czasu odpytuję bazę danych, sprawdzając czy są nowe zadania do wykonania. Jeśli znajdzie takowe, zaczyna je liczyć, zaprzęgając do pracy węzły MPI.

Główna funkcja programu jest dość prosta. W zależności od roli, jaką ma pełnić program, uruchamiany jest master lub slave.

```
#include "FractalCalc.h"
#include "Display.h"
#include "Utils.h"
#include "Comunicator.h"
#include "Master.h"
#include "Slave.h"

double zoom = 1.01;
extern bool noDisplay;

int main(int argc, char** argv)
{
  // compareArguments(argc, argv);

  // MPI initialization
    MPI_Init(&argc, &argv);
    registerMPIDataTypes();
    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    // Get the rank of the process
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if(rank == 0)
    {
      Master m;
        system("echo Master: $(hostname)");
      m.work(argc, argv);
    }
    else
    {
      Slave s;
      s.work(argc, argv);
    }

    // Finalize the MPI environment.
    MPI_Finalize();
    return 0;
}
```

3.2.1 Master

Master odpowiada za sprawdzanie bazy danych.

```
MysqlComm com(HOST, USER, PASS, DB);
try
{
  com.Init();
  com.Connect();
  do
```

```
  {
    sleep(1);
    com.Select("tasks_task","*");
    cout<<"Bool: "<<com.AnythingToDo()<<endl;
  }
  while( ! com.AnythingToDo() );
} catch(const char* e){
  printf("%s\n", e);
  fflush(stdout);
}
```

Jeśli znajdzie zadanie, pobiera jego parametry i rozsyła zadanie slave'om. Po odebraniu zadania aktualizowany jest jego status, dzięki czemu na stronie internetowej widać pasek postępu – o którym w dalszej części dokumentacji.

```
Task task = com.GetTask();
Scene s = com.GetScene();
com.TaskStart();
com.PrintRow();

vector<Order> orders(ordersCount);
int orderLength = generateOrders(orders, s, s.frameSize.x * s.frameSize.y / atoi(argv[1])
    );

int ordersPendingCount = ordersCount;
system("echo start_tasks $(date +%X) $(($(date +%s%N)/1000000)) >> time.log");
for(int i = 1; i < world_size; ++i)
{
  sendOrder(orders[ordersCount - ordersPendingCount], i, WORKTAG);
  ordersPendingCount--;
}
map<int, vector<double>> results;
while(ordersPendingCount > 0)
{
  receiveResult(results, MPI_ANY_SOURCE);
  sendOrder(orders[orders.size() - ordersPendingCount], status.MPI_SOURCE, WORKTAG);
  ordersPendingCount--;
  int progress = 100 - (ordersPendingCount*100) / ordersCount;
  com.TaskUpdateProgress(progress);
  cout << progress << endl;
}
```

Slave'y pobierają zadania – pojedyncze klatki do wygenerowania. Wygenerowane obrazy odbierane są od slave'ów:

```
for(; orderID < ordersCount; ++orderID)
{
  if( orderID > 0 && (orderID % ordersPerFrame == 0) )
  {
    stringstream ss;
    ss<<"images/"<<(orderID / ordersPerFrame)<<".bmp";
    image.save_image(ss.str().c_str());
    cout << "IMG "<<ss.str()<<endl;
  }
  for(int i = 0; i < results[orderID].size() / 3; ++i)
  {
```

```
      r = results[orderID][3*i+0];
      g = results[orderID][3*i+1];
      b = results[orderID][3*i+2];

      image.set_pixel(x++, y, r, g, b);
      int tempX = x;
      x = tempX % s.frameSize.x;
      y = (y + (tempX / s.frameSize.x)) % s.frameSize.y;
    }
}
```

Kiedy wszystkie zadania sleve'ów zostaną wykonane, master łączy klatki w film za pomocą programu ffmpeg:

```
stringstream ss;
ss << "ffmpeg -framerate 30 -y -i images/%d.bmp -c:v libx264 -r 30 -pix_fmt yuv420p " <<
    task.id <<".mp4";
system(ss.str().c_str());
```

3.2.2 Slave

Slave jest odpowiedzialny za generowanie fraktala.

```
void Slave::work(int &argc, char** &argv)
{
  int64_t size = 0;
  int tag = 0;
  while(doWork)
  {
    waitForOrder(order);
    doWork = order.doWork;
    size = 0;

    if(true == order.doWork)
    {
      size = executeOrder(order, resultArray);
      if(size > 0)
      {
        int64_t id = order.orderID;
        sendResult(id, resultArray, size);
      }
      else
        doWork = false;
    }
    else
  }
}
```

Wykonywanie zadania na poziomie slave'a odbywa się w metodzie executeOrder, a jego wynik dostępny w vectorze resultArray (polu klasy Slave).

```
int64_t Slave::executeOrder(Order &order, vector<double> &resultArray)
{
  int64_t size = 0;
```

```
    if(order.count > 0)
    {
        resultArray.resize(order.count);
        size = FractalCalc::calcMandelbrotPart(resultArray.data(), order);
        vector<double> colorArray(3*size);
        int x = order.beginX;
        int y = order.beginY;
        for(int i = 0; i < order.count; ++i)
        {
            double color = resultArray[i];
            double r, g, b;
            r = 0.5*(sin(2*M_PI*color - M_PI/2 + M_PI /3) + 1) * 255.0;
            g = 0.5*(sin(2*M_PI*color - M_PI/2) + 1) * 255.0;
          b = 0.5*(sin(2*M_PI*color - M_PI/2 - M_PI / 3) + 1) * 255.0;
              colorArray[3*i + 0] = r;
              colorArray[3*i + 1] = g;
              colorArray[3*i + 2] = b;
        int tempX = ++x;
        x = tempX % order.pictureWidth;
        y = (y + (tempX / order.pictureWidth)) % order.pictureHeight;

        }
        resultArray.swap(colorArray);
        size = resultArray.size();
    }
      return size;
}
```

3.2.3 Mandelbrot – generowanie fraktala

Slave wywołuje metodę liczącą fraktal – calcMandelbrotPart – z klasy FractalCalc.

```
class FractalCalc
{
public:
  typedef void (* FractalFnPtrD)(complex<double>&, complex<double>&);
  static const int convergenceSteps;
  static const double divergenceLimitD;
  //double
  static void mandelbrotFractal(complex<double> &z, complex<double> &c);
  static double getConvergence(double z_real, double z_imag, double c_real, double c_imag
    , FractalFnPtrD fn);
  static int calcMandelbrotPart(double* mandelbrot, Order &order);
};

void FractalCalc::mandelbrotFractal(complex<double> &z, complex<double> &c)
{
  z = z * z + c;
}

double FractalCalc::getConvergence(double z_real, double z_imag, double c_real, double
    c_imag, FractalFnPtrD fn)
{
  complex<double> z(z_real, z_imag);
  complex<double> c(c_real, c_imag); // c_real => c(e)real xD

  // Sprawdzanie zbieznosci
```

```
  int j = 0;
  for (; j < convergenceSteps && abs(z) < divergenceLimitD; j++)
  {
    fn(z, c);
  }
  double log_zn, nu, result;
  if ( j < convergenceSteps )
  {
    log_zn = log( abs(z) );
    nu = log( log_zn / log(2) ) / log(2);
    result = (j + 1 - nu) / convergenceSteps;
  }
  return result;
}

int FractalCalc::calcMandelbrotPart(double* mandelbrot, Order &order)
{
  int iX = order.beginX;
  int iY = order.beginY;
  int i = 0;
  for (; i < order.count; ++i, ++iX)
  {
    if(iX >= order.pictureWidth)
    {
      iX = 0;
      ++iY;
    }
    double dX = (order.fractalX - order.pictureWidth/2 * order.dotSize) + order.dotSize*
    iX;
    double dY = (order.fractalY - order.pictureHeight/2 * order.dotSize) + order.dotSize*
    iY;
    mandelbrot[i] = getConvergence(0, 0, dX, dY, mandelbrotFractal);
  }
  return i;
}
```

3.3 Django

Aplikacja napisana z użyciem frameworku Django jest kolejnym ważnym elementem systemu. To ona odpowiada za komunikację użytkownik – baza danych.

Widoki dostępne w interfejsie użytkownika, to strona główna, logowanie, rejestracja, dodawanie nowego fraktala i przeglądanie już dodanych w postaci wygenerowanych animacji. Poniżej umieszczono przykładowy szablon strony głównej i strony Moje animacje:

```
{% load staticfiles %}
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>{{ site.name }}</title>
<meta name="keywords" content="" />
<meta name="description" content="" />
<link href="http://fonts.googleapis.com/css?family=Varela" rel="stylesheet" />
<link href="{% static "default.css" %}" rel="stylesheet" />
<link href="{% static "fonts.css" %}" rel="stylesheet" />
<!--
<script src="//code.jquery.com/jquery-1.11.1.min.js"></script>
```

```
<script src="{% static "js/test.js" %}"></script>
-->
<!--[if IE 6]><link href="{% static "default_ie6.css" %}" rel="stylesheet" type="text/css" /><![endif]-->
</head>
<body>
<div id="wrapper">
  <div id="header-wrapper">
    <div id="header" class="container">
      <div id="logo">
        <h2><a href="{% url 'index' %}">FRACTAL<br />GENERATOR</a></h2>
      </div>
            {% block menu %}
            {% endblock %}
    </div>
  </div>
  <div id="extra" class="container">
        {% block reg_log %}
            {% if user.is_anonymous %}
                {% block main %}
                {% endblock %}
            {% endif %}
        {% endblock %}
        {% if user.is_authenticated %}
        {% block content %}
            {% endblock %}
        {% endif %}
  </div>
</div>
<div id="copyright" class="container">
  <p>&copy; Untitled. All rights reserved. | Photos by <a href="http://fotogrph.com/">Fotogrph</a> | Design by <a href="http://templated.co" rel="nofollow">TEMPLATED</a>.</p>
</div>
</body>
</html>
```

```
{% extends "tasks/index.html" %}
{% load staticfiles %}
{% block menu %}
    <div id="menu">
        <ul>
          <li><a href="{% url 'index' %}" accesskey="1" title="">Strona glowna</a></li>
                    {% if user.is_authenticated %}
                <li><a href="{% url 'generate' %}" accesskey="2" title="">Generuj fraktal</a></li>
                <li><a href="{% url 'fractals' %}" accesskey="3" title="">Moje fraktale</a></li>
                <li class="current_page_item"><a href="{% url 'animations' %}" accesskey="4" title="">Moje animacje</a></li>
                <li><a href="{% url 'auth_logout' %}" accesskey="5" title="">Wyloguj</a></li>
                    {% else %}
                        <li><a href="{% url 'auth_login' %}" accesskey="2" title="">Zaloguj</a></li>
                    {% endif %}
        </ul>
```

```
      </div>
{% endblock %}
{% block content %}
    <h1>Moje Animacje</h1>
    <br /><br />
    {% if videos.count > 0 %}
        {% for video in videos %}
            <h3>{{ forloop.counter }}.   {{ video.created_time }}</h3>
            <video width="320" height="240" controls>
                <source src="{% static "videos/"%}{{  video.file_path }}" type="video/mp4
    ">
                Your browser does not support the video tag.
            </video>
        {% endfor %}
    {% else %}
        Nie znaleziono zadnych animacji wygenerowanych przez Ciebie.
    {% endif %}
    <script src="//code.jquery.com/jquery-1.11.1.min.js"></script>
<script src="{% static "js/get_tasks_progress.js" %}"></script>
{% endblock %}
{% block main %}
    <h1>Strona Glowna</h1>
    Zaloguj sie aby rozpoczac korzystanie z serwisu. Jezeli nie masz jeszcze konta <a
    href="{% url 'registration_register' %}">zarejestruj sie!</a>
{% endblock %}
```

Formularz rejestracji generuje Django.

```
{% extends "tasks/tasks.html" %}
{% load i18n %}

{% block reg_log %}
        <form method="post" action=".">
            {% csrf_token %}
            {{ form.as_p }}
            <label> </label><input type="submit" value="{% trans 'Submit' %}"/>
        </form>
{% endblock %}
```

Widoki w pliku views.py:

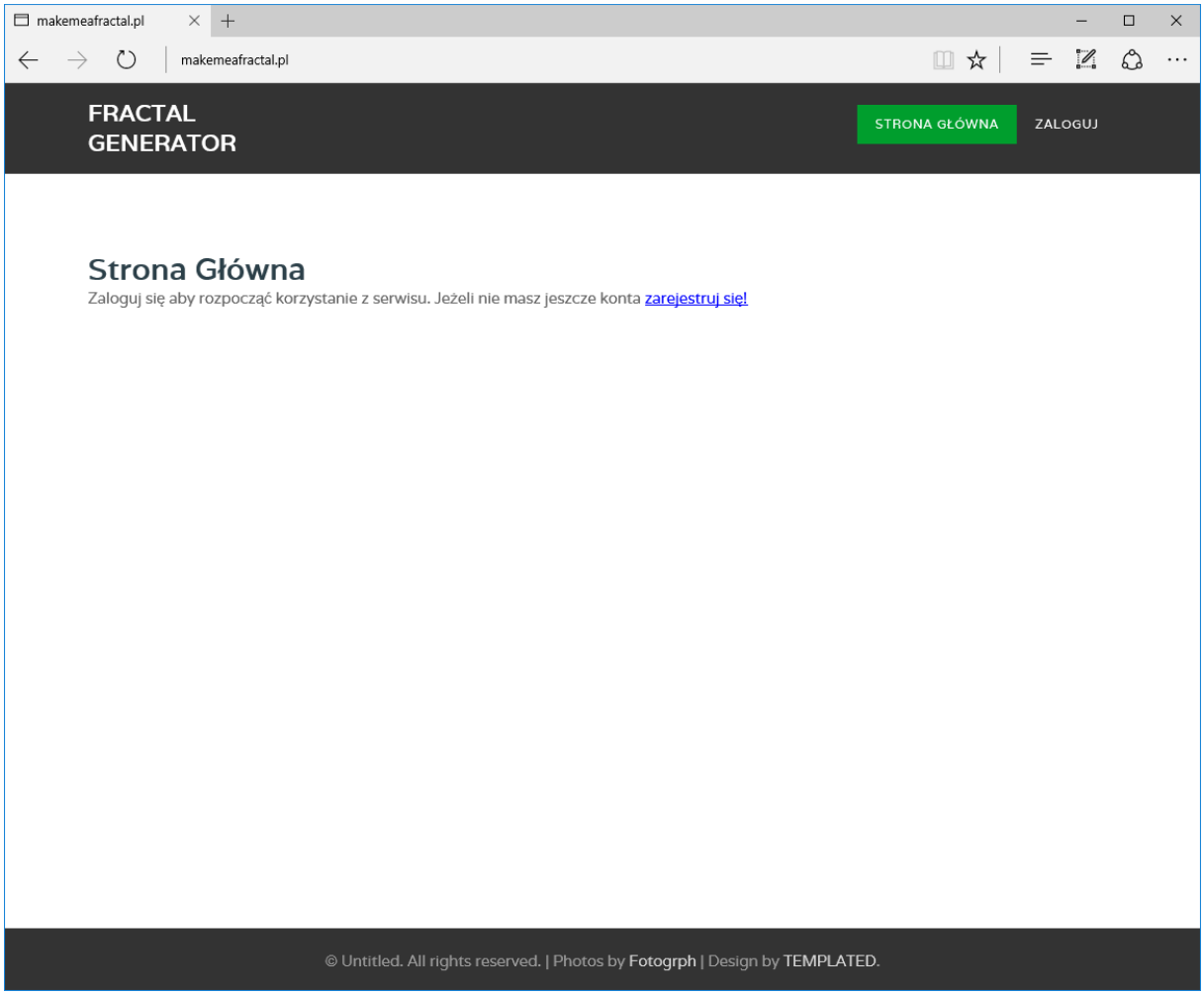
```
from django.shortcuts import redirect, render
from django.contrib.auth.decorators import login_required
from .forms import AddTaskForm
from .models import Task

def get_dotsize(value):
    return 4.0 / value

def index(request):
    if request.user.is_authenticated():
        tasks = Task.objects.filter(user=request.user).order_by('-created_time')
        return render(request, 'tasks/tasks.html', {'tasks': tasks})
    else:
        return render(request, 'tasks/tasks.html', {})

@login_required(login_url='/')
```

```
def add(request):
    if request.method == "POST":
        form = AddTaskForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.user = request.user
            post.dotSize = get_dotsize(post.frameSizeX)
            post.save()
            return redirect('/', pk=post.pk)
    else:
        form = AddTaskForm()
    return render(request, 'tasks/addTask.html', {'form': form})

@login_required(login_url='/')
def delete(request, task_id):
    Task.objects.filter(user_id=request.user.id, id=task_id).delete()
    return redirect('/')

@login_required(login_url='/')
def fractals(request):
    return render(request, 'tasks/fractals.html', {})

@login_required(login_url='/')
def animations(request):
    if request.user.is_authenticated():
        tasks = Task.objects.filter(user=request.user, status='done').order_by('-
    created_time')
        return render(request, 'tasks/animations.html', {'videos': tasks})
    else:
        return render(request, 'tasks/animations.htmll', {})
```

Widoki, modele, baza danych, gotowy komponent do rejestracji i logowania, kody z Django, przykładowa templatka.

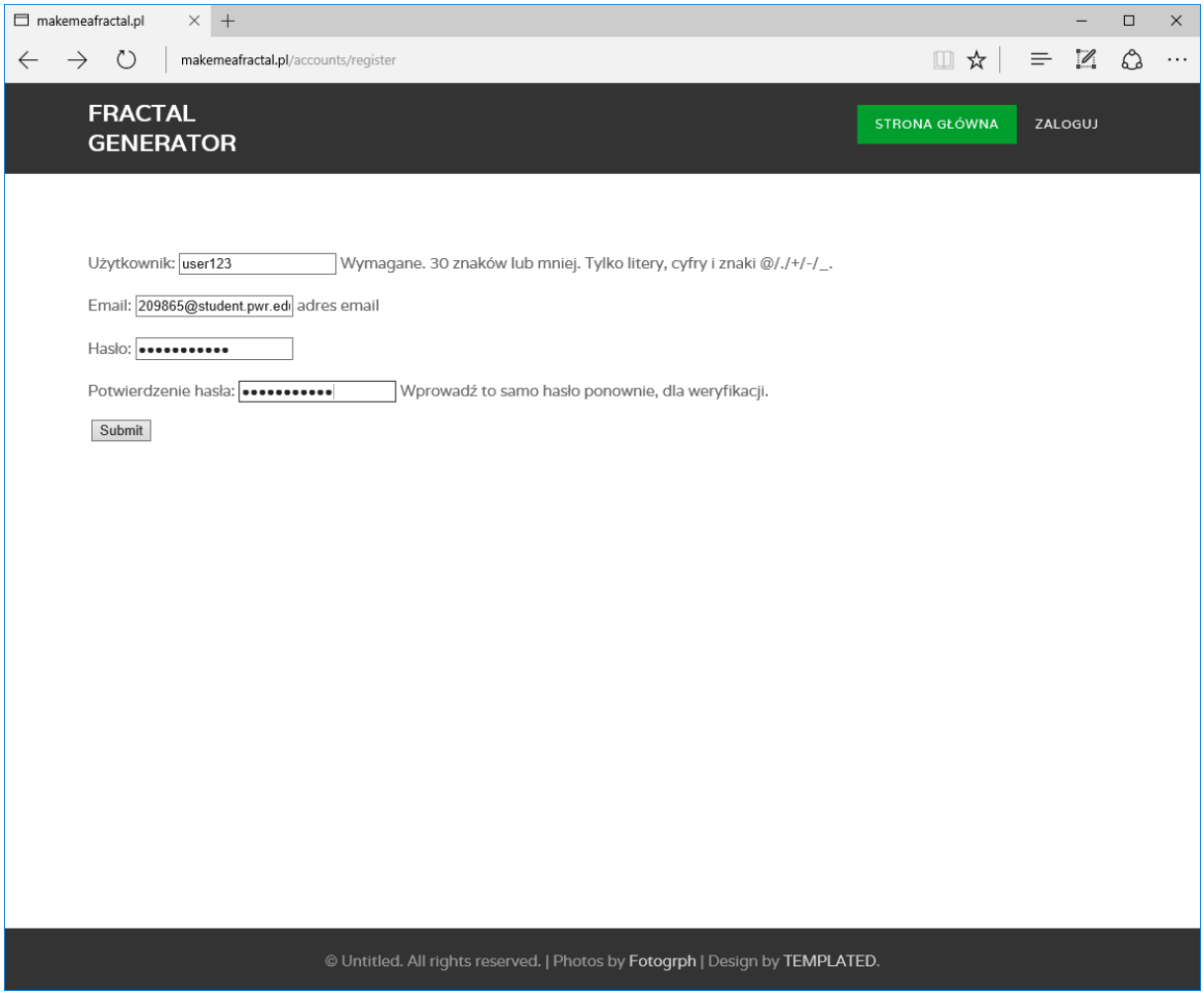
3.4 Strona internetowa – interfejs użytkownika

Poniższe zrzuty ekranu prezentują działanie serwisu:



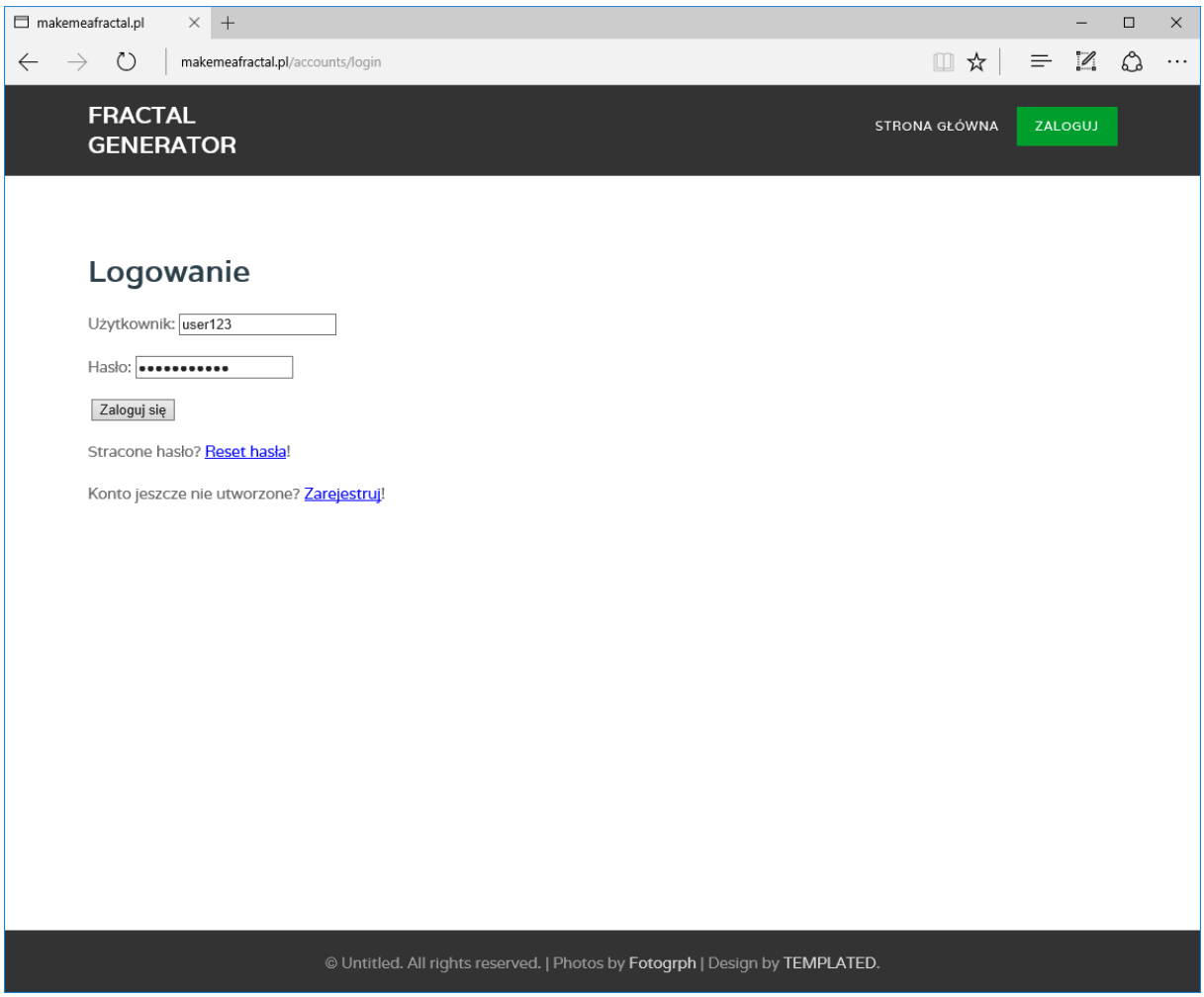
Rysunek 2: Strona główna

Na stronie głównej widoczna jest propozycja zalogowania się, lub rejestracji. Bez tego niemożliwe jest wysyłanie zadań do wykonania.

