Politechnika Krakowska im. Tadeusza Kościuszki

Wydział Fizyki, Matematyki i Informatyki

Programowanie równoległe i rozproszone

Równoległe działanie silnika do gry w szachy za pomocą MPI w jezyku C#

Data oddania:

17.06.2018

Kierunek: Informatyka Stosowana

Rok akademicki: 2017/2018

Wykonali:

Zdobysław Antas,
Michał Kisielewski

Nr albumu: 124602, 108673

Spis treści

1. Cel i opis projektu

Celem projektu jest zrównoleglenie silnika do gry w szachy za pomocą standardu MPI w języku C#. Użyty w projekcie silnik do gry w szachy to Chess Core autorstwa Adam'a Berent'a zaimplementowany oryginalnie w technologii .NET Core. Sam silnik nie miał zaimplementowanego w żaden sposób zrównoleglenia wyszukiwania najlepszego ruchu, więc sam silnik musiał również być wyedytowany. W projekcie użyta została implementacja standardu MPI o nazwie MPI.NET.

2. Zastosowane technologie

Projekt został wykonany w środowisku Microsoft Visual Studio 2017 w języku C#. Sam silnik został napisany przy użyciu frameworka .NET Core, który pozwala m.in. na możliwość portowania aplikacji pomiędzy różnymi systemami operacyjnimi. Niestety implementacja MPI, z której korzystaliśmy w projekcie, nie wspierała wystarczająco wysokiej wersji .NET Standard, przez co byliśmy zmuszeni do zmiany frameworku na starszy - .NET Framework 4.6.2.

MPI.NET to implementacja standardu MPI w technologii .NET. Wywodzi się z Indiana University w USA. Implementacja ta pozwala za opracowywanie i uruchamianie programów ze standardem MPI w technologii .NET w C#. Obecnie projekt nie jest już utrzymywany. MPI.NET można zainstalować poprzez np. menadżer pakietów NuGet.

2. Opis algorytmu

Program po uruchomieniu inicjalizuje środowisko MPI. Główny proces ma zadanie wyświetlenie interfejsu i obsługę żądań od użytkownika, a pozostałe procesy czekają na otrzymanie danych do przeprocesowania od procesu głównego. Po odebraniu odpowiedniej komendy, czyli definicji ruchu do wykonania, zostaje wywołana część odpowiedzialna za wyszukanie najlepszego ruchu dla przeciwnika (komputera). W tej właśnie części zostało zaimplementowane zrównoleglenie obliczeń.

Sam algorytm wyszukiwania opiera się na algorytmie min-max. Jest to metoda zminimalizowania możliwych strat, co w szachach sprowadza się do analizowania najlepszej

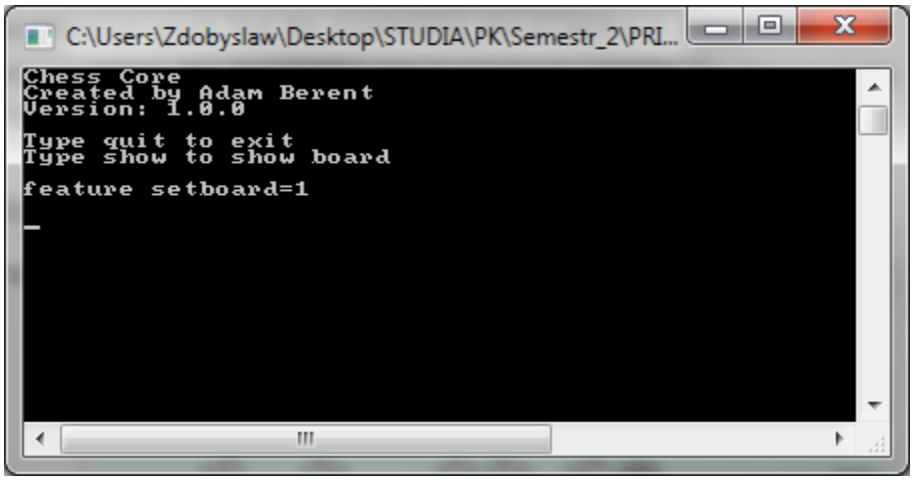
możliwej odpowiedzi ze strony przeciwnika na nasz ruch. Algorytm min-max tworzy drzewo, które przez swoją wielkość jest trudne do przeszukania. Dlatego do algorytmu został zastosowany algorytm Alfa-Beta, który pozwala na jego optymalizację poprzez redukcję liczby węzłów, które muszą być przeszukane/procesowane. Warunkiem stopu w przeszukiwaniu jest znalezienie przynajmniej jednego rozwiązania czyniącego obecnie badaną opcję ruchu gorszą od poprzednio zbadanych opcji.

