Spis treści

[1. Wstęp 5](#_Toc429925461)

[1.1. Cel pracy dyplomowej 5](#_Toc429925462)

[1.2. Krótka historia szachów 5](#_Toc429925463)

[1.3. Historia programów szachowych 5](#_Toc429925464)

[1.4. Wygląd szachownicy 5](#_Toc429925465)

[2. Podobne rozwiązania 5](#_Toc429925466)

[2.1. Fritz 5](#_Toc429925467)

[2.1.1. Historia 5](#_Toc429925468)

[2.1.2. Najciekawsze funkcjonalności 7](#_Toc429925469)

[2.2. Brutus/Hydra 8](#_Toc429925470)

[2.2.1. Historia 8](#_Toc429925471)

[2.2.2. Rozwiązania techniczne 9](#_Toc429925472)

[2.3. Deep Blue 10](#_Toc429925473)

[2.3.1. Historia 10](#_Toc429925474)

[2.3.2. Rozwiązania techniczne 12](#_Toc429925475)

[2.4. Internetowy rozproszony algorytm szachowy z bazą wiedzy i uczeniem. 13](#_Toc429925476)

[2.4.1. Założenia projektu 13](#_Toc429925477)

[2.4.2. Baza danych 14](#_Toc429925478)

[2.4.3. Klasteryzacja 14](#_Toc429925479)

[2.4.4. Algorytm Genetyczny 15](#_Toc429925480)

[3. Gra energooszczędna 17](#_Toc429925481)

[3.1. Założenia 17](#_Toc429925482)

[3.2. Przegląd problematyki 17](#_Toc429925483)

[3.2.1. Miernik prądu, watomierz 17](#_Toc429925484)

[3.2.2. PowerStat, PowerTOP 18](#_Toc429925485)

[3.2.3. Obliczanie zużycia na podstawie obciążenia 19](#_Toc429925486)

[3.2.4. Performance Counters for Linux (PCL) 19](#_Toc429925487)

[3.2.5. Model Specific Registers (MSR) 20](#_Toc429925488)

[3.2.6. nVidia Management Library (NVML) 20](#_Toc429925489)

[3.2.7. CUDA 20](#_Toc429925490)

[3.2.8. Performance API (PAPI) 20](#_Toc429925491)

[3.3. Podsumowanie 22](#_Toc429925492)

[3.4. Instalacja PAPI 22](#_Toc429925493)

[3.5. PapiManager 22](#_Toc429925494)

[3.5.1. Opis 22](#_Toc429925495)

[3.5.2. Diagram klas 22](#_Toc429925496)

[3.5.3. Klasa CPapiManager 24](#_Toc429925497)

[3.5.3.1. Opis 24](#_Toc429925498)

[3.5.3.2. Pola 24](#_Toc429925499)

[3.5.3.3. Funkcje i metody 24](#_Toc429925500)

[3.5.4. Klasa CPapiMeasure 28](#_Toc429925501)

[3.5.4.1. Opis 28](#_Toc429925502)

[3.5.4.2. Pola 28](#_Toc429925503)

[3.5.4.3. Funkcje i metody 28](#_Toc429925504)

[3.5.5. Klasa CComponentInfo 29](#_Toc429925505)

[3.5.5.1. Opis 29](#_Toc429925506)

[3.5.5.2. Pola 29](#_Toc429925507)

[3.5.5.3. Funkcje i metody 29](#_Toc429925508)

[3.5.6. Klasa CEventInfo 30](#_Toc429925509)

[3.5.6.1. Opis 30](#_Toc429925510)

[3.5.6.2. Pola 30](#_Toc429925511)

[3.5.6.3. Typy wyliczeniowe 31](#_Toc429925512)

[3.5.6.4. Funkcje i metody 31](#_Toc429925513)

[3.5.7. Klasa CEventResult 33](#_Toc429925514)

[3.5.7.1. Opis 33](#_Toc429925515)

[3.5.7.2. Pola 33](#_Toc429925516)

[3.5.7.3. Funkcje i metody 34](#_Toc429925517)

[3.5.8. Klasa CLogger 35](#_Toc429925518)

[3.5.8.1. Opis 35](#_Toc429925519)

[3.5.8.2. Typy wyliczeniowe 35](#_Toc429925520)

[3.5.8.3. Funkcje i metody 35](#_Toc429925521)

[3.5.9. Przykładowy kod w C++ korzystający z biblioteki PapiManager 35](#_Toc429925522)

[3.6. Java Native Access 37](#_Toc429925523)

[3.7. JPapiManager - C **NOTE: Możnaby zmienić nazwę – jest myląca (J oznacza, ze jest w JAVA, a to C)** 37](#_Toc429925524)

[3.7.1. Opis 37](#_Toc429925525)

[3.7.2. Diagram 38](#_Toc429925526)

[3.7.3. Struktura EventInfoStruct 39](#_Toc429925527)

[3.7.3.1. Opis 39](#_Toc429925528)

[3.7.3.2. Pola 39](#_Toc429925529)

[3.7.4. Struktura EventInfoArrayStruct 39](#_Toc429925530)

[3.7.4.1. Opis 39](#_Toc429925531)

[3.7.4.2. Pola 39](#_Toc429925532)

[3.7.5. Struktura EventResultStruct 40](#_Toc429925533)

[3.7.5.1. Opis 40](#_Toc429925534)

[3.7.5.2. Pola 40](#_Toc429925535)

[3.7.6. Struktura EventResultArrayStruct 40](#_Toc429925536)

[3.7.6.1. Opis 40](#_Toc429925537)

[3.7.6.2. Pola 40](#_Toc429925538)

[3.7.7. Struktura ComponentInfoStruct 40](#_Toc429925539)

[3.7.7.1. Opis 40](#_Toc429925540)

[3.7.7.2. Pola 40](#_Toc429925541)

[3.7.8. Struktura ComponentInfoArrayStruct 41](#_Toc429925542)

[3.7.8.1. Opis 41](#_Toc429925543)

[3.7.8.2. Pola 41](#_Toc429925544)

[3.7.9. Funkcje i metody 41](#_Toc429925545)

[3.8. JPapiManager – Java 43](#_Toc429925546)

[3.8.1. Opis 43](#_Toc429925547)

[3.8.2. Diagram klas 43](#_Toc429925548)

[3.8.3. Interface PapiManagerLibrary 43](#_Toc429925549)

[3.8.3.1. Opis 43](#_Toc429925550)

[3.8.3.2. Pola 43](#_Toc429925551)

[3.8.3.3. Zagnieżdżone klasy 43](#_Toc429925552)

[3.8.3.4. Funkcje i metody 44](#_Toc429925553)

[3.8.4. Klasa JPapiManager 45](#_Toc429925554)

[3.8.4.1. Opis 45](#_Toc429925555)

[3.8.4.2. Interface’u 45](#_Toc429925556)

[3.8.4.3. Funkcje i metody 45](#_Toc429925557)

[3.8.5. Klasa PapiEventInfo 46](#_Toc429925558)

[3.8.5.1. Opis 46](#_Toc429925559)

[3.8.5.2. Pola 46](#_Toc429925560)

[3.8.5.3. Typy wyliczeniowe 47](#_Toc429925561)

[3.8.5.4. Funkcje i metody 47](#_Toc429925562)

[3.8.6. Klasa PapiEventResult 48](#_Toc429925563)

[3.8.6.1. Opis 48](#_Toc429925564)

[3.8.6.2. Pola 48](#_Toc429925565)

[3.8.6.3. Funkcje i metody 48](#_Toc429925566)

[3.8.7. Klasa PapiComponentInfo 49](#_Toc429925567)

[3.8.7.1. Opis 49](#_Toc429925568)

[3.8.7.2. Pola 49](#_Toc429925569)

[3.8.7.3. Funkcje i metody 50](#_Toc429925570)

[3.8.8. Przykładowy kod korzystający z JPapiManager – Java 50](#_Toc429925571)

[4. Użyte protokoły i interface’y 51](#_Toc429925572)

[4.1. Universal Chess Interface (UCI) 51](#_Toc429925573)

[4.1.1. Krótki opis do czego służy protokół 51](#_Toc429925574)

[4.1.2. Jak to działa? 51](#_Toc429925575)

[4.2. Forsyth-Edwards Notation (FEN) 52](#_Toc429925576)

[4.2.1. Opis 52](#_Toc429925577)

[4.2.2. Notacja 52](#_Toc429925578)

[4.2.3. Przykład 54](#_Toc429925579)

[4.3. Standard Algebraic Notation (SAN) 56](#_Toc429925580)

[4.3.1. Opis 56](#_Toc429925581)

[4.3.2. Notacja 56](#_Toc429925582)

[4.3.3. Przykłady 57](#_Toc429925583)

[4.4. Portable Game Notation (PGN) 59](#_Toc429925584)

[4.4.1. Opis 59](#_Toc429925585)

[4.4.2. Notacja 59](#_Toc429925586)

[4.4.3. Przykład 60](#_Toc429925587)

[5. Aplikacja 60](#_Toc429925588)

[5.1. Opis aplikacji 60](#_Toc429925589)

[5.2. Nasza baza danych 60](#_Toc429925590)