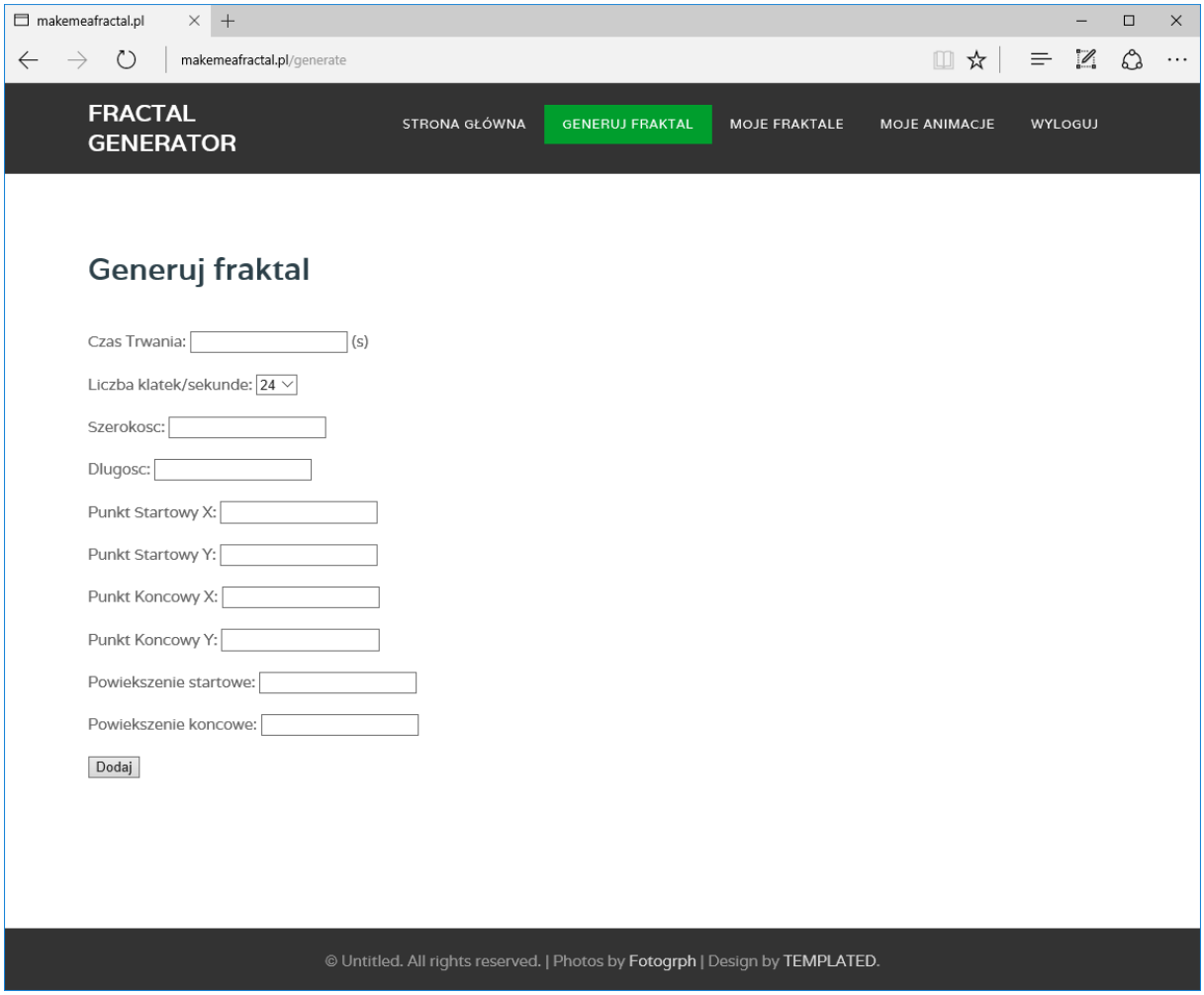


Rysunek 3: Rejestracja

Rejestracja wymaga podania nazwy użytkownika, adresu email i hasła. Na podany adres zostaje wysłany link aktywacyjny, po kliknięciu którego można korzystać z serwisu.

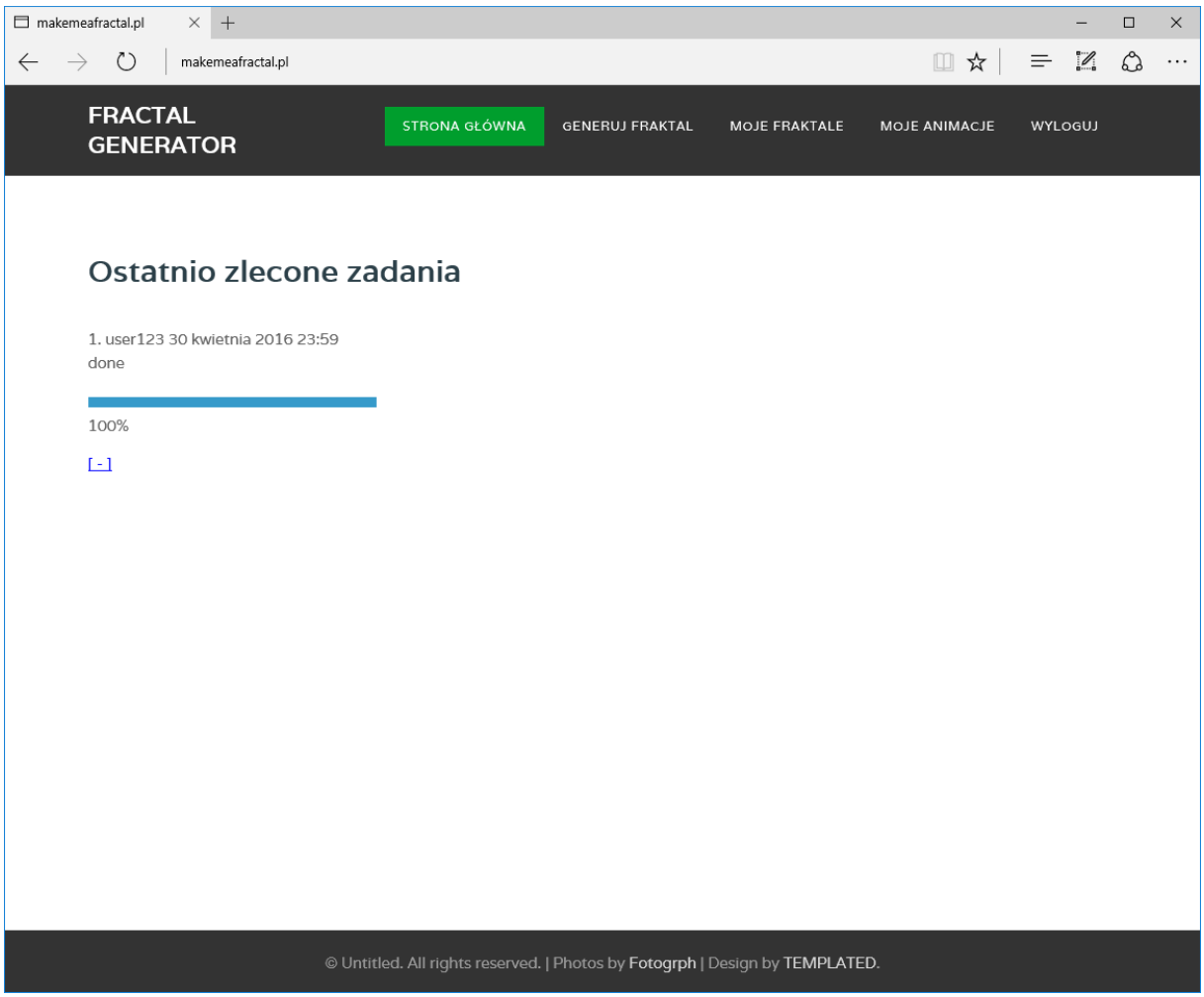


Rysunek 4: Logowanie



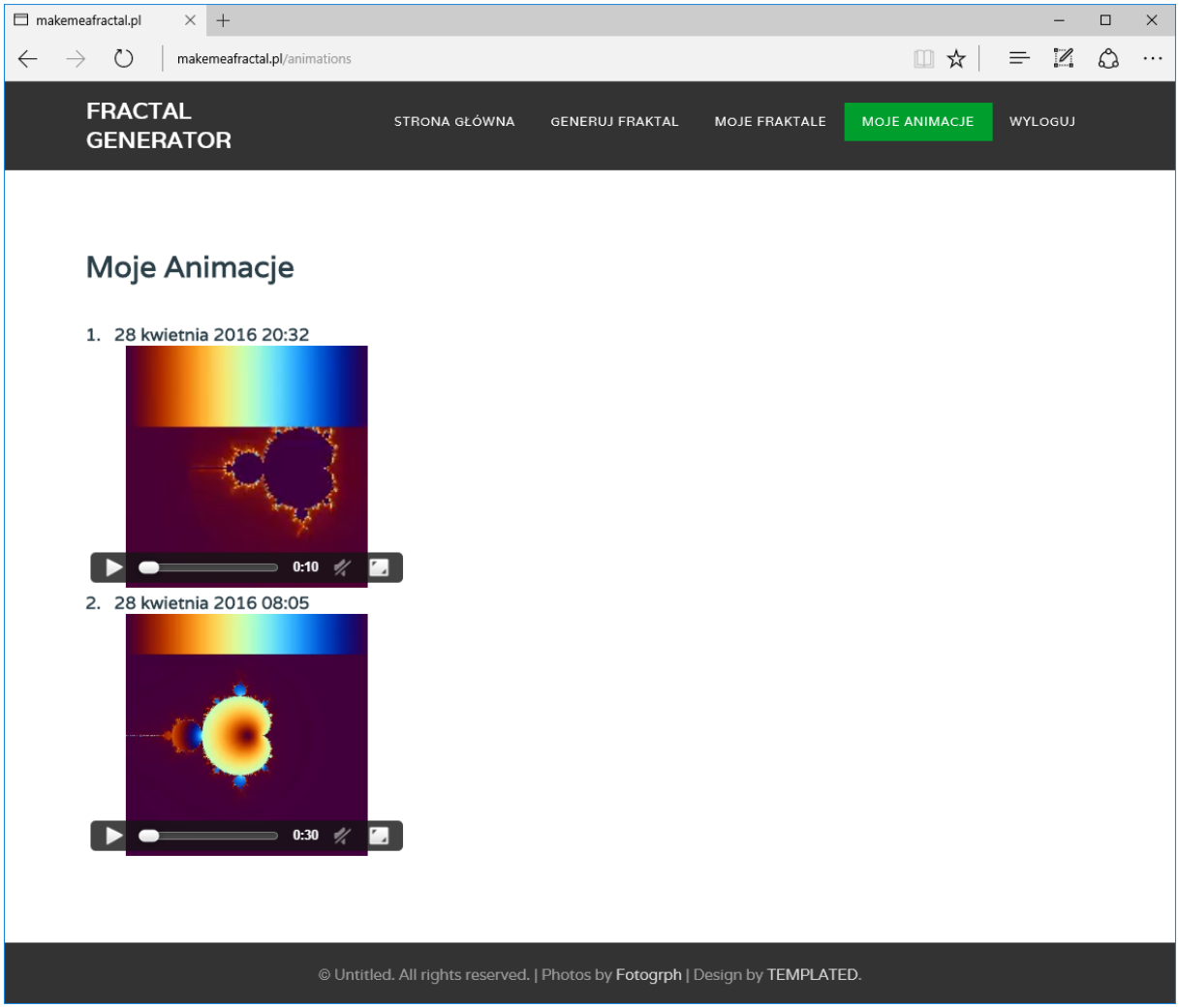
Rysunek 5: Generowanie fraktala

Na tej stronie istnieje możliwość dodania nowego zadania do wykonania. Podawane są parametry animacji - dotyczące zarówno pliku wideo, jak i samego fraktala i jego pozycji.