Do algorytmu odpowiedzialnego za obliczenie są przesyłane między innymi wszystkie możliwe, dla danej pozycji na szachownicy, ruchy do wykonania. Algorytm ma za zadanie dla każdego ruchu obliczenie wartości szachownicy wynikowej po wykonaniu podanej, również jako parametr, liczby ruchów od podanej pozycji.

Taki podział obliczeń pozwala na łatwe jego zrównoleglenie, ponieważ pozycje można podzielić na paczki i przesyłać je do osobnych procesów. Proces główny odpowiednio dzieli dane na liczbę procesów (sam również zajmuje się jedną porcją danych) i wysyła je do reszty procesów. Każdy proces osobno procesuje swoją porcję danych i przesyła z powrotem do procesu głównego najlepszy ruch, który znalazł w swojej porcji danych. Po otrzymaniu wszystkich wyników proces główny wybiera na podstawie wartości szachownicy najlepszy z otrzymanych ruchów. Wybranych ruch zostaje wykonany na szachownicy, po czym kontrola zostaję zwrócona użytkownikowi – silnik oczekuję na następny ruch.

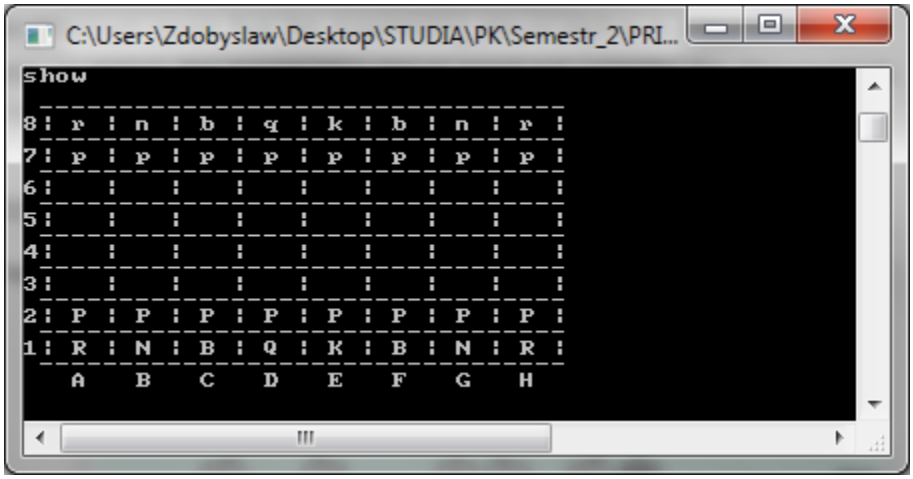
3. Interfejs użytkownika

Interfejs dostępny użytkownikowi jest w pełni tekstowy. Poniżej zaprezentowane jest menu startowe silnika do gry Chess Core.



Rys 1. Menu startowe gry w szachy.

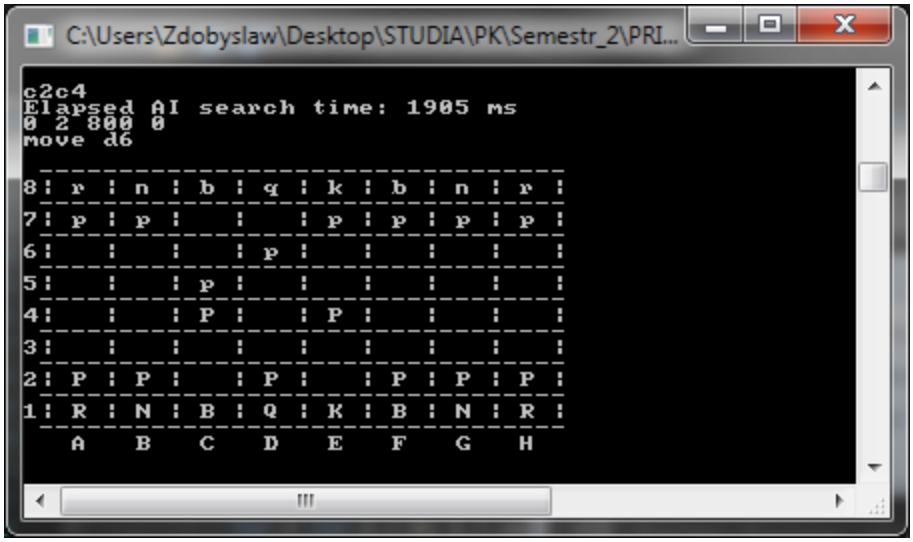
Użytkownik po starcie programu ma do dyspozycji 2 komendy – *show* oraz *quit*. Komenda *quit* pozwala na wyjście z programu. Użycie komendy *show* powoduje wyświetlenie planszy. Poniżej znajduje się zrzut ekranu prezentujący ekran startowy programu.