[5.2.1. Struktura bazy danych 60](#_Toc429925591)

[5.2.2. Przykładowy wpis w bazie 60](#_Toc429925592)

[5.2.3. Przykładowe zapytanie używane w naszej aplikacji 60](#_Toc429925593)

[5.3. Klasteryzacja 60](#_Toc429925594)

[5.4. Uczenie algorytmem genetycznym 60](#_Toc429925595)

[5.5. Architektura 60](#_Toc429925596)

[5.6. Instrukcja użytkownika 60](#_Toc429925597)

[5.6.1. Konfiguracja systemu 60](#_Toc429925598)

[5.6.1.1. Instalacja PAPI 60](#_Toc429925599)

[5.6.1.2. Instalacja biblioteki PapiManager 62](#_Toc429925600)

[5.6.2. Interface uzytkownika 63](#_Toc429925601)

[5.6.3. Gra z uzytkownikiem 63](#_Toc429925602)

[5.6.4. Gra aplikacji samej ze soba 63](#_Toc429925603)

[5.6.5. Gra energooszczędna 63](#_Toc429925604)

# Wstęp

## Cel pracy dyplomowej

## Krótka historia szachów

## Historia programów szachowych

## Wygląd szachownicy

# Podobne rozwiązania

## Fritz

### Historia

Fritz to nazwa serii gier szachowych stworzonych przez studio ChessBase pochodzące z Hamburga. Korzenie serii sięgają 1991 roku, kiedy to ukazała się pierwsza wersja gry na system operacyjny MS DOS. Od tego czasu na rynku pojawiło się łącznie 27 różnych wersji gry. Najnowsza z nich, Deep Fritz 14, miała swoją premierę w listopadzie 2013 roku. Aplikacja wyposażona jest w autorki silnik firmy ChessBase.

Dla Fritz 1.0 silnik został stworzony przez holendra Fransa Morscha oraz Mathiasa Feista – programistę i byłego gracza szachowego niemieckiego pochodzenia. Podstawą silnika w pierwszej odsłonie Fritz był silnik Quest stworzony przez Morsh’a jeszcze przed rozpoczęciem pracy w ChessBase. Rozwiązanie to było zaimplementowane w assemblerze 6502 z dedykacją dla 8-bitowych komputerów szachowych Sphinx, doczekało się także protu na procesory z serii H8. Quest zwyciężył między innymi w turnieju Dutch Computer Chess Championship (DOCCC) w 1988 roku.



Obrazek 1. Oryginalny Fritz 1.0 na MS-DOS.

Źródło: <https://chessprogramming.wikispaces.com/Fritz>

Wraz z przejściem Fransa do ChessBase oraz rozwoju silnika wraz z pomocą Mathiasa silnik odnosił kolejne sukcesy i był swojego czasu uważany za jeden z najsilniejszych silników szachowych. Wraz z rozwojem kolejnych jego wersji wygrał kolejne edycje DOCCC w 1994, 1999 oraz 2001 roku.

Największym sukcesem odniesionym przez Fritz było uzyskanie tytułu WCCC (World Computer Chess Champion) w 1995 roku. W półfinale turnieju pokonał on prototyp komputera Deep Blue firmy IBM – na rok przed jego słynnym meczem z Garrym Kasparovem. Podczas turnieju wykorzystywano silnik Quest w wersji 3.0.

|  |
| --- |
| [Event "WCCC 1995"]  [Site "Chinese University HKG"]  [Date "1995.05.29"]  [Round "5"]  [White "Deep Blue Prototype"]  [Black "Fritz"]  [Result "0-1"]    1.e4 c5 2.Nf3 Nc6 3.d4 cxd4 4.Nxd4 Nf6 5.Nc3 e5 6.Ndb5 d6 7.Bg5 a6 8.Na3 b5 -9.Bxf6 gxf6 10.Nd5 f5 11.Bd3 Be6 12.Qh5 f4 13.O-O Rg8 14.Kh1 Rg6 5.Qd1 Rc8 16.c4 Qh4 17.g3 Qh3 18.Qd2 f3 19.Rg1 Rh6 20.Qxh6 Qxh6 21.cxb5 Bxd5 22.exd5 Nb4 23.Bf5 Rc5 24.bxa6 Nxa6 25.Nc2 Qd2 26.Ne1 Rxd5 27.Nxf3 Qxf2 28.Be4 Ra5 29.Rg2 Qe3 30.Re1 Qh6 31.Bc6+ Kd8 32.a3 f5 33.Rc2 Rc5 34.Rxc5 Nxc5 35.Rf1 Be7 36.a4 f4 37.gxf4 Qxf4 38.Rg1 Nxa4 39.b4 Qxb4 0-1 |

Tabela 1. Przebieg meczu Deep Blue - Fritz. Półfinał WCCC1995.

Źródło: <https://chessprogramming.wikispaces.com/WCCC+1995>

Silnik Quest opierał się na chętnie badanej przez holenderskich informatyków acz mało popularnej jeszcze na świecie heurystyce pustych posunięć. Heurystyka ta polega na minimalizacji drzewa gry poprzed wykonanie sztuczengo, pustego ruchu i przeanalizowaniu możliwości naszego zwycięstwa po jego wykonaniu jak i możliwości obrony przeciwnika – pozwala to na odcinanie kolejncyh gałęzi drzewa i tym samym jego redukcję. Następnie na podstawie zredukowanego drzewa podejmowana jest decyzja o finalnie już wykonywanym ruchu.

Wraz z wydaniem Fritz13 Morsch opuścił zespół ChessBase i zaczął zajmować się kolejnymy projektami. Jego miejsce zajął Gyula Horvath, elektronik i programista szachowy. Puki co, pod jego pieczą została wydana na razie tylko jedna wersja gry – Fritz14.



Obrazek 2. Fritz14.

Źródło: <https://chessprogramming.wikispaces.com/Fritz>

### Najciekawsze funkcjonalności

Najciekawszą funkcionalnością dostępna w grach serii Fritz jest dostępna od wersji 13 możliwość analizy gry nie tylko przez silnik dostępny w grze ale również dowolny inny zgodny z Universal Chess Interface (UCI).

Aby skorzystać z tej funkcjonalności wystarczy wybrać grę zapisaną w formacie PGN i wybrać kilka silników, którymi chcemy przeprowadzić analizę gry. W odpowiedzi program wyświetli nam gałąź wybraną przez każdy z wybranych silników dla każdego z ruchów wykonanych w trakcie rozgrywki. Kolejne silniki odpalane są sekwencyjnie.

Rysunek przedstawiony poniżej pokazuje wynik analizy gry przez 3 silniki: Fritz13 (zielony), Shredder 12 (czerwony) oraz Junior 12 (fioletowy).



Obrazek 3. Fritz14 - funkcja Compare Analysis.

Źródło: <https://www.youtube.com/watch?v=4aCmkE_SAm8>

Aplikacje z serii Fritz są aplikacjami płatnymi. Pojedyncza kopia najnowszej wersji Deep Fritz 14 to koszt około 70 euro.

## Brutus/Hydra

### Historia

Brutus to silnik stworzony przez Chrilliego Donningera z pomocą Alexa Kurea and Ulfa Lorenza. Podstawową cechą różniącą go od innych silników było zastosowanie akceleratora na bazie płyty FPGA. Wzorem dla twórców był komputer szachowy Belle autorstwa Kena Thompsona – zwycięska maszyna WCCC w 1980 roku.

Ken na początku 2000 roku podzielił się swoim pomysłem na wparcie obliczeń silnika szachowego przez układ FPGA swojemu przyjacielowi Frederic Friedel – współzałożycielowi studia ChessBase. Frederic zauważył w tym rozwiązaniu korzyści marketingowe oraz technologiczne, stworzył więc w ramach firmy nowy projekt pod przewodnictwem Chrilliego Donningera – doktora matematyki i nauk informatycznych Politechniki Wiedeńskiej.

Prace nad rozwiązaniem rozpoczęły się w listopadzie 2000 roku, a już po ok. 2 latach Brutus rozegrał swój pierwszy mecz. Po tym wydażeniu do projektu zaangażowado dwie nowe osoby – Alexa Kura oraz Ulfa Lorenza. Do rozwoju przyczyniła się również współpraca z Uniwersytetem w Paterborn.

Podczas WCCC 2002 oraz IPCCC 2003 Brutus zajął wysokie, 3 miejsce a w turnieju WCCC 2003 był bliski zwycięstwa, jednak dwie porażki z rzędu ostatecznie uplasowały go na 4 pozycji. Podczas ostatniego z tych meczy do obliczeń wykorzystano 4 komputery uzbrojonye w dwa układy FPGA każdy. Komputery połączone były ze sobą w klaster obliczeniowy. Po tym wydażeniu zmieniono założenia projektowe Brutusa i przemianowano go na Hydre.

Hydra miała być rozwiązaniem bazującym na dużym klastrze obliczeniowym z jednostkami obliczeniowymi wyposarzonymi w układy FPGA. Wykorzystanie klastra miało dodatkowo przyśpieszyć obliczenia poprzez zastosowanie zrównoleglonych obliczeń. Projektowi udało się uzyskać inwestorów z Abu Dhabi, co umożliwiło wystartowanie projektu.