Rysunek 6: Kolejka zadań

Po zalogowaniu się na stronie głównej widoczne są zlecone zadania i ich status wykonania wraz z paskiem postępu.



Rysunek 7: Moje animacje

Na stronie Moje animacje widoczny jest podgląd wygenerowanych animacji.

4 Uruchamianie

W celu uruchomienia aplikacji potrzebne są następujące składniki:

- lib mysql
- libctdc++6
- serwer ssh
- authorizedkeys
- rsync

4.1 Kompilacja

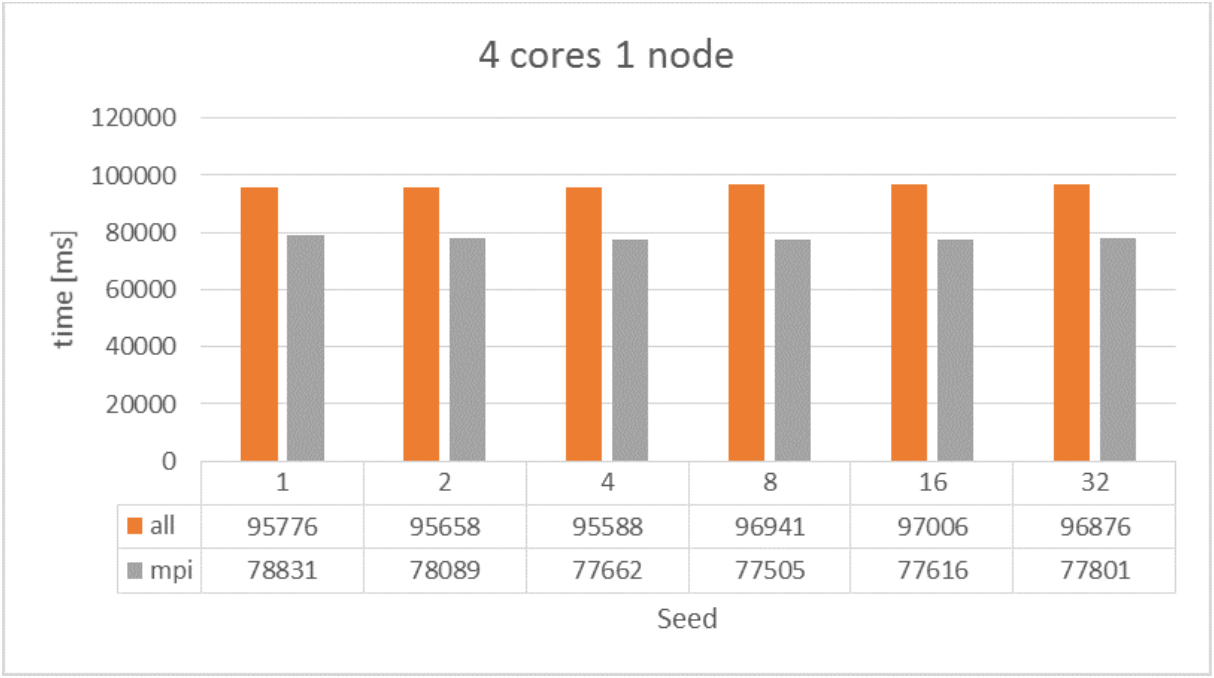
5 Testy wydajności

Testy zostały wykonane przy użyciu trzech fizycznych maszyn. 2 z nich były wyposażone w 2 (intel i5), a jedna 4 (intel i7) rdzenie procesora. Zastosowana w nich technologia HyperTreading pozwala na jednoczesną pracę dwóch wątków na jednym rdzeniu, zatem można stwierdzić, że były dostępne odpowiednio 4 i

8 rdzeni Maszyna na któtej uruchamiany był master pracowała pod kontrolą systemu operacyjnego Linux Mint, natomiast na pozostałych były zainstalowane maszyny wirtualne z systemem Linux Debian. Komputery były spięte do jednego switcha z portami FastEthernet. Dostępny sprzęt pozwolił przetestować następujące konfiguracje:

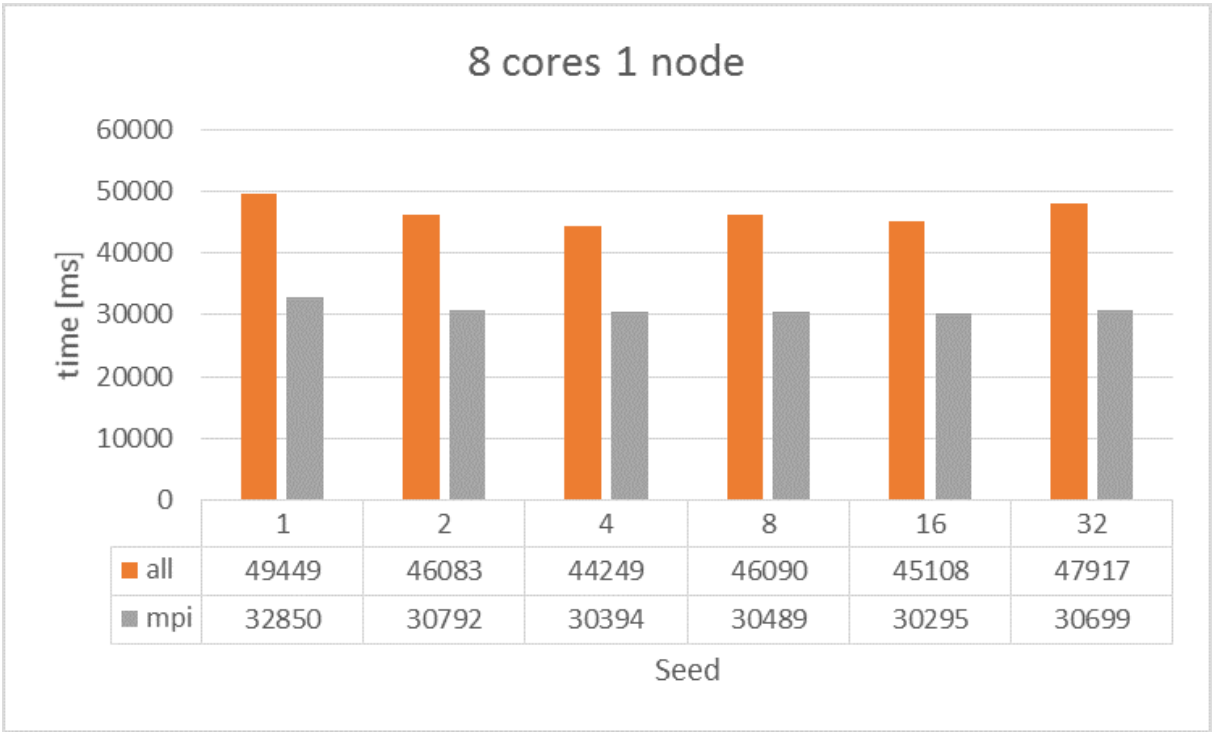
1. 1 maszyna fizyczna, 4 rdzenie
2. 1 maszyna fizyczna, 8 rdzeni
3. 2 maszyny fizyczne, 8 rdzeni (4 + 4)
4. 2 maszyny fizyczne, 12 rdzeni (4 + 8)
5. 3 maszyny fizyczne, 12 rdzeni (4 + 4 + 4)
6. 3 maszyny fizyczne, 16 rdzeni (4 +4 +8)

Na wszystkich maszynach zostało uruchomione zadanie wygenerowania 1-sekundowej animacji (30 klatek) w różnej rozdzielczości (400x400 – 2000x2000 px ze skokiem co 200 px). Drugi scenariusz testowy dotyczył podziału klatki na mniejsze podzadania. W tym przypadku rozdzielczość została ustalona na 2000 x 2000 px, natomiast klatka była dzielona na 1, 2, 4, 8, 16 podzadań (seed). Wszystkie pomiary zostały wykonane 10-krotnie, a następnie uśredniono wyniki. Widoczny na wykresach przypadek all dotyczył czasu wykonania całego zadania, wraz z generowaniem animacji z gotowych klatek, natomiast mpi dotyczy wyłącznie czasu obliczeń zrównoleglonych. Wykresy na których nie ma tego podziału pokazują czasy tylko obliczeń zrównoleglonych.

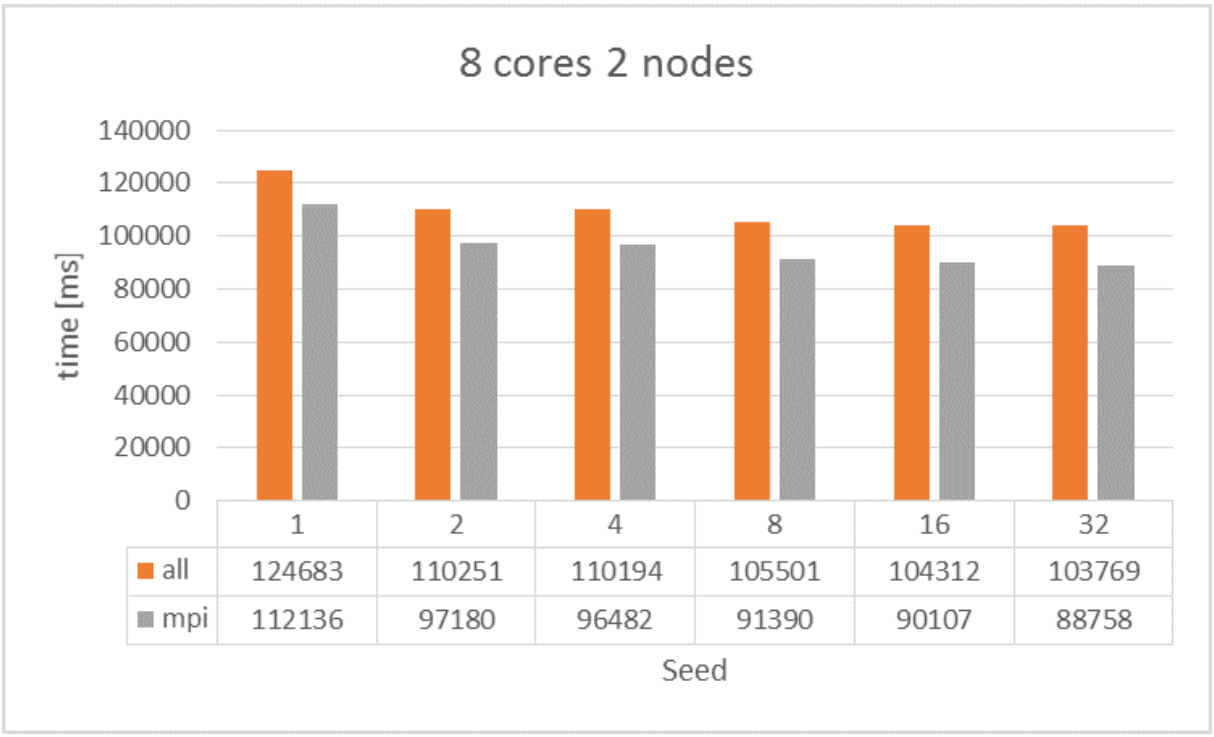


	1	2	4	8	16	32
all	95776	95658	95588	96941	97006	96876
mpi	78831	78089	77662	77505	77616	77801