Rys 2. Plansza startowa gry w szachy.

Na przedstawionej na Rys 2 planszy Wielkimi literami oznaczone są bierki gracza, a małymi literami oznaczone są bierki przeciwnika (algorytmu). W celu wykonania ruchu użytkownik musi podać w konsoli ruch w formacie [a-hA-H][1-8][a-hA-H][1-8], gdzie pierwsza litera oraz cyfra odpowiada współrzędnej bierki, którą wykonuje się ruch, a druga cyfra oraz litera odpowiada polu, na które ma się przesunąć wybrana bierka. Obecne oznaczenia bierek są nastepujące: k – król, q – hetman, b –goniec, n – skoczek, r – wieża oraz p - pion. Jeśli przy

określaniu ruchu został użyty zrównoleglony algorytm dodatkowo wyświetli się komunikat w postaci *Elapsed AI serach time: <czas szukania ruchu> ms*, który informuje jak długo zajęło algorytmowi wyszukanie rozwiązania. Poniżej została przedstawiona plansza gry po wykonaniu ruchu przez użytkownika oraz algorytm.



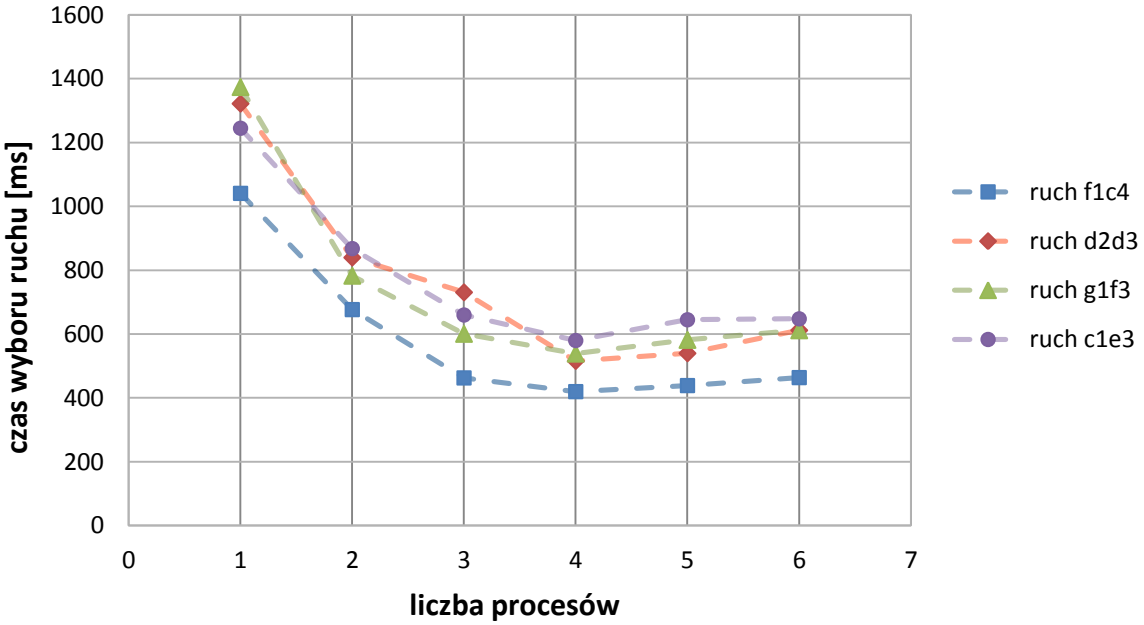
Rys 3. Plansza gry po wykonaniu ruchu przez użytkownika oraz algorytm.