Hydra zwyciężyła w IPCCC 2004 i 2005 oraz zajęła drugie miejsce w ICT 2004. Ostatni mecz rozegrała w 2006 roku a w 2009 projekt oficjalnie został zamknięty.

### Rozwiązania techniczne

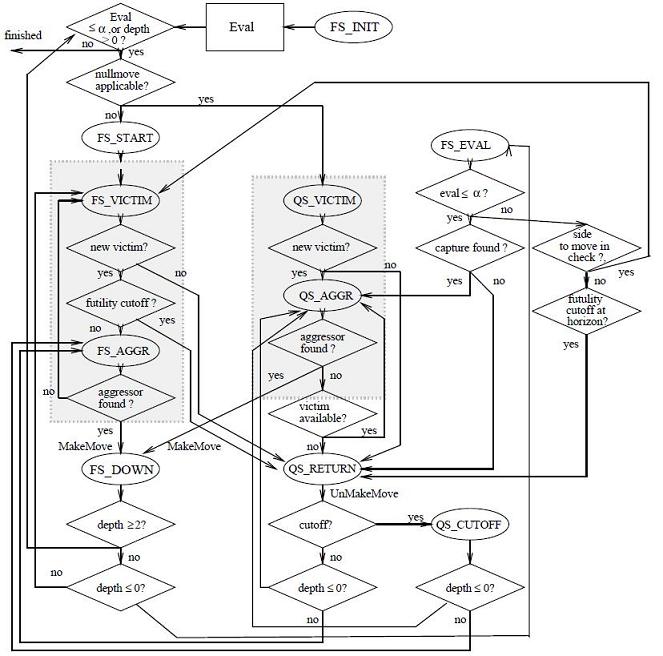
Brutus bazował na zwykłym komputerze klasy PC. Dodatkowo, do płyty głównej podpinana była odpowiednio zaprogramowana karta z modułem FPGA – był to moduł developerski Xilinx Virtex V405E. Na nim z kolei znajdował się program napisany w języku Verilog. Sam silnik był programem napisanym w C oraz asemblerze.



Obrazek 4. Akcelerator FPGA Brutusa na bazie Xilinx Virtex V405E.

Źródło: <https://chessprogramming.wikispaces.com/Brutus>

Program wgrany na akcelerator był stanowym automatem skończonym posiadającą 54 stany. Na układzie obliczane były ostatnie 3 półruchy przeliczane przez silnik – było to spowodowane dużą ilością obliczeń kombinatorycznych do których przeliczania układ i jego programowanie idealnie się nadawały.



Obrazek 5. Uproszczony diagram stanów akceleratora FPGA Brutusa.

Źródło: <https://chessprogramming.wikispaces.com/Brutus>

Podsumowując, Brutus to tak naprawdę pojedynczy silnik, który wykorzystuje obliczenia równoległe i akceleracje sprzętową w celu generowania i przyspieszenia przeglądania drzewa gry oraz eliminowania w nim niewłaściwych gałęzi.

## Deep Blue

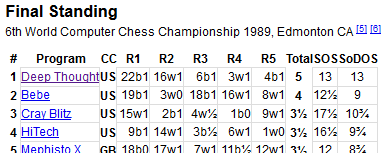
### Historia

W 1985 roku na Uniwersytecie Carnegie Mellon w Pitsburgu (USA) doktorant Feng-hsiung Hsu założył projekt o nazwie ChipTest. Jego bazą był układ scalony w technologii VLSI (*ang. Very-Large-Scale Integration*) stanowiący generator ruchów podpięty do komputera SUN3/160 – maszyny pracującej pod kontrolą systemu UNIX wyposażonej w 16.67 MHz procesor oraz 16MB pamięci RAM.

W roku 1986 Hsu wraz ze swoim zespołem wystawił ChipTest do ACM North American Computer Chess Championship (ACM 1986) działającego pod patronatem Association for Computing Machinery. Zajął na nim dopiero 11 miejsce.

Rok później w turnieru tej samej serii ChipTest zajął 1 miejsce pokonując na swojej drodze wszystkich przeciwników. Od 1986 roku ChipTest przyśpieszył ponad pięciokrotnie osiągając szybkość pół miliona przeliczonych ruchów na sekundę. Jego generator ruchów był tablicą o rozmiarze 8x8 przeliczaną przez kartę VLSI przy użyciu zaprogramowanej na niej logiki kombinatorycznej. Wzorem dla tego rozwiązania był komputer szachowy Belle dominujący na tle innych rozwiązań na przełomie lat 70 i 80.

Na bazie zdobytych doświadczeń zespół postanowił stworzyć zupełnie nową maszynę opartą o idee ChipTest’u. Tej nadano nazwę Deep Thought, nawiązującą do hiperinteligętnego komputera z powieści Autostopem przez Galaktykę. Już pierwsza wersja tego rozwiązania oznaczona numerem 0.01 wygrała turniej ACM 1988, a jej kolejna wersja 0.02 w 1989 zdobyła pierwsze miejsce w rozgrywach WCCC 1989 pokonując po drodze wszystkich napotkanych rywali.

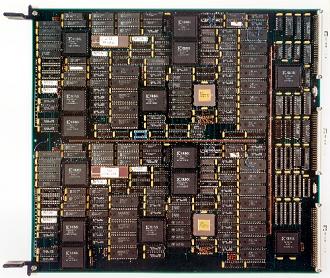


Obrazek 6. Tabela wyników WCCC 1989.

Źródło: <https://chessprogramming.wikispaces.com/WCCC+1989>

Deep Thought był również pierwszym rozwiązaniem, które uzyskało ranking powyżej 2500 nadawanych przez United States Chess Federation (USCF) – ten wynosił dokładnie 2551.

Dobra passa nie opuszczała twórców – ich rozwiązanie zwyciężyło również w ACM 1989 oraz pokonało mistrza szachowego Davida Levy. Deep Thought był pierwszym rozwiązaniem komputerowym, któremu udało się go pokonać (a szachista ten chętnie stawał w szranki z innymi silnikami tego typu).



Obrazek 7. Płyta VLSI Deep Thought.

Źródło: <https://chessprogramming.wikispaces.com/Deep+Thought>

Również w 1989 roku, dzięki sponsorowi ACM, po raz pierwszy przeprowadzono mecz między komputerem a arcymistrzem Garrym Kasparovem. Człowiek jednak bez problemu pokonał maszynę w dwóch rozgrywkach.

Na przełomie lat 80 i 90 IBM, zaintrygowany poczynaniami i sukcesami Deep Thought, rozpoczął we współpracy z Hsu nowy projekt na nim bazujący. Jego założenia były jasne: rozwiązanie miało być najsilniejszym silnikiem szachowym wszechczasów oraz miało pokonać Garrego Kasparova w bezpośredniej potyczce. Projekt zyskał nazwę Deep Blue, które było złożeniem Deep Thought, na którym bazował, oraz Big Blue, jak potocznie nazywano firmę IBM.

Prototyp Deep Blue (nazywany również ze wzgędów PR-owskich Deep Thought II) pierwszy turniej rozegrał na WCCC 1995, gdzie ku niezadowoleniu twórców przegrał mecz półfinałowy z programem Fritz (opisywanym wcześniej w tym rozdziale).

Jednak najważnieszym starciem, jakie miał przed sobą Deep Blue było rozegranie meczu przciw Kasparovowi. Mecz ten odbył się w 1996 roku i pomimo przegranej w całym starciu to po raz pierwszy udało się maszynie wygrać pojedynczą grzę przeciw arcymistrzowi (1 wygrana, 3 przegrane, 2 remisy).

|  |
| --- |
| 1. e4 c5 2. c3 d5 3. ed5 Qd5 4. d4 Nf6 5. Nf3 Bg4 6. Be2 e6 7. O-O Nc6 8. Be3 cd4 9. cd4 Bb4 10. a3 Ba5 11. Nc3 Qd6 12. Ne5 Be2 13. Qe2 Bc3 14. bc3 Ne5 15. Bf4 Nf3 16. Qf3 Qd5 17. Qd3 Rc8 18. Rfc1 Qc4 19. Qc4 Rc4 20. Rcb1 b6 21. Bb8 Ra4 22. Rb4 Ra5 23. Rc4 O-O 24. Bd6 Ra8 25. Rc6 b5 26. Kf1 Ra4 27. Rb1 a6 28. Ke2 h5 29. Kd3 Rd8 30. Be7 Rd7 31. Bf6 gf6 32. Rb3 Kg7 33. Ke3 e5 34. g3 ed4 35. cd4 Re7 36. Kf3 Rd7 37. Rd3 Rad4 38. Rd4 Rd4 39. Ra6 b4 |

Tabela 2. Zapis pierwszej gry wygranej przez Deep Blue z Garrym Kasparovem (rok 1996).