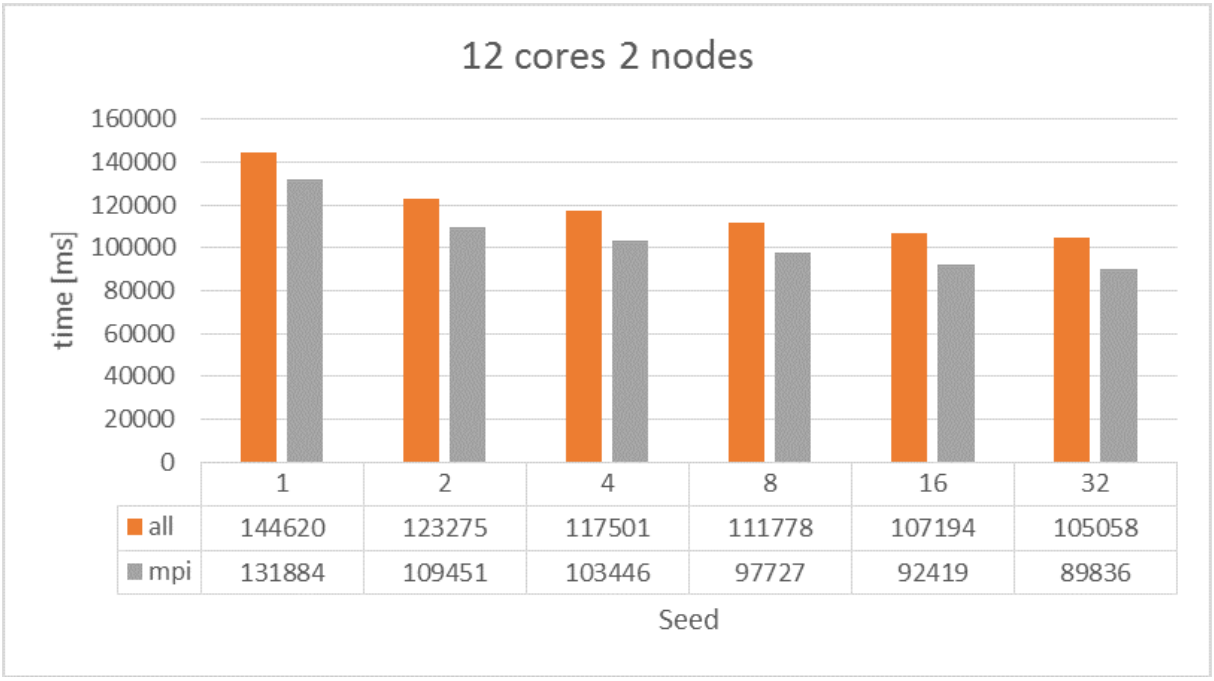
Rysunek 8: 4 rdzenie, 1 węzeł



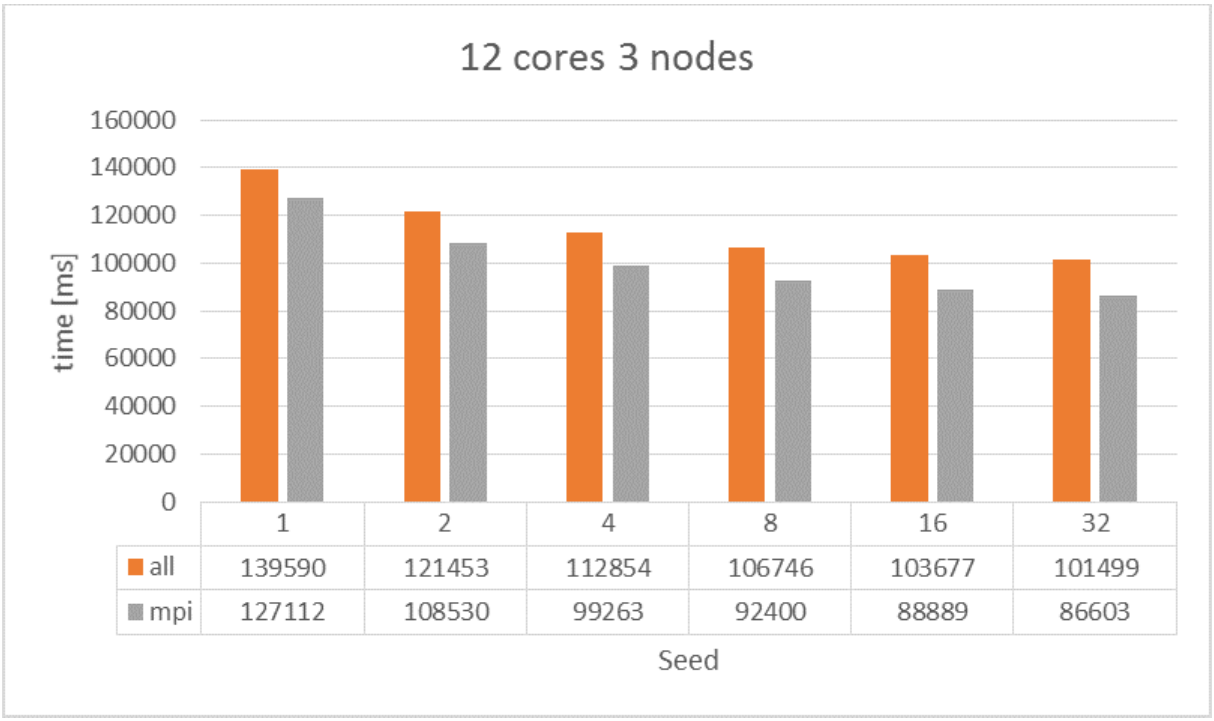
Rysunek 9: 8 rdzeni, 1 węzeł



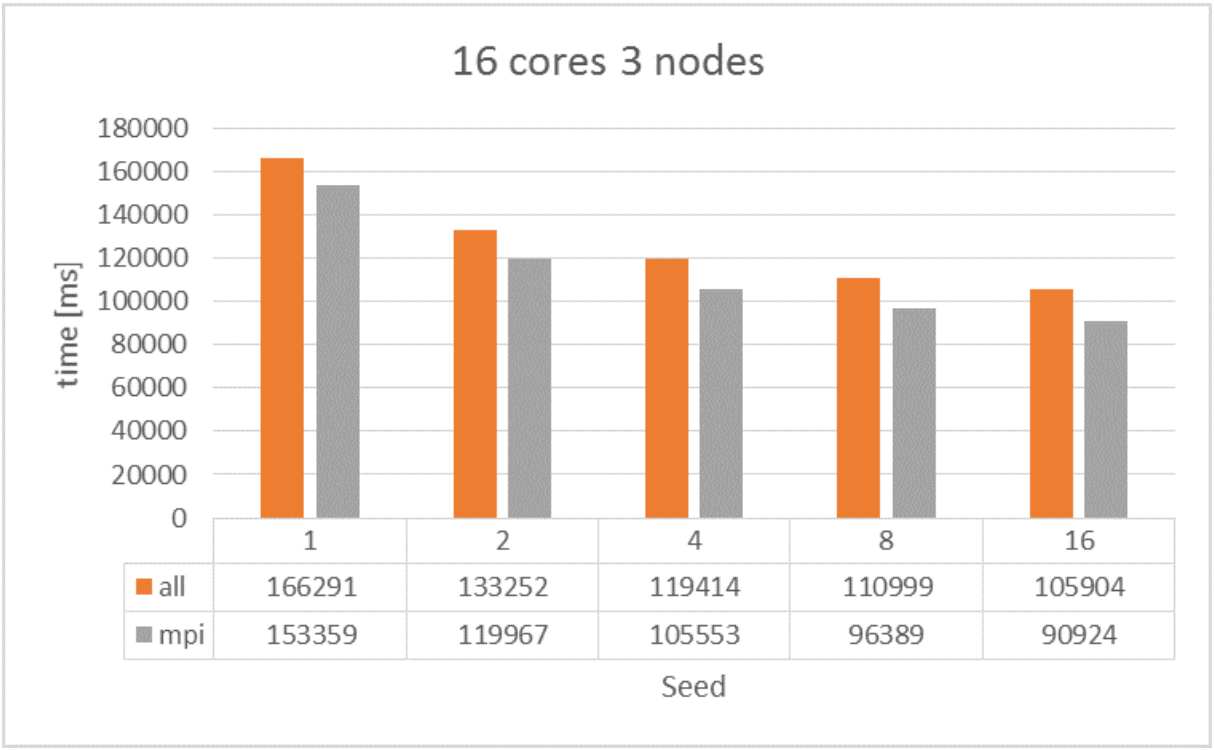
Rysunek 10: 8 rdzeni, 2 węzły



Rysunek 11: 12 rdzeni, 2 węzły

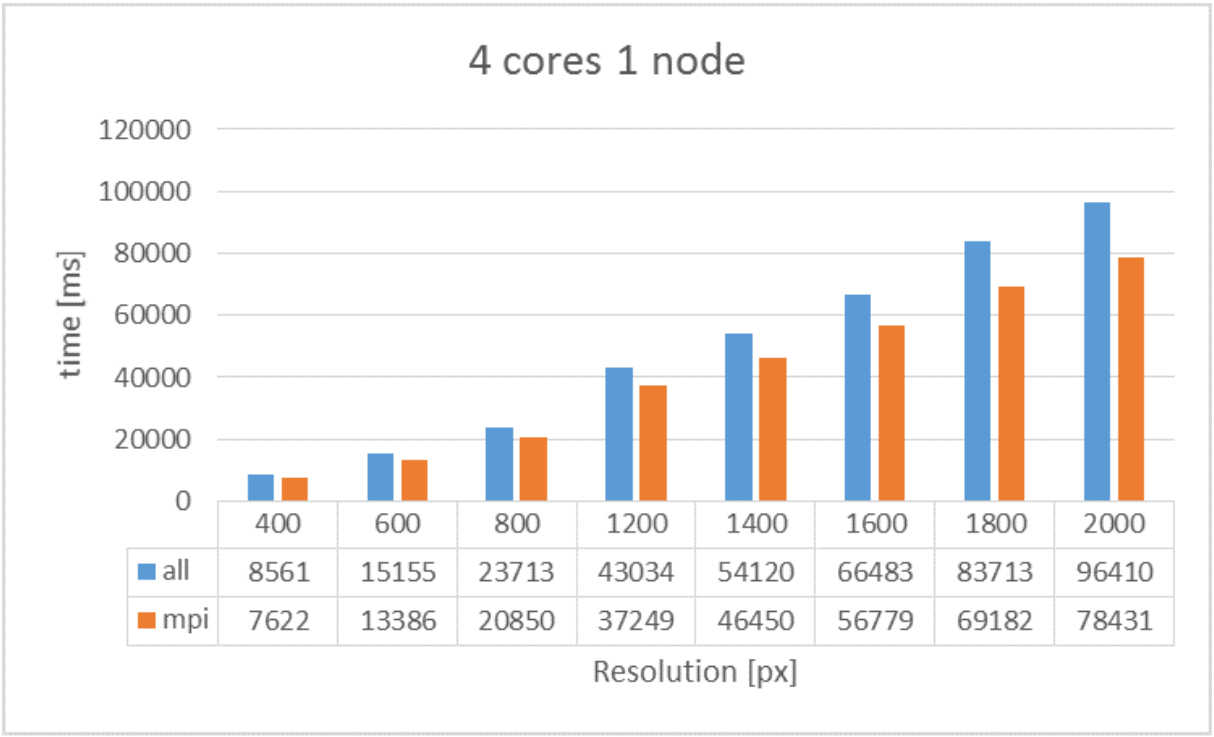


Rysunek 12: 12 rdzeni, 3 węzły

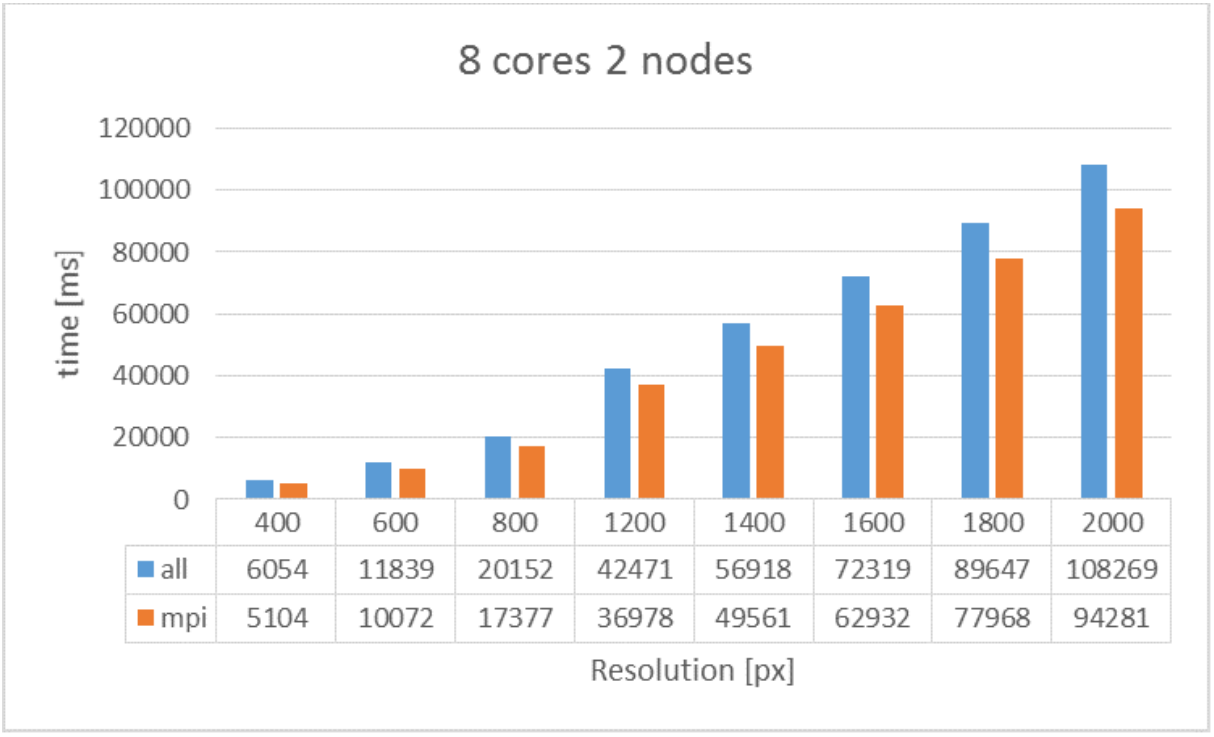


Rysunek 13: 16 rdzeni, 3 węzły

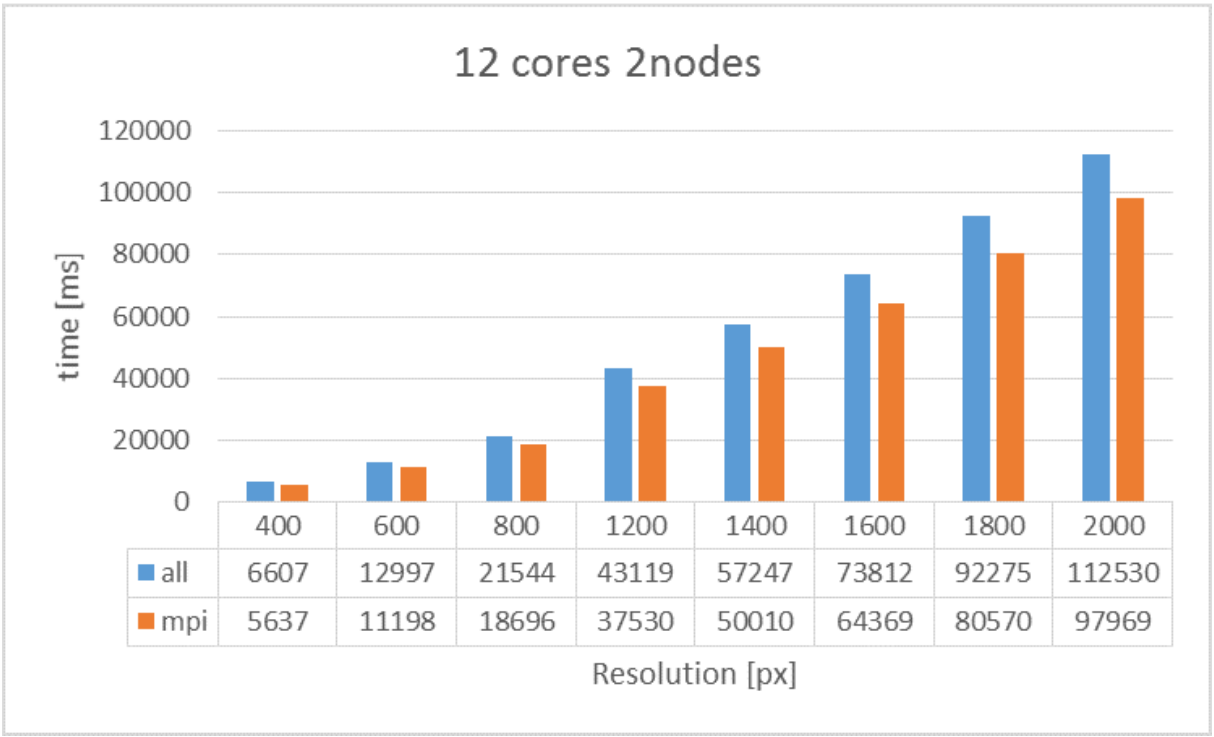
Z powyższych wykresów widać, że jeśli wszystkie rdzenie liczące znajdują się na jednej maszynie to podział zadania na różne fragmenty nie ma większego wpływu na czas obliczeń. Jeżeli jednak zwiększymy ilość maszyn liczących, to podział zadania na mniejsze pozwala zaobserwować spadek czasu obliczeń. Różnica ta zwiększa się wraz z dokładaniem do klastra kolejnych maszyn. Również ilość przydzielonych rdzeni procesora ma znaczący wpływ na szybkość omawianej zmiany.