4. Wyniki

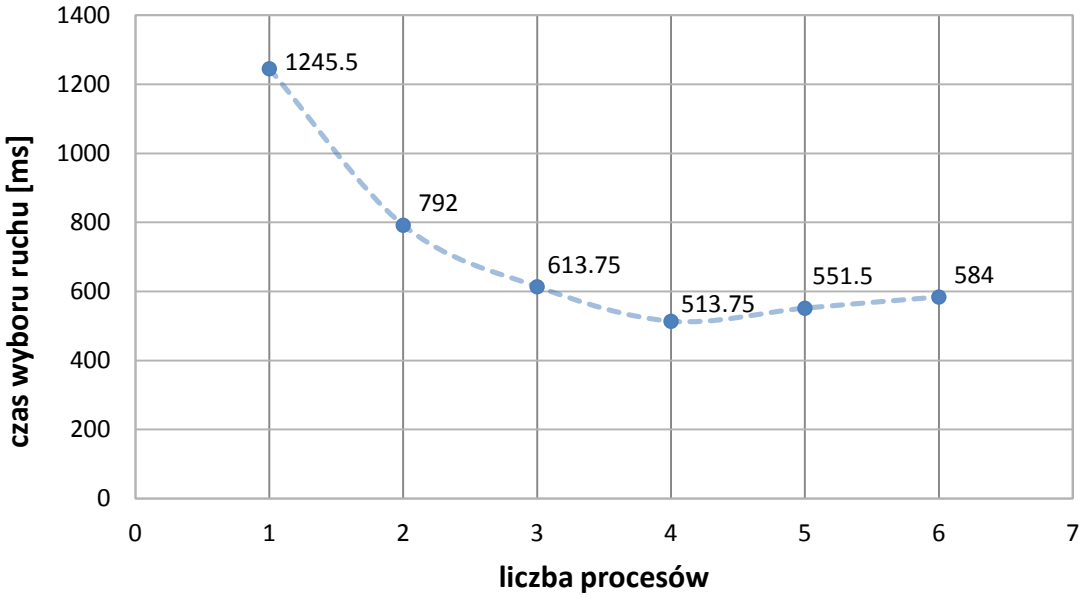
Wyniki ilustrujące przyspieszenie działania zrównoleglonego algorytmu zostały zebrane uruchamiając program na komputerze z 4 rdzeniami fizycznymi. Poniżej zostały przedstawione wyniki dotyczące czasu oraz przyspieszenia działania algorytmu wyszukującego optymalny ruch przeciwnika w zależności od ilości procesów.

	czas [ms]				
liczba procesów	**ruch f1c4**	**ruch d2d3**	**ruch g1f3**	**ruch c1e3**	**średnia**
1	1041	1322	1374	1245	1245.5
2	677	840	783	868	792
3	463	731	601	660	613.75
4	420	516	539	580	513.75
5	439	540	582	645	551.5
6	464	612	612	648	584

Rys 4. Tabela przedstawiająca czas szukania ruchu przeciwnika w zależności od ilości procesów dla kolejnych ruchów gracza.



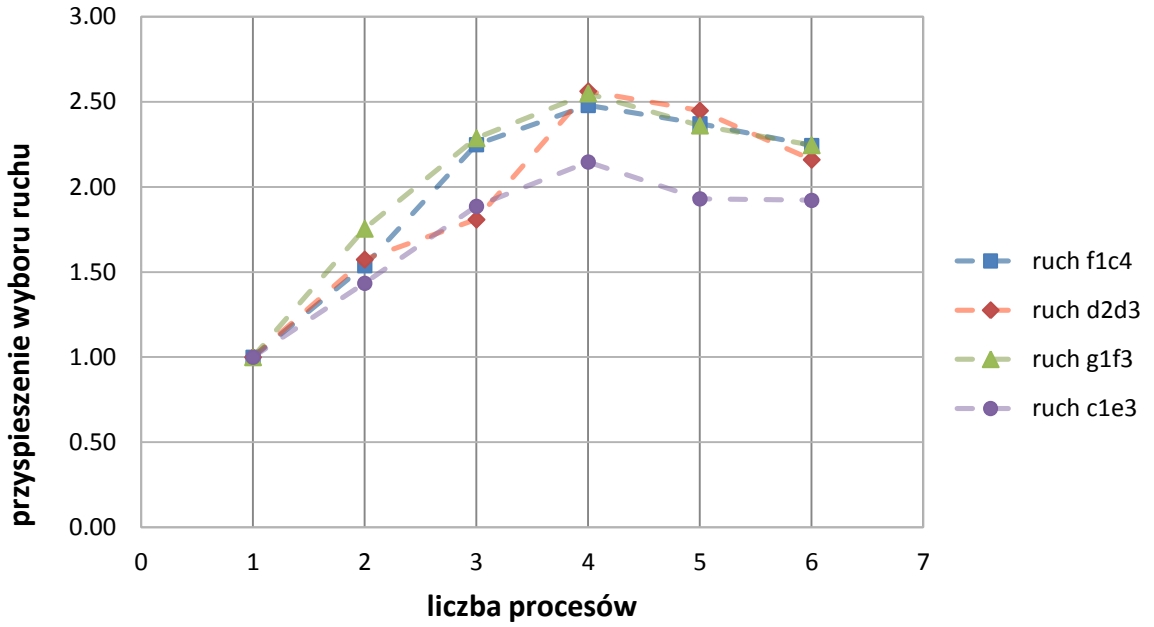
Rys 5. Wykres przedstawiający czas szukania ruchu przeciwnika w zależności od ilości procesów dla kolejnych ruchów gracza.