Źródło: <http://www.chessgames.com/perl/chessgame?gid=1070876>

Po rozgrywce Deep Blue przeszedł liczne modyfikacje i ponownie zagrał przeciw Kasparovowi w 1997 roku tym razem odnosząc zwycięstwo nad arcymistrzem 3½–2½. Garry Kasparov po meczu oskarżył IBM o oszustwo uznając, że wbrew regulaminowi programiści dokonywali modyfikacji w oprogramowaniu podczas meczu i zarządał rewanżu. IBM jednak nie przyznał się do stawianych mu zarzutów i odmówił kolejnej rozgrywki. Superkomputer został zdemontowany a projekt został zamknięty. Ze względu na kontrowersyjne wyniki nigdy nie obliczono rankingu USCF dla Deep Blue.

### Rozwiązania techniczne

Deep Blue był profesjonalnie przygotowaną maszyną przystosowaną tylko do jednego celu, jakim było zwycięstwo nad Garrym Kasparovem. W trakcie meczu w 1997 roku superkomputer był wyposażony w 30 węzłów uzbrojonych w mikroprocesory P2SC o częstotliwości taktowania 120MHz oraz 480 specjalnie zaprojektowanych akceleratorów VLSI. Oprogramowanie, napisane w języku C, działało pod kontrolą systemu operacyjnego AIX opacowanego przez IBM na bazie UNIX. Obliczenia były bardzo silnie zrównoleglone.

W takiej konfiguracji Deep Blue był w stanie przeprowadzić ocenę 200 milionów pozycji na sekundę (co stanowiło dwukrotnie lepszy wynik aniżeli w przypadku wersji z roku 1996). W czasie meczu komputer zajmował 259 pozycję na liście TOP500 najszybszych komputerów świata – wg. Benchmarku LINPACK jego moc obliczeniowa wynosiła 11.38 GFLOPS.



Obrazek 8. Deep Blue.

Źródło: <https://www.flickr.com/photos/amitrajit/5356032927/>

Podczas analizy i generowania ruchów system potrafił przewidywać grę średnio do 6-8 ruchów w przód, jednak w specyficznych przypadkach wartość ta przekraczała 20 ruchów. Wiele parametrów silnika mogło być modyfikowanych, jak np.: priorytet obrony króla ponad zajmowaną powieszchnię na planszy. System docelowo samodzielnie kalibrował te wartości poprzez analizowanie bazy danych z zapisanymi ponad 700000 rozgrywek arcymistrzów szachowych. Sama funkcja oceny pozycji została podzielona na ponad 8000 części, z których większość dotyczyła konkretnych konfiguracji.

Do dzisiejszego dnia Deep Blue pozostaje najszybszym (pod względem ilości analizowanych ruchów na sekundę) komputerem, jaki kiedykolwiek został wystawiony na turnieju WCCC.

## Internetowy rozproszony algorytm szachowy z bazą wiedzy i uczeniem.

### Założenia projektu

Rozwiązanie realizowane przez dwóch studentów Politechniki Gdańskiej, Łukasza Konkola i Macieja Rzeszutka, w ramach pracy magisterskiej w 2005 roku.

Założenie tego projektu dość silnie różni się (z punktu widzenia oprogramowania) od przedstawionych wcześniej rozwiązań. Algorytm miał za zadanie agregować wiele dostępnych już silników szachowych i poprzez algorytmy uczące dokonywać wyboru, którego silnika posłuchać w danym ruchu.

Aby w jakiś sposób podzielić silniki, autorzy dokonują klasteryzacji pozwalającej na przeprowadzenie podziału gry na fazy. Dla każdej z faz wyszukiwane są najlepsze silniki poprzez wytrenowany algorytm genetyczny.

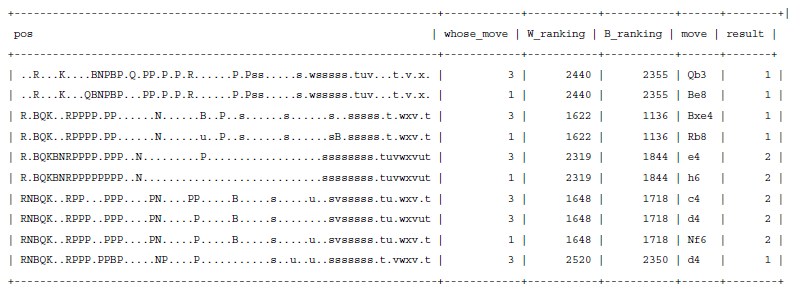
Trening aplikacji odbywa się poprzez rozegranie sztucznych partii szachowych w oparciu o rozproszoną bazę danych zawierającą ponad 8 milionów układów figur szachowych i sprawdzaniu poprawności zwracanych przez algorytmy odpowiedzi. Po wytrenowaniu algorytmu taką ilością danych aplikacja jest gotowa do gry.

Sama rozgrywka może być prowadzona na wirtualnej szachownicy obsługującej notację FEN (np.: eBoard) lub też w trybie textowym (przez TelNet).

Celem projektu było więc nie stworzenie nowego silnika szachowego, lecz wykorzystanie potencjału drzemiącego w mnogich istniejących już rozwiązaniach. Z założenia program miał działać w taki sposób, aby jako całość był w stanie wygrać z każdym z silników, które wchodzą w jego skład.

### Baza danych

W bazie danych przechowywanych było ponad 8 milionów układów figur szachowych wraz z rankingiem ELO osób grających i ostatecznym wynikiem potyczki.



Obrazek 9. Przykładowy fragment bazy danych.

Źródło: Dokumentacja projektu.

### Klasteryzacja

Wykorzystanie klasteryzacji pozwoliło autorom na podział gry na 15 faz. W tym celu przebadali oni 100 tysięcy różnych ustawień figur szachowych. Dla każdej z przebadanych konfiguracji obliczony został wektor 5 znormalizowanych elementów. Z tych później, na podstawie algorytmu K-średnich, wyłonionych zostało 15 środków klasteryzacji. Poniżej zacytowano punkty opisujące, w jaki sposób wypełniane były poszczególne pola wektora:

|  |
| --- |
| * **Wartość materialną figur** – przeglądana jest cała szachownica, i w zależności od ilości i rodzaju figur umieszczonych na szachownicy jest obliczana wartość, która na początku wynosi zero. Każda biała figura zwiększa tą wartość, a czarna zmniejsza. Każdy rodzaj figury jest inaczej oceniany i tak: pion ma wartość 100, wieża – 500, skoczek – 290, goniec – 310, królowa – 950. Króle nie są oceniane, gdyż zawsze musza być oba na planszy i ich wzajemne wartości się redukują. Wartość ta jest normalizowana przez podzielenie przez 7000 i dodanie 0.5. * **Liczba otwartych linii** – mówi nam ile linii kontroluje każdy z kolorów. Figury które mogą kontrolować linie, to gońce, wieże i hetmany. Dla każdej kontrolowanej linii, jeżeli jest to linia prosta (w pionie lub poziomie) to jest dodawana wartość 1,5, zaś jeżeli jest to linia ukośna – dodawana jest wartość 1. Wartość ta jest normalizowana przez podzielenie przez 10. * **Suma ocen każdej z figur** – każda figura ma inne parametry, inne możliwości, także każda jest inaczej oceniana. Dla wieży bierzemy pod uwagę czy znajduje się ona na otwartej linii (jeżeli tak to do wartości dodajemy 10), jak daleko króla przeciwnika się znajduje (2 razy odległość w polach od siebie) i na której linii się znajduje – jeżeli na przedostatniej, a król na ostatniej do dodajemy do otrzymanej uprzednio wartości 40. Dla skoczka brana jest pod uwagę bliskość skoczka od centrum szachownicy. Dla gońca sprawdzamy jak długą możliwość ruchu po przekątnej w każdym kierunku dany goniec posiada. Dla hetmana brana pod uwagę jest bliskość od centrum i bliskość od swojego króla. Dla króla liczone jest jego bezpieczeństwo. Każda pozycja króla jest trochę inaczej liczona. Ogólnie im dalej od swojej początkowej pozycji I pierwszych dwóch linii stoi król tym niższa ocena. Ocena jest podwyższana jeżeli król jest chroniony przez piony. Poza tym, jeżeli król stoi na pierwszej linii to sprawdzamy czy ma on “ furtkę” między pionami w drugiej linii. Ocena ta jest normalizowane przez podzielenie przez 1200 i dodanie 0.33. * **Położenie pionów** – bierzemy tu pod uwagę piony izolowane, zdublowane I sąsiednie, ta także odległość pionów od centrum. Piony izolowane to piony stojące same. Piony zdublowane to piony stojące jeden za drugim. Takie ułożenie jest złe. Pionki sąsiednie to pionki stojące jeden obok drugiego. Odległość od środka – oznacza jak blisko środka piony się znajdują. Ocena ta jest normalizowana przez dzielenie przez 100 i dodanie wartości 0.5 * **Kontrola centrum** – ta ocena, mówi nam w jakim stopniu każdy z graczy kontroluje środek planszy. Oczywiście im większa kontrola tym lepiej. Sprawdzamy czy figury znajdują się w centrum, lub są w stanie w jednym ruchu się tam dostać. Ocena ta normalizowana jest przez podzielenie przez 90 |

Tabela 3. Formuły obliczania wartości wektorów dla klasteryzacji.