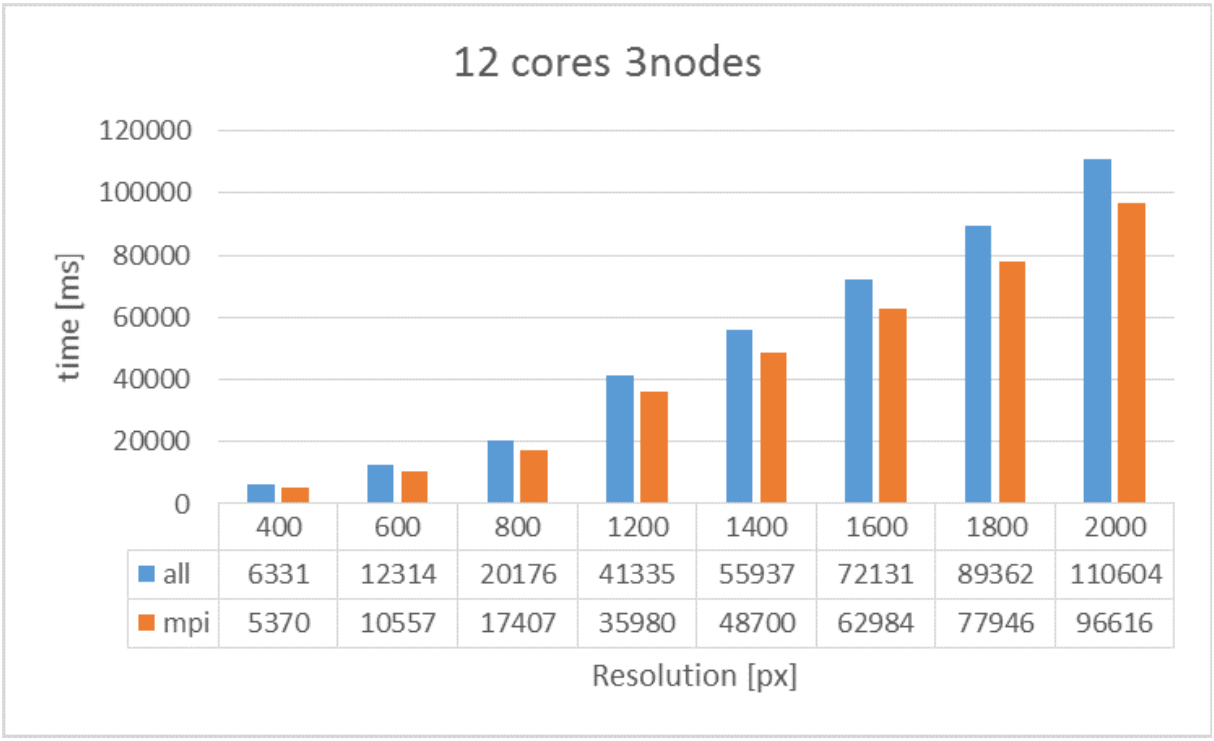
Rysunek 14: 4 rdzenie, 1 węzeł



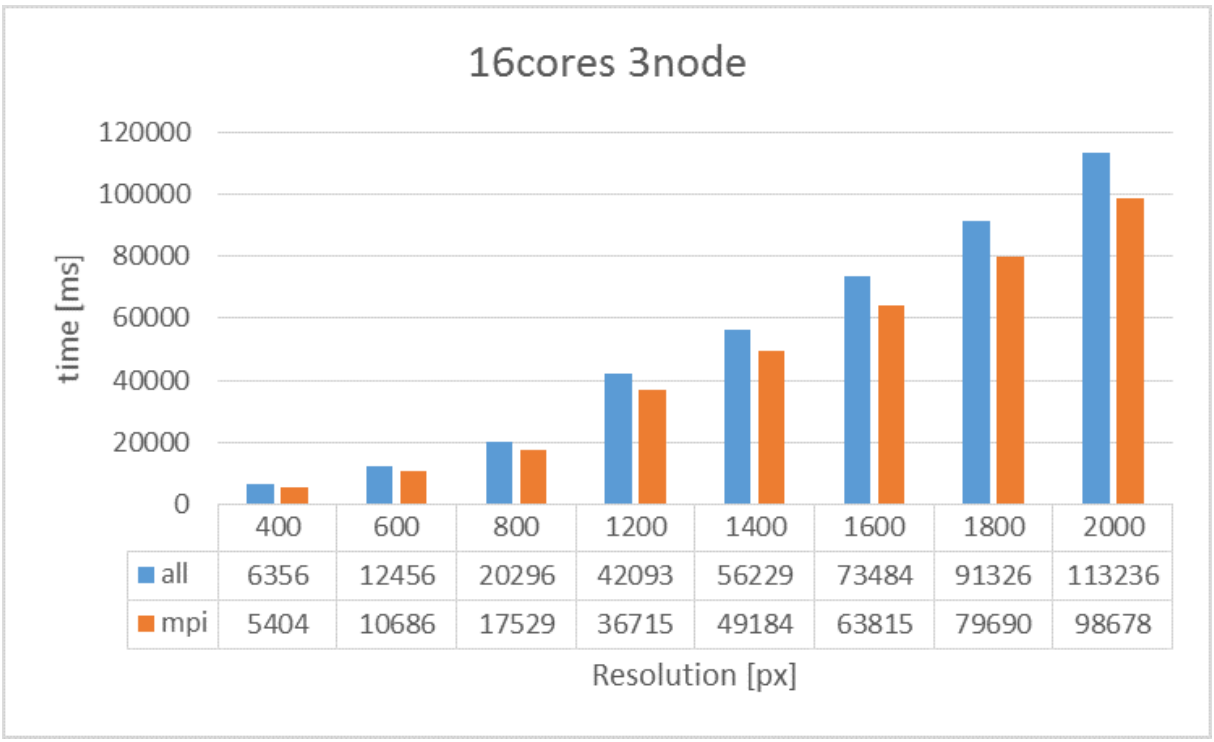
Rysunek 15: 8 rdzeni, 2 węzły



Rysunek 16: 12 rdzeni, 2 węzły



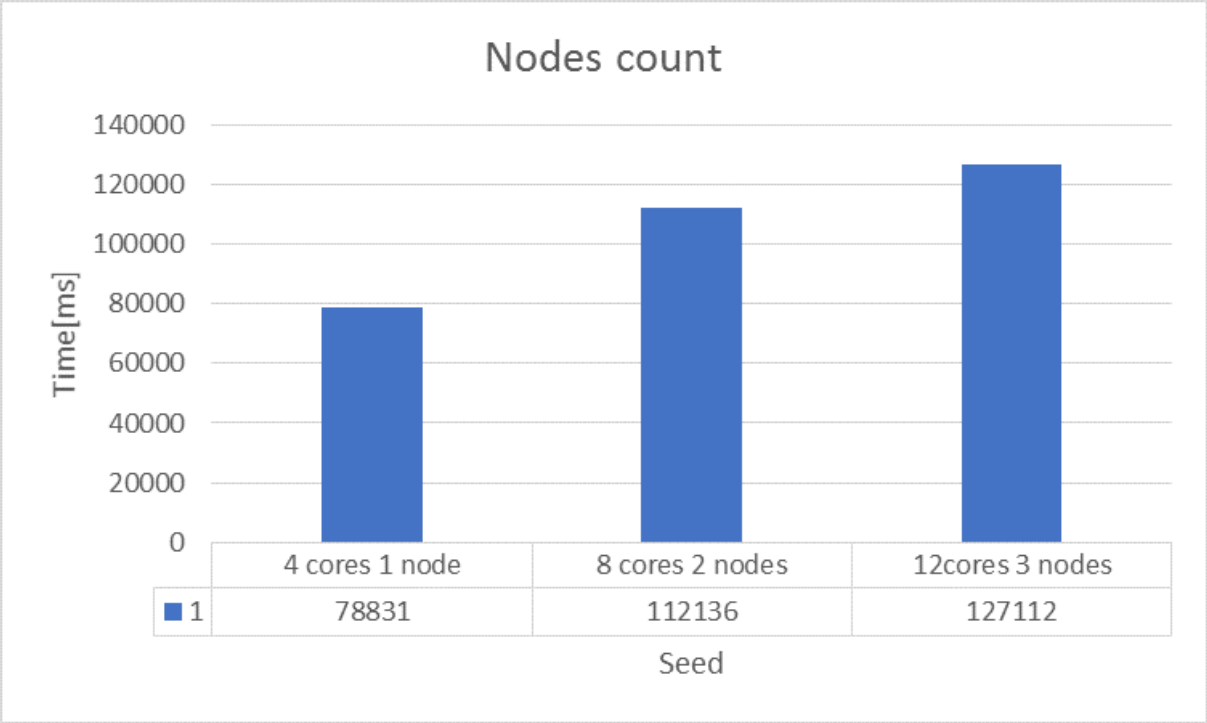
Rysunek 17: 12 rdzeni, 3 węzły



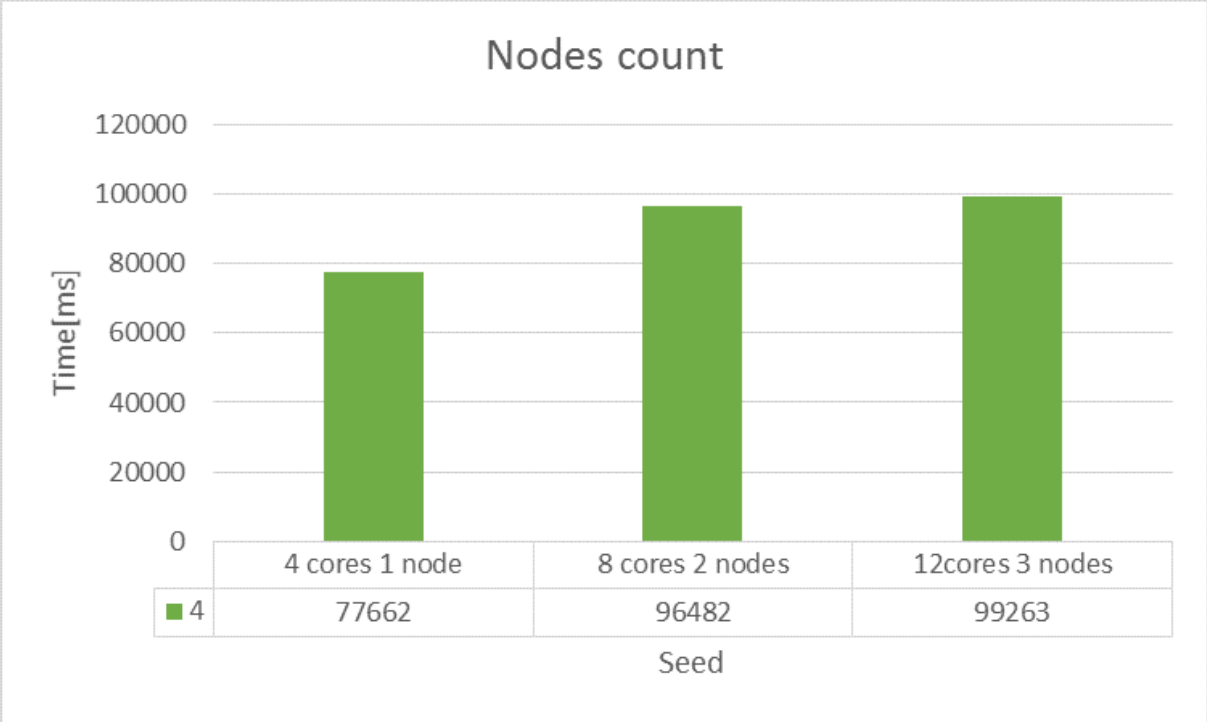
Rysunek 18: 16 rdzeni, 3 węzły

Porównanie czasu obliczeń w zależności od rozmiaru kaltki obrazu ukazuje wyniki zgodne z oczekiwaniami. Ilość maszyn liczących i rdzeni procesora nia ma wpływu na charakter zmiany czasu obliczeń. We wszystkich przetestowanych przypadkach jest to zależność wielomianowa, zbliżona do funkcji kwadratowej. Ma to uzasadnienie w zmianie rozmiarze zadania – dwukrotne zwiększenie boku klatki powoduje czterokrotne zwiększenie rozmiaru zadania.

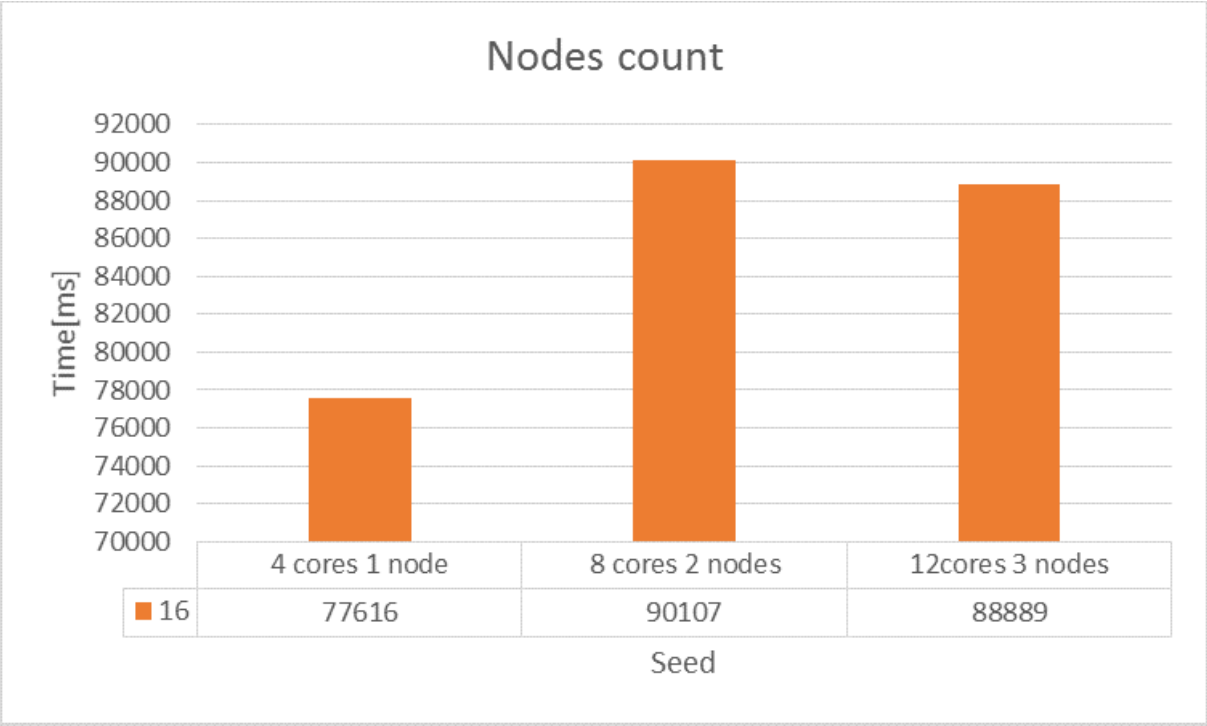
Otrzymane wyniki są dość niespodziewane. Wydawać by się mogło, że dodatkowe węzły liczące powinny powodować redukcję czasu potrzebnego na obliczenia. Tymczasem, nasze pomiary nie wykazują tej własności. Najprawdopodobniej użyty do łączenia maszyn interfejs FastEthernet ma zbyt małą przepustowość przez co procesory nie są maksymalnie wykorzystane i występuje długi czas oczekiwania na komunikację pomiędzy masterem i slavem.