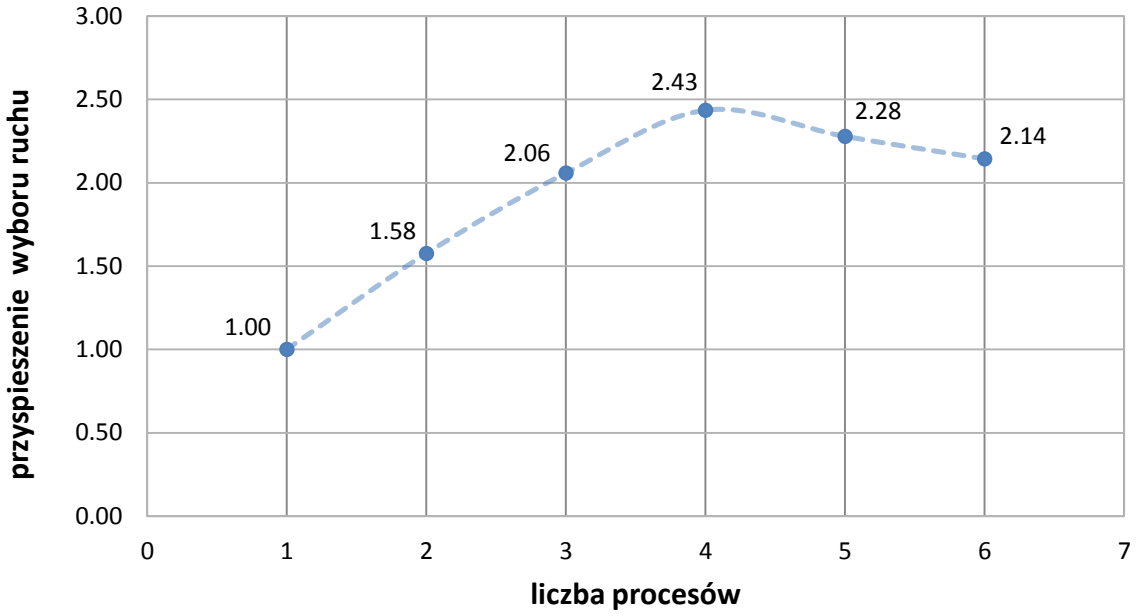
Rys 6. Wykres przedstawiający średni czas szukania ruchu przeciwnika w zależności od ilości procesów.

	przyspieszenie [ms]				
liczba procesów	**ruch f1c4**	**ruch d2d3**	**ruch g1f3**	**ruch c1e3**	**średnia**
1	1.00	1.00	1.00	1.00	1.00
2	1.54	1.57	1.75	1.43	1.58
3	2.25	1.81	2.29	1.89	2.06
4	2.48	2.56	2.55	2.15	2.43
5	2.37	2.45	2.36	1.93	2.28
6	2.24	2.16	2.25	1.92	2.14

Rys 7. Tabela przedstawiająca przyspieszenie szukania ruchu przeciwnika w zależności od ilości procesów dla kolejnych ruchów gracza.



Rys 8. Wykres przedstawiający przyspieszenie obliczeń ruchu przeciwnika w zależności od ilości procesów dla kolejnych ruchów gracza.



Rys 9. Wykres przedstawiający średnie przyspieszenie szukania ruchu przeciwnika w zależności od ilości procesów.

Przedstawione wyniki obrazują skuteczność równoległego algorytmu. Widać na wykresach sukcesywne zwiększanie się przyspieszenia do 4 procesów, jednak to

przyspieszenie trochę odbiega od optymalnego przypadku. Należy wspomnieć, że na wyniki mogła wpłynąć duża liczba innych procesów, które były uruchomione w czasie działania programu.

5. Podsumowanie i wnioski

Udało się w projekcie zrealizować zrównoleglenie algorytmu wyszukującego optymalny ruch przeciwnika w grze w szachy. Jednym z największych problemów pojawiających się w projekcie była sama komunikacja międzyprocesowa. Algorytm min-max obecny w tym projekcie nie należy do najłatwiejszych do optymalnego zrównoleglenia, ponieważ jest algorytmem rekurencyjny. Jeśli chodzi o samą implementację MPI w C# to MPI.NET jest rozwiązaniem znacznie upraszczającym pewne elementy standardu MPI, przez co używanie tej wersji standardu jest trochę łatwiejsze niż wersji na C/C++.