Źródło: Dokumentacja projektu

Podczas prowadzenia rozgrywki aktualna sytuacja na szachownicy analizowana była w identyczny sposób. Obliczenie 5-elementowego wektora pozwalało na odniesienie go do przestrzeni z wyspecyfikowanymi centrami klastrowania. Dany punkt przynależał do tej fazy gry, od której centrum klastrowania dzieliła go najmniejsza odległość.

### Algorytm Genetyczny

Algorytm genetyczny składał się z 48 genów, z których każdemu przyporządkowywane jest 15 rekordów (każdy odpowiada pojedynczej fazie gry) z 3 wartościami każdy:

* Silnik, którego ruchy będzie wykonywał
* Czy będzie korzystał z bazy danych
* Jeżeli tak, jakie minimalne ELO graczy go interesuje

Następnie, w fazie uczenia każdy z genów rozgrywa pojedynek przeciw dokładnie dwum losowo wybranym silnikom zarówno jako gracz biały jak i czarny. Daje to 4 rozgrywki. Na podstawie ilości zwycięstw i remisów wyłanianych jest 24 najlepszych genów. Te łączone są w pary i krzyżowane, w skutek czego uzyskujemy 12 potomków. Następnie wartości dla 8 losowo wybranych faz mieszane są między potomkami. W ostatnim etapie uczenia losowo wybranym 10 genom mutuje się 6 losowo wybranych faz.

Podczas uczenia wykonywanych jest wiele iteracji powyższych kroków. Wynik uczenia zapisywany jest w pliku.

# Gra energooszczędna

## Założenia

Gra energooszczędna ma pozwalać użytkownikowi na włączenie opcji uwzględnienia konsumpcji energetycznej poszczególnych, wykorzystanych w systemie silników podczas głosowania. Dodatkowo funkcjonalność ta powinna spełniać następujące założenia:

* Ma pozwalać na oszacowanie/obliczenie ilości prądu zużytego przez silnik. Pomiar nie musi być dokładną wartością wyrażoną w watach. Może być to dowolna jednostka lub zestaw jednostek, które pozwolą na ocenę konsumpcji oraz rankingowanie silników od najmniej do najbardziej prądożernych.
* Będzie łatwo przenaszalna na nowe platformy bez generowania dodatkowych kosztów jak np.: zakup dodatkowego sprzętu.
* Będzie niezależny od sprzętu, na którym jest uruchamiany.
* Będzie działać na maszynach pracujących w środowisku równoległym.
* Będzie działać na maszynach z różnymi dystrybucjami systemów Linux.

## Przegląd problematyki

### Miernik prądu, watomierz

Miernik prądu to urządzenie pozwalające na pomiar zużycia prądu przez dowolne urządzenie podłączone do gniazdka elektrycznego. Jego podstawowymi zaletami jest przedstawienie rzeczywistego zużycia przez podłączone urządzenia z zadaną przez producenta dokładnością. Błąd pomiaru dla danego miernika jest stały, co umożliwia rankingowanie silników na podstawie zebranych przy jego pomocy pomiarów. Jako zewnętrzne urządzenie jest on również nieświadomy i kompletnie niezależny od urządzenia mierzonego. Jest to rozwiązanie szeroko wykorzystywane do pomiarów prądożerności algorytmów choćby na katerze KASK Wydziału ETI Politechniki Gdańskiej.

Rysunek 1. Watomierz

Podstawowym problemem przy zastosowaniu miernika prądu jest konieczność wykonywania pomiarów ręcznie osobno dla każdej maszyny i każdego programu, który chcemy zmierzyć. W przypadku naszej aplikacji, gdyby założyć, że posiadamy 10 silników, które możemy odpalić na jednej z 10 maszyn, musielibyśmy wykonać najmniej 100 pomiarów.

Dodatkowo, mierniki prądu przystosowane są raczej do długotrwałych pomiarów dla sprzętów domowych na potrzeby kontroli domowego zużycia oraz budżetu. Ich dokładność wynosi przeważnie 0.001 kWh. Oznacza to, że aby uzyskać różne wartości dla różnych algorytmów pomiar musiałby trwać co najmniej 1 godzinę (im dłuższy pomiar, tym bardziej zauważalna różnica) oraz należałoby zadbać, aby podczas pomiaru każdy z silników był wykorzystany tyle samo razy. Tym sposobem wykonanie 100 pomiarów trwałoby około 100 godzin, co daje prawie 2,5 roboczo-tygodnia. Po dokonaniu pomiarów dane musiałyby zostać wprowadzone ręcznie do systemu, co wprowadza ryzyko błędu ludzkiego.

Warto również zauważyć, że dla każdej zmiany w mierzonym programie czy też przy każdym dodaniu nowego programu do puli, pomiar dla tego programu musiałby zostać ponownie wykonany dla wszystkich maszyn w systemie.

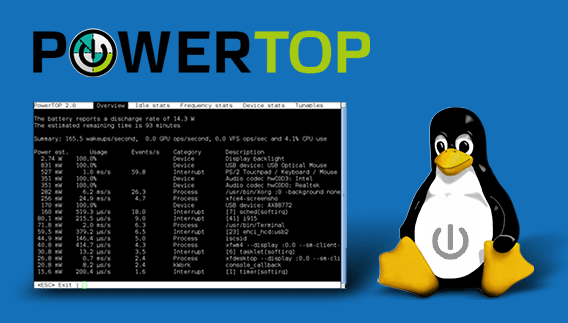
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metoda** | **Szacowanie zużycia** | **Możliwość rankingowania** | **Tania i łatwa przenaszalność** | **Niezależność sprzętowa** | **Działanie w środ. równoległym** | **Różne dystrybujce linuxa** |
| **Watomierz** | **Tak** | **Tak (ręczna)** | **Nie** | **Tak** | **Tak** | **Tak** |

### PowerStat, PowerTOP

Zarówno PowerStat jak i PowerTop to łatwo dostępne, gotowe rozwiązania pozwalające na pomiar energii zużywanej przez komputer w zadanym okresie czasu. Dodatkowo PowerTOP pozwala na rozróżnienie zużycia przez poszczególne procesy. Niestety niosą one za sobą również duże ograniczenia z punktu widzenia projektu.



Aplikacja PowerStat została zaprojektowana na systemy z rodziny Ubuntu oraz pozwala na prowadzenie pomiarów tylko na urządzeniach przenośnych (laptopy, notebooki, itd.), które w danym momencie nie są podłączone do zasilania.



Nieco lepiej ma się sytuacja w przypadku PowerTOP. Jest to rozwiązanie dostarczone przez firmę Intel wpierające systemy Linux oraz Solaris. Pozwala ono na prowadzenie pomiarów dla procesorów Intel, AMD czy ARM. Niestety w dalszym ciągu pozwala na obliczanie zużycia tylko na platformach przenośnych i mobilnych.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metoda** | **Szacowanie zużycia** | **Możliwość rankingowania** | **Tania i łatwa przenaszalność** | **Niezależność sprzętowa** | **Działanie w środ. równoległym** | **Różne dystrybujce linuxa** |
| **PowerStat** | **Tak** | **Tak (nakładka)** | **Tak** | **Nie** | **Tak** | **Nie** |
| **PowerTOP** | **Tak** | **Tak (nakładka)** | **Tak** | **Nie** | **Tak** | **Tak** |

### Obliczanie zużycia na podstawie obciążenia

Pracownicy Francuskiego Narodowego Instytutu Nauk Komputerowych INRIA opublikowali w 2012 roku pracę traktującą o dynamicznym obliczaniu zużycia prądu na poziome procesu dla dowolnego systemu operacyjnego.

Rozwiązanie oparte o przedstawione przez nich rozwiązania pozwoliłyby na wykonywanie pomiarów na dowolnym systemie na poziomie procesu. Niestety, aby taki pomiar był możliwy w najprostszym możliwym scenariuszu musi być znany maksymalny oraz minimalny pobór prądu przez mierzone urządzenie oraz poziom jego obciążenia przed dany proces. Wykorzystanie takiego mechanizmu wymagałoby zatem wprowadzenia specyfikacji każdego sprzętu znajdującego się na maszynach wykonujących obliczenia. Dodatkowo informacje o maksymalnym oraz minimalnym poborze nie zawsze podawane są przez producentów.

Warto również zauważyć, że scenariusz podstawowy zakłada, że zużycie energii przyrasta liniowo do obciążenia, co nie jest prawdą. Aby pomiar był bardziej dokładny wymagane jest wykonanie pomiarów dla różnych poziomów obciążenia sprzętu i przeprowadzenie interpolacji dla wartości niezmierzonych. Oznacza to wykonanie dodatkowych ręcznych pomiarów dla każdego nowo instalowanego sprzętu oraz dodatkowe informacje, które trzeba by wprowadzić do systemu.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metoda** | **Szacowanie zużycia** | **Możliwość rankingowania** | **Tania i łatwa przenaszalność** | **Niezależność sprzętowa** | **Działanie w środ. równoległym** | **Różne dystrybujce linuxa** |
| **Obliczanie n.p. obciążenia** | **Tak** | **Tak** | **Nie** | **Nie** | **Tak** | **Tak** |

### Performance Counters for Linux (PCL)

PCL to rozwiązanie oparte o liczniki systemowe zawarte w jądrze systemów Linux w wersjach od 3.14 w górę. W skład oferowanych statystyk wchodzą między innymi:

* Ilość migracji między rdzeniami procesora
* Ilość trafionych i chybionych odwołań do pamięci cache
* Ilość wykonanych instrukcji
* Ilość odwołań do pamięci
* I wiele innych.