Rysunek 19: Klatka podzielona na 1 podzadanie

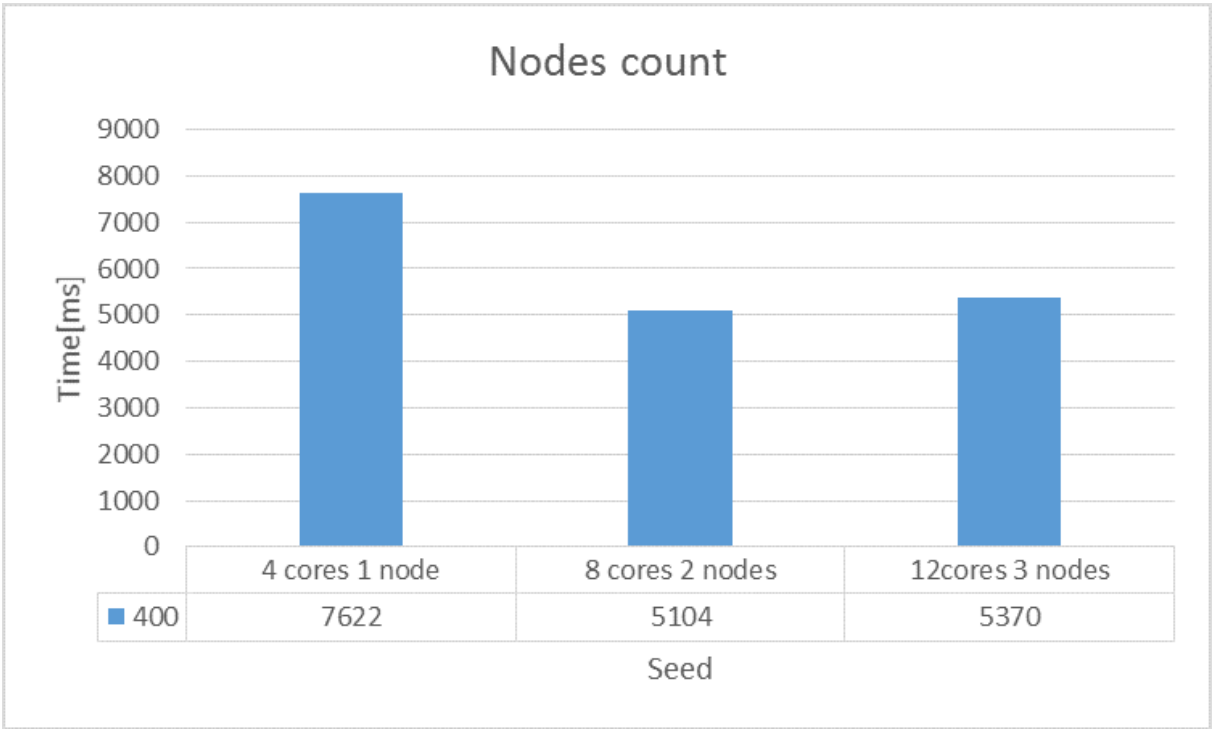


Rysunek 20: Klatka podzielona na 4 podzadania

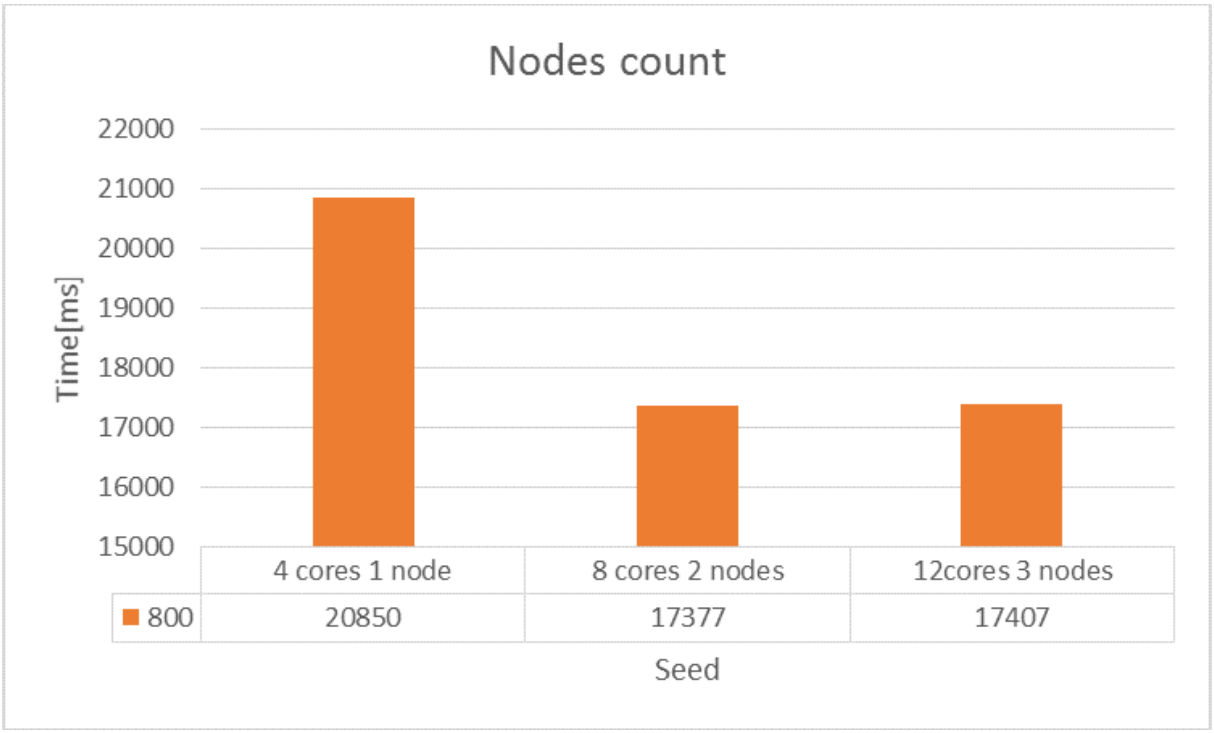


Rysunek 21: Klatka podzielona na 16 podzadań

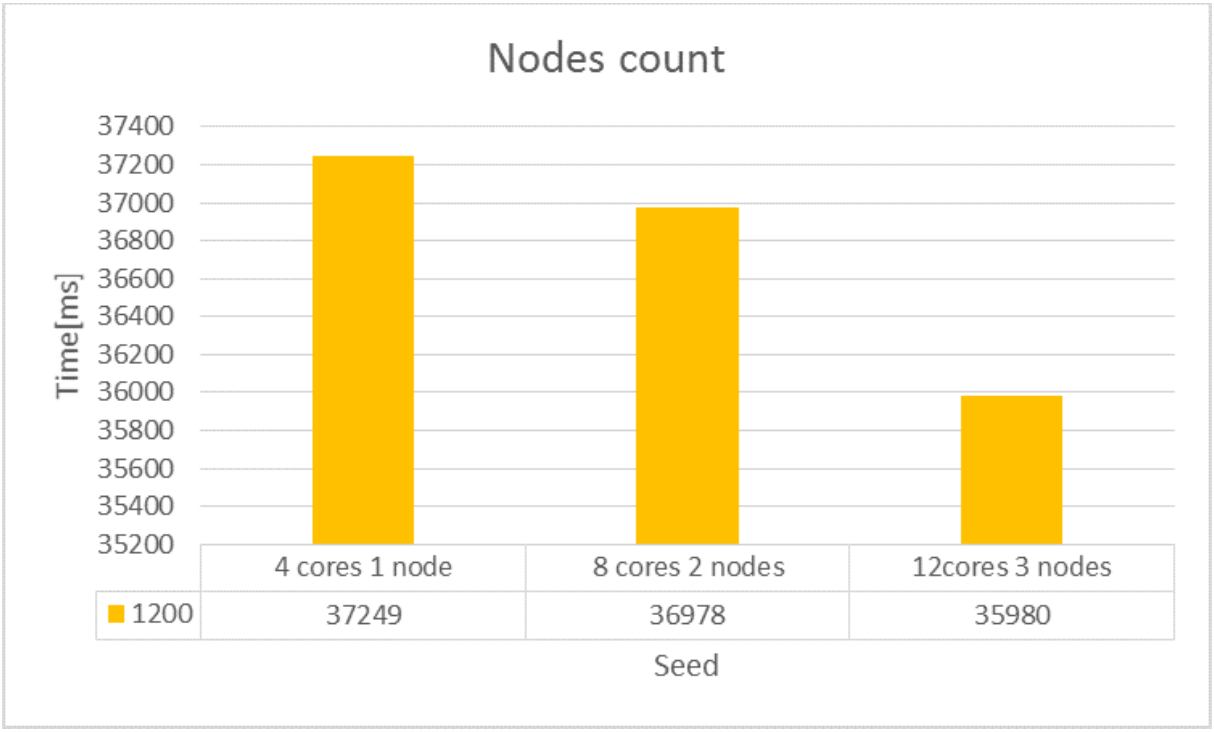
Na wszystkich wykresach można zaobserwować wzrost czasu trwania wykonania zadania zależnie od ilości połączonych węzłów. Jest to spowodowane narzutem komunikacji pomiędzy węzłem master a wszystkimi węzłami slave. Patrząc jednak na wielkość ziarna zadania, czyli podziału jednej klatki na kolejno 1, 4 i 16 podzadań można zauważyć, że większe rozdrobnienie idzie w parze z przyspieszeniem czasu wykonania całości zadania, jest to lepiej widoczne im więcej węzłów połączymy ze sobą. Większa ziarnistość powoduje zmniejszenie zużycia łącza na przesłanie dużego fragmentu danych (przy ziarnie 1 przesyłana jest cała klatka, a przy 16 tylko 1/16). Przesyłanie trwa na tyle krótko żeby nie doszło do zbyt długiego blokowania węzła master podczas gdy inne węzły slave czekają na swoją kolej obsługi.



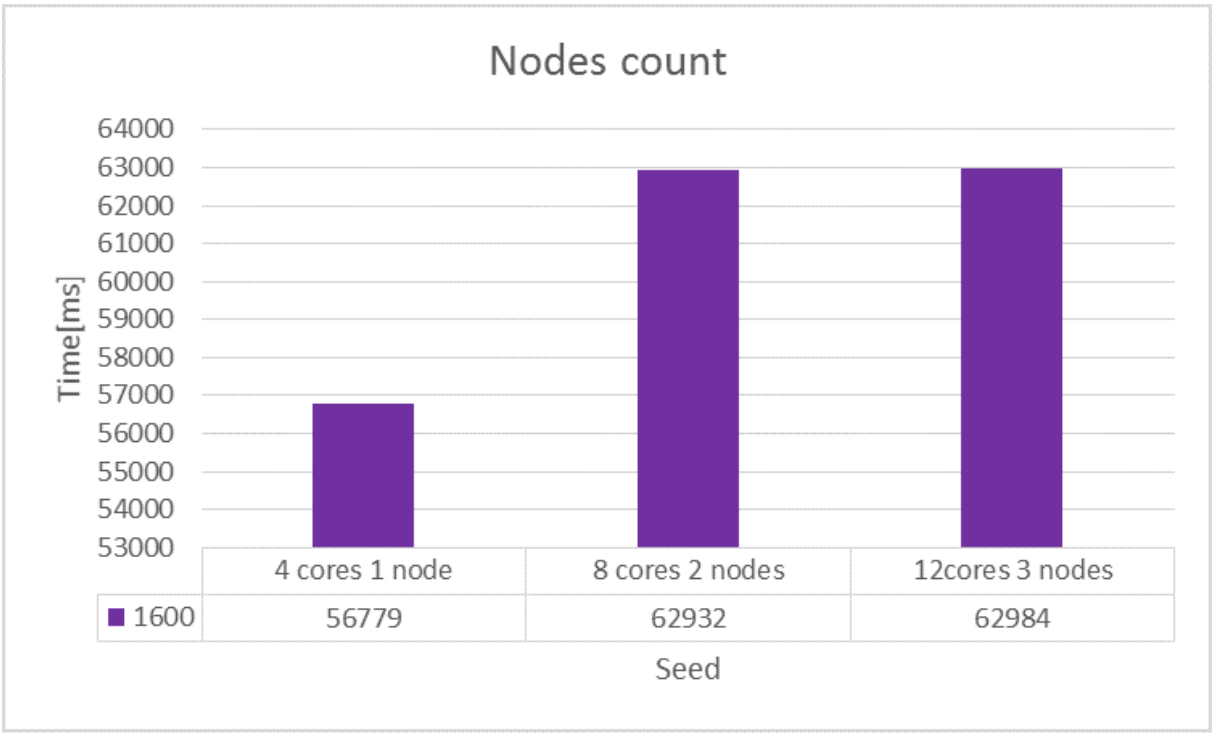
Rysunek 22: Klatka 400x400



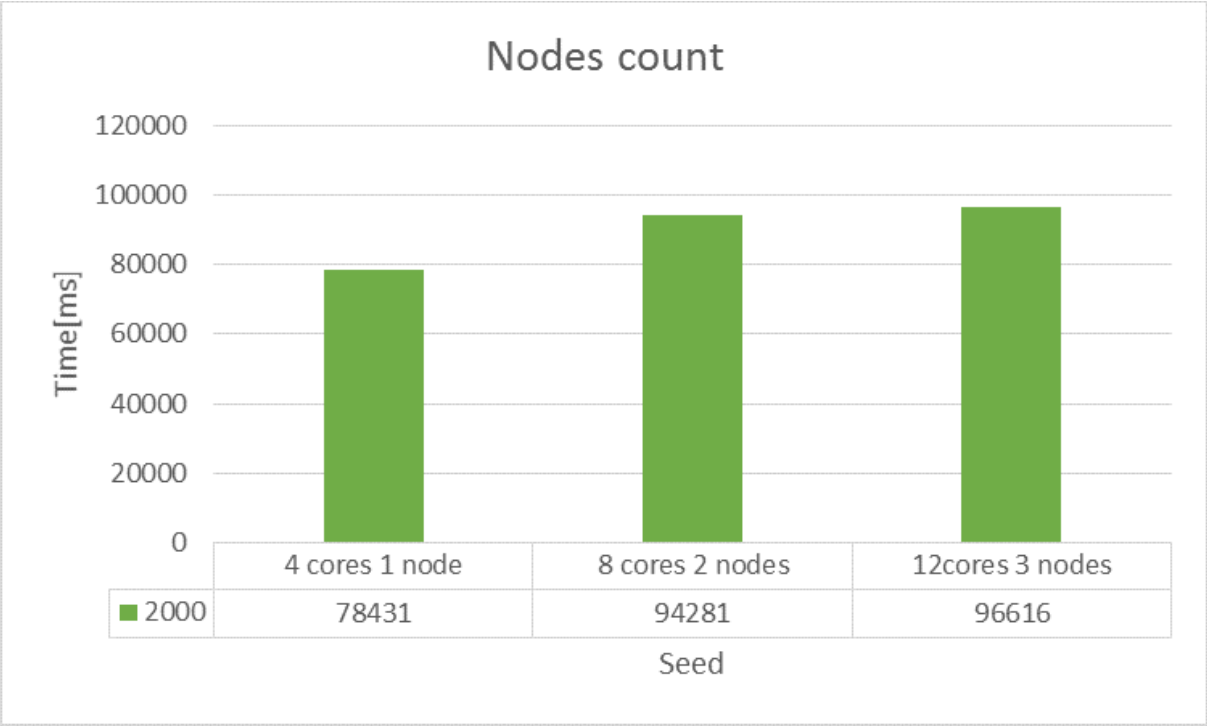
Rysunek 23: Klatka 800x800



Rysunek 24: Klatka 1200x1200



Rysunek 25: Klatka 1600x1600



Rysunek 26: Klatka 2000x2000

Na wykresach czasu trwania wykonania zadania od wielkości klatki w pikselach możemy zauważyć dwie tendencje dla czasu wykonania. Pierwsza z nich to spadkowa, jeśli zwiększamy ilość węzłów i jest ona zauważalna dla rozmiarów klatkek 400, 800, 1200. Druga z nich to tendencja wzrostowa jeśli zwiększamy ilość węzłów i widać ją dla rozmiaru klatek 1600 i 2000. Tendencja spadkowa przy małych klatkach jest spowodowana zwiększeniem mocy obliczeniowej klastra. Natomiast tendencja wzrostowa wynika ze zbyt długiego czasu przesyłania wyniku do węzła master. W tym czasie inne węzły mogą ukończyć zadanie i niestety muszą czekać dłużej na swoją kolej obsługi. Drugim, dość oczywistym wnioskiem, jest to że im większy rozmiar klatki tym więcej czasu potrzeba na wykonanie zadania.

6 Podsumowanie

Projekt ten ukazał zalety wynikające z rozpraszania obliczeń na wielu maszynach. Ponadto pokazał, jak używać technologii webowych (serwera Django, języka Python).