Łącznie dostępnych jest 40 statystyk. Pozwalają one na monitorowanie wydajności różnych aplikacji jak i całego systemu. Dodatkowo, przy użyciu odpowiednich przełączników, możliwe jest wykonanie pomiaru dla konkretnego, wybranego procesu poprzez podanie jego PID.

Jedynym ograniczeniem niesionym przez PCL jest wymóg dotyczący wersji jądra systemu.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metoda** | **Szacowanie zużycia** | **Możliwość rankingowania** | **Tania i łatwa przenaszalność** | **Niezależność sprzętowa** | **Działanie w środ. równoległym** | **Różne dystrybujce linuxa** |
| **PCL** | **Tak** | **Tak (nakładka)** | **Tak** | **Nie** | **Tak** | **Nie** |

### Model Specific Registers (MSR)

Rejestry MSR pozwalają na odczytywanie danych dostarczanych przez procesory firmy Intel. Od momentu pojawienia się chipset SandyBridge (i5 oraz i7) w 2011 roku pojawiła się również funkcjonalność odczytywania z rejestru danych tych procesorów dotyczących ich poboru energetycznego. Dostępne statystyki są podobne, jak w przypadku PCL. Liczba dostępnych liczników jest co prawda znacznie mniejsza, jednak w zamian uzyskujemy wsparcie sprzętowe i pewność co do otrzymanych informacji.

Odczyt z rejestrów MSR wykorzystywany jest choćby przez aplikację Intel Power Gadget. Jedynym ograniczeniem tego rozwiązania jest konieczność posiadania procesora opartego o chipset SandyBridge lub nowszy.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metoda** | **Szacowanie zużycia** | **Możliwość rankingowania** | **Tania i łatwa przenaszalność** | **Niezależność sprzętowa** | **Działanie w środ. równoległym** | **Różne dystrybujce linuxa** |
| **Rejestry MSR** | **Tak** | **Tak (nakładka)** | **Tak** | **Nie** | **Tak** | **Tak** |

### nVidia Management Library (NVML)

### CUDA

### Performance API (PAPI)

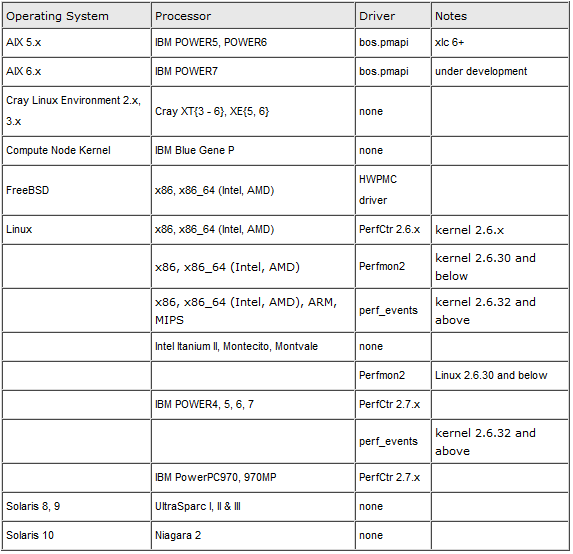
Performance API to opracowane przez zespół z Innovative Computing Labolatory Uniwersytetu w Tennessee rozwiązanie, które łączy zależy wielu popularnych rozwiązań pomiaru zużycia energii. Jego podstawowymi zaletami jest mnogość wspieranych rozwiązań – API to uwspólnia korzystanie z liczników udostępnianych choćby przez Intel (MSR), nVidia (NVML) czy dedykowane liczniki kart wyposażonych z technologię CUDA.

Dodatkowo możliwe jest również zbieranie statystyk na innych płaszczyznach, jak np.: o generowanym czy otrzymywanym przez maszynę ruchu sieciowym.

PAPI charakteryzuje się niskim narzutem podczas prowadzenia pomiarów oraz możliwością jednoczesnego zbierania danych z dowolnej liczby dostępnych liczników. Samo API jest bardzo przystępne i nieźle udokumentowane. Z jego możliwości korzysta wiele firm, jak np.: Intel, HP, IBM, AMD czy Microsoft. Firmy te dodatkowo udzielają finansowania na potrzeby dalszego rozwoju projektu.

Architektura PAPI pozwala na dynamiczne uzyskiwanie informacji o dostępnych komponentach oraz licznikach, ewentualnych błędach ich inicjalizacji itp. Pozwala to na łatwe zarządzanie dostępnymi możliwościami pomiarów na wielu maszynach za pomocą jednego API i łatwą obróbkę tych danych.

PAPI utworzone jest w języku C++ i wspiera następujące systemy oraz architektury:



Z perspektywy projektu istotne są systemy Linux oraz wsparcie dla popularnych w komputerach klasy PC oraz Server procesory. PAPI w tej kwestii stawia niewielkie ograniczenia wymuszające pracę na procesorach Intel, AMD lub IBM na systemach z kernelem w wersji wyższej niż 2.6.32. Ograniczenia te narzucają same komponenty obsługiwane przez PAPI, które nie są dostępne w na innych procesorach lub w starszych wersjach kernela (część z nich została już opisana powyżej).

Dodatkowo PAPI od wersji 5.4.1 wspiera liczniki udostępniane przez CUDA oraz NVML.

**DOPISAC CZYM SA ZDARZENIA NATIVE I PRESET**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metoda** | **Szacowanie zużycia** | **Możliwość rankingowania** | **Tania i łatwa przenaszalność** | **Niezależność sprzętowa** | **Działanie w środ. równoległym** | **Różne dystrybujce linuxa** |
| **PAPI** | **Tak** | **Tak** | **Tak** | **Duża** | **Tak** | **Tak** |

## Podsumowanie

Tabela poniżej zestawia wszystkie przedstawione powyżej rozwiązania.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metoda** | **Szacowanie zużycia** | **Możliwość rankingowania** | **Tania i łatwa przenaszalność** | **Niezależność sprzętowa** | **Działanie w środ. równoległym** | **Różne dystrybujce linuxa** |
| **Watomierz** | **Tak** | **Tak (ręczna)** | **Nie** | **Tak** | **Tak** | **Tak** |
| **PowerStat** | **Tak** | **Tak (nakładka)** | **Tak** | **Nie** | **Tak** | **Nie** |
| **PowerTOP** | **Tak** | **Tak (nakładka)** | **Tak** | **Nie** | **Tak** | **Tak** |
| **Obliczanie n.p. obciążenia** | **Tak** | **Tak** | **Nie** | **Nie** | **Tak** | **Tak** |
| **PCL** | **Tak** | **Tak (nakładka)** | **Tak** | **Tak** | **Tak** | **Nie** |
| **Rejestry MSR** | **Tak** | **Tak (nakładka)** | **Tak** | **Nie** | **Tak** | **Tak** |
| **PAPI** | **Tak** | **Tak** | **Tak** | **Duża** | **Tak** | **Tak** |

Na zestawieniu widać, że najlepiej spełniającym wymagania rozwiązaniem jest PAPI. Łączy ono w sobie zalety wielu innych przedstawianych rozwiązań (CUDA **DODAĆ**, NVML **DODAĆ**, MSR) zapewniając przy tym jedno, spójne API. Dodatkowo jego obsługa jest stosunkowo prosta a dokumentacja całkiem obszerna.

Liczne publikacje na temat PAPI jak i grono wiodących w przemyśle IT firm korzystających z tego rozwiązania i dbających o jego rozwój dodatkowo potwierdza jakość tego produktu.

Licencja PAPI pozwala na redystrybucję wraz z wykorzystującym ją produktem, o ile same źródła pozostały niezmienione a do produktu końcowego dołączona jest licencja PAPI.

**TU: Opis napisac o PAPI Manager, JPapiManager, JNA i dokładności obliczeń. Dodać źródła o PAPI i pozostałych do bibliogafii.**

## Instalacja PAPI

## PapiManager

### Opis

Korzystając z możliwości oferowanych przez PAPI stworzona została biblioteka PapiManager, która dodatkowo upraszcza interface komunikacji z PAPI do funkcji i metod przydatnych w projekcie oraz przekazywania ich za pomocą struktur niosących wszystkie niezbędne dane.

### Diagram klas



### Klasa CPapiManager

#### Opis

Klasa stanowiąca interface komunikacji z całą biblioteką. Udostępnia dostęp do wszystkich funkcjonalności koniecznych do prawidłowego enumerowania dostępnych komponentów czy liczników, rozpoczyniania pomiaru, kończenia go i finalnie zebranie wyników.

#### Pola

|  |  |
| --- | --- |
| **\_isInitialized** | |
| **Zasięg** | private |
| **Typ** | bool |
| **Opis** | Informuje, czy biblioteka PAPI została zainicjalizowana. |

|  |  |
| --- | --- |
| **\_isCounting** | |
| **Zasięg** | private |
| **Typ** | bool |
| **Opis** | Informuje, czy wykonywany jest aktualnie jakiś pomiar. |

|  |  |
| --- | --- |
| **\_instance** | |
| **Zasięg** | private static |
| **Typ** | CPapiManager\* |
| **Opis** | Wskaźnik na instancję obiektu. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **CPapiManager()** | |
| **Zasięg** | private |
| **Zwracany typ** | CPapiManager |
| **Parametry** | Brak. |
| **Opis** | Konstruktor. |
| **Wynik** | Nowa instancja obiektu CPapiManager. |

|  |  |
| --- | --- |
| **ExecuteCmd(…)** | |
| **Zasięg** | private |
| **Zwracany typ** | std::string |
| **Parametry** | * std::string cmd   Komenda, jaka ma zostać wywołana. |
| **Opis** | Wywołuje komendę *cmd*, po czym zwraca wynik jej działania. |
| **Wynik** | Wynik działania komendy *cmd*. |

|  |  |
| --- | --- |
| **GetComponentEventsInfo(…)** | |
| **Zasięg** | private |
| **Zwracany typ** | std::vector<CEventInfo\*> |
| **Parametry** | * int componentId   Identyfikator komponentu PAPI. |
| **Opis** | Wyszukuje wszystkie zdarzenia, jakich pomiar możliwy jest w ramach komponentu o identyfikatorze *componentId*. |
| **Wynik** | Wektor wskaźników na obiekty CEventInfo opisujących zdarzenia dostępne w ramach komponentu o identyfikatorze *componentId*. |

|  |  |
| --- | --- |
| **GetComponentNativeEventsInfo(…)** | |
| **Zasięg** | private |
| **Zwracany typ** | void |
| **Parametry** | * int componentId   Identyfikator komponentu PAPI.   * std::vector<CEventInfo\*>\* componentEventsInfo   Wskaźnik na wektor wskaźników na obiekty CEventInfo. |
| **Opis** | Wyszukuje wszystkie zdarzenia typu Native, jakich pomiar możliwy jest w ramach komponentu o identyfikatorze *componentId* poprzed wywołanie funkcji GetComponentEventsInfoGeneric(…) z odpowiednimi parametrami. |
| **Wynik** | Dodanie informacji o zdarzeniach typu Native do wektora *componentEventsInfo*. |

|  |  |
| --- | --- |
| **GetComponentPresetEventsInfo(…)** | |
| **Zasięg** | private |
| **Zwracany typ** | void |
| **Parametry** | * int componentId   Identyfikator komponentu PAPI.   * std::vector<CEventInfo\*>\* componentEventsInfo   Wskaźnik na wektor wskaźników na obiekty CEventInfo. |
| **Opis** | Wyszukuje wszystkie zdarzenia typu Preset, jakich pomiar możliwy jest w ramach komponentu o identyfikatorze *componentId* poprzed wywołanie funkcji GetComponentEventsInfoGeneric(…) z odpowiednimi parametrami. |
| **Wynik** | Dodanie informacji o zdarzeniach typu Preset do wektora *componentEventsInfo*. |

|  |  |
| --- | --- |
| **GetComponentEventsInfoGeneric(…)** | |
| **Zasięg** | private |
| **Zwracany typ** | void |
| **Parametry** | * int componentId   Identyfikator komponentu PAPI.   * int eventType   Wartość informująca o typie szukanych zdarzeń zgodna z typem wyliczeniowym CEventInfo::EventType.   * std::vector<CEventInfo\*>\* componentEventsInfo   Wskaźnik na wektor wskaźników na obiekty CEventInfo. |
| **Opis** | Wyszukuje wszystkie zdarzenia typu *eventType*, jakich pomiar możliwy jest w ramach komponentu o identyfikatorze *componentId*, poprzez komunikację z PAPI. |
| **Wynik** | Dodanie informacji o zdarzeniach typu *eventType* do wektora componentEventsInfo. |

|  |  |
| --- | --- |
| **GetInstance()** | |
| **Zasięg** | public static |
| **Zwracany typ** | CPapiManager\* |
| **Parametry** | Brak |
| **Opis** | Zwraca instancję obiektu CPapiManager *\_instance* lub tworzy nową, jeżeli ta nie została wcześniej utworzona. |
| **Wynik** | Wskaźnik na instancję obiektu CPapiManager. |

|  |  |
| --- | --- |
| **~CPapiManager()** | |
| **Zasięg** | public |
| **Zwracany typ** | Brak. |
| **Parametry** | Brak. |
| **Opis** | Destruktor. |
| **Wynik** | Zwolnienie zasobów. |

|  |  |
| --- | --- |
| **Init()** | |
| **Zasięg** | public |
| **Zwracany typ** | bool |
| **Parametry** | Brak. |
| **Opis** | Inicjalizuje PAPI oraz ustawia flagę *\_isInitialized.* |
| **Wynik** | Aktualna wartość flagi *\_isInitialized*:   * true – gdy inicjalizacja PAPI powiodła się * false – gdy inicjalizacja PAPI się nie powiodła |

|  |  |
| --- | --- |
| **Close()** | |
| **Zasięg** | public |
| **Zwracany typ** | bool |
| **Parametry** | Brak. |
| **Opis** | Zamyka PAPI wcześniej przerywając pomiary, jeżeli takie były jeszcze prowadzone. Ustawia flagę *\_isInitalized*. |
| **Wynik** | Negację aktualnej wartości flagi *\_isInitialized*:   * true – gdy zamknięcie PAPI powiodło się. * false – gdy zamknięcie PAPI się nie powiodło. |

|  |  |
| --- | --- |
| **StartMeasure(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | bool |
| **Parametry** | * std::vector<CEventInfo\*>\* events   Wektor ze wskaźnikami na obiekty CEventInfo, zawierającymi opisy zdarzeń udostępnianych przez PAPI. |
| **Opis** | Uruchamia pomiar zdarzeń, których opis znajduje się w wektorze *events* poprzez przekazanie ich do funkcji CPapiMeasure::Start(…). |
| **Wynik** | Informacja o poprawności uruchomienia pomiarów:   * true – gdy pomiary zostały uruchomione prawidłowo. * false – gdy podczas uruchamiania pomiarów wystąpił błąd lub gdy jakiś pomiar jest już prowadzony. |

|  |  |
| --- | --- |
| **StopMeasure()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::vector<CEventResult\*>\* |
| **Parametry** | Brak. |
| **Opis** | Kończy pomiar zdarzeń, dla których poprzednio wykonana została funkcja CPapiManager::Start(…), poprzez wywołanie funkcji CPapiMeasure::Stop(). |
| **Wynik** | Wektor wskaźników na obiekty CEventResult zawierające wyniki przeprowadzonych pomiarów. NULL w przypadku błędu podczas zbierania wyników. |

|  |  |
| --- | --- |
| **GetComponentsInfo()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::vector<CComponentInfo\*>\* |
| **Parametry** | Brak. |
| **Opis** | Wyszukuje wszystkie komponenty PAPI dostępne w systemie poprzez komunikację z PAPI. Dodatkowo dla każdego znalezionego komponentu wyszukuje wszystkie dostępne w jego obrębie zdarzenia poprzez wywołanie funkcji CPapiManager::GetComponentEventsInfo(…). |
| **Wynik** | Wektor obiektów opisujących wszystkie komponenty PAPI dostępne w systemie. |

|  |  |
| --- | --- |
| **GetPapiNativeAvailStringCmdResult()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Wywołuje komendę *papi\_native\_avail*, która zwraca informację o wszystkich dostępnych w systemie zdarzeniach typu Native. Do wywołania komendy wykorzystywana jest funkcja CPapiManager::ExecuteCmd(…). |
| **Wynik** | Wynik działania komendy *papi\_native\_avail*. |

|  |  |
| --- | --- |
| **GetPapiComponentAvailCmdResult()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Wywołuje komendę *papi\_component\_avail*, która zwraca informację o wszystkich dostępnych w systemie komponentach oraz ich aktualnym stanie. Do wywołania komendy wykorzystywana jest funkcja CPapiManager::ExecuteCmd(…). |
| **Wynik** | Wynik działania komendy *papi\_component\_avail*. |

|  |  |
| --- | --- |
| **GetPapiAvailCmdResult()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Wywołuje komendę *papi\_avail*, która zwraca informację o wszystkich dostępnych w systemie zdarzeniach typu Preset. Do wywołania komendy wykorzystywana jest funkcja CPapiManager::ExecuteCmd(…). |
| **Wynik** | Wynik działania komendy *papi\_avail*. |

### Klasa CPapiMeasure

#### Opis

Statyczna klasa pozwalająca na prowadzenie pomiarów oraz zbieranie ich wyników za pośrednictwem PAPI.

#### Pola

|  |  |
| --- | --- |
| **\_isCounting** | |
| **Zasięg** | private static |
| **Typ** | bool |
| **Opis** | Informuje, czy wykonywany jest aktualnie jakiś pomiar. |

|  |  |
| --- | --- |
| **\_eventSet** | |
| **Zasięg** | private static |
| **Typ** | int |
| **Opis** | Identyfikator zbioru zdarzeń tworzony przez PAPI. Tworzony jest podczas rozpoczynania pomiarów i przekazywany z powrotem podczas ich kończenia. |

|  |  |
| --- | --- |
| **\_eventsVector** | |
| **Zasięg** | private static |
| **Typ** | std::vector<CEventResult\*>\* |
| **Opis** | Wskaźnik na wektor przechowywujący wskaźniki za obiekty CEventResult. Gdy prowadzony jest pomiar, wskazywany wektor zawiera zbiór odpowiadający aktualnie mierzonym zdarzeniom. Po zakończeniu pomiarów do wektora wpisywanie są wyniki. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **Start(…)** | |
| **Zasięg** | public static |
| **Zwracany typ** | bool |
| **Parametry** | * std::vector<CEventInfo\*>\* events   Wkaźnik na wektor z informacjami o zdarzeniach, które mają podlegać pomiarowi. |
| **Opis** | Funkcja tworzy w PAPI EventSet ze zdarzeniami zawartymi w wektorze *events*. Generuje również dla każdego z obiektów wskazywanego przez wektor *events* odpowiadający mu obiekt CEventResult, do którego później zostanie wpisany wynik pomiaru.  Utworzony EventSet zapisywany jest w polu *\_eventSet*, wygenerowane obiekty przechowywane są w wektorze *\_eventsVector*, a flaga *\_isCounting* zostaje ustawiona na true. |
| **Wynik** | Informacja o poprawności uruchomienia pomiarów:   * true – gdy pomiary zostały uruchomione prawidłowo. * false – gdy podczas uruchamiania pomiarów wystąpił błąd lub gdy jakiś pomiar jest już prowadzony. |

|  |  |
| --- | --- |
| **Stop()** | |
| **Zasięg** | public static |
| **Zwracany typ** | vector<CEventResult\*>\* |
| **Parametry** | Brak. |
| **Opis** | Kończy w PAPI pomiar zdarzeń z EventSet’u *\_eventSet*, po czym przyporządkowuje wyniki odpowiednim obiektom w wektorze *\_eventsVector* i zwraca go. |
| **Wynik** | Wektor wskaźników na obiekty CEventResult zawierające wyniki przeprowadzonych pomiarów. NULL w przypadku błędu podczas zbierania wyników. |

### Klasa CComponentInfo

#### Opis

Klasa opisująca pojedynczy komponent wchodzący w skład PAPI.

#### Pola

|  |  |
| --- | --- |
| **\_name** | |
| **Zasięg** | private |
| **Typ** | std::string |
| **Opis** | Nazwa komponentu. |

|  |  |
| --- | --- |
| **\_events** | |
| **Zasięg** | private |
| **Typ** | std::vector<CEventInfo\*> |
| **Opis** | Wektor zawierający wskaźniki na obiekty niosące informacje o zdarzeniach wchodzących w skład danego komponentu. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **CComponentInfo(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | CComponentInfo\* |
| **Parametry** | * std::string name   Nazwa tworzonego komponentu.   * std::vector<CEventInfo\*> events   Wektor zawierający wskaźniki na obiekty niosące informacje o zdarzeniach wchodzących w skład tworzonego komponentu. |
| **Opis** | Konstruktor. |
| **Wynik** | Nowa instancja obiektu CPapiManager. |

|  |  |
| --- | --- |
| **~CComponentInfo()** | |
| **Zasięg** | public |
| **Zwracany typ** | Brak. |
| **Parametry** | Brak. |
| **Opis** | Destruktor. |
| **Wynik** | Zwolnienie zasobów. |

|  |  |
| --- | --- |
| **GetName()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie nazwy danego komponentu. |
| **Wynik** | Zwraca aktualną wartość pola *\_name*. |

|  |  |
| --- | --- |
| **GetEvents ()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::vector<CEventInfo\*> |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie wektora zawierającego wskaźniki na obiekty niosące informacje o zdarzeniach wchodzących w skład danego komponentu.. |
| **Wynik** | Zwraca aktualną wartość pola *\_events*. |

### Klasa CEventInfo

#### Opis

Klasa opisująca pojedyncze zdarzenie wchodzące w skład danego komponentu PAPI.

#### Pola

|  |  |
| --- | --- |
| **\_name** | |
| **Zasięg** | private |
| **Typ** | std::string |
| **Opis** | Nazwa zdarzenia. |

|  |  |
| --- | --- |
| **\_unit** | |
| **Zasięg** | private |
| **Typ** | std::string |
| **Opis** | Jednostka, w jakich wyrażana jest miara zdarzenia. |

|  |  |
| --- | --- |
| **\_code** | |
| **Zasięg** | private |
| **Typ** | int |
| **Opis** | Kod zdarzenia, identyfikujący je w obrębie PAPI. |

|  |  |
| --- | --- |
| **\_returnDataType** | |
| **Zasięg** | private |
| **Typ** | int |
| **Opis** | Typ danych wyniku pomiaru danego zdarzenia. Rzutowane na typ wyliczeniowy CEventInfo::ReturnValueType. |

|  |  |
| --- | --- |
| **\_eventType** | |
| **Zasięg** | private |
| **Typ** | int |
| **Opis** | Typ zdarzenia PAPI. Rzutowane na typ wyliczeniowy CEventInfo::EventType. |

#### Typy wyliczeniowe

|  |  |
| --- | --- |
| **ReturnDataType** | |
| **Zasięg** | public |
| **Wartości** | * INT64 – wartość wiązana z typem danych long (C++). * UINT64 – wartość wiązana z typem danych unsigned long (C++). * FP64 – wartość wiązana z typem danych double (C++). * BIT64 – wartość wiązana z typem danych unsigned char[8] (C++). |
| **Opis** | Typ danych wyniku zdarzenia PAPI. |

|  |  |
| --- | --- |
| **EventType** | |
| **Zasięg** | public |
| **Wartości** | * NATIVE – wartość wiązana ze zdarzeniami typu Native. * PRESET – wartość wiązana ze zdarzeniami typu Preset. |
| **Opis** | Typ zdarzenia PAPI. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **SetReturnDataType(…)** | |
| **Zasięg** | private |
| **Zwracany typ** | void |
| **Parametry** | * int returnDataType   Typ danych wyniku zdarzenia. Wartości związane z definicjami w PAPI:   * PAPI\_DATATYPE\_INT64 * PAPI\_DATATYPE\_UINT64 * PAPI\_DATATYPE\_FP64 * PAPI\_DATATYPE\_BIT64 |
| **Opis** | Metoda rzutuje wartość parametru *returnDataType* wynikającego z definicji w PAPI na wartość typu wyliczeniowego CEventInfo::ReturnDataType. |
| **Wynik** | Ustawienie pola *\_returnDataType*. |

|  |  |
| --- | --- |
| **ReturnValueTypeToString(…) (zmienić na ReturnDataTypeToString(…))** | |
| **Zasięg** | public static |
| **Zwracany typ** | std::string |
| **Parametry** | * int eventReturnDataType (**zmienić na returnDataType**)   Typ danych wyniku pomiaru zdarzenia. Rzutowane na typ wyliczeniowy CEventInfo::ReturnValueType. |
| **Opis** | Funkcja rzutuje wartość parametru *eventReturnDataType* na string. |
| **Wynik** | Nazwa wartości CEventInfo::ReturnValueType wskazywanej przez argument *eventReturnDataType*. |

|  |  |
| --- | --- |
| **GetEventMaskFromType(…)** | |
| **Zasięg** | public static |
| **Zwracany typ** | int |
| **Parametry** | * int eventType   Typ zdarzenia PAPI. Rzutowane na typ wyliczeniowy CEventInfo::EventType. |
| **Opis** | Zwraca maskę zdefiniowaną w PAPI powiązaną z typem wskazywanym przez parametr *eventType*. |
| **Wynik** | Wartość maski zdefiniowanej w PAPI powiązanej z typem wskazywanym przez parametr *eventType*. |

|  |  |
| --- | --- |
| **CEventInfo(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | CEventInfo\* |
| **Parametry** | * str::string name   Nazwa tworzonego zdarzenia.   * std::string unit   Jednostka, w jakich wyrażana jest miara tworzonego zdarzenia.   * int code   Kod zdarzenia, identyfikujący je w obrębie PAPI.   * int returnDataType   Typ danych wyniku pomiaru danego zdarzenia. Rzutowane na typ wyliczeniowy CEventInfo::ReturnValueType.   * int eventType   Typ zdarzenia PAPI. Rzutowane na typ wyliczeniowy CEventInfo::EventType. |
| **Opis** | Konstruktor. |
| **Wynik** | Nowa instancja obiektu CEventInfo. |

|  |  |
| --- | --- |
| **~ CEventInfo()** | |
| **Zasięg** | public |
| **Zwracany typ** | Brak. |
| **Parametry** | Brak. |
| **Opis** | Destruktor. |
| **Wynik** | Zwolnienie zasobów. |

|  |  |
| --- | --- |
| **GetName()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie nazwy danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_name*. |

|  |  |
| --- | --- |
| **GetUnit()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie jednostki, w jakiej wyrażana jest miara danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_unit*. |

|  |  |
| --- | --- |
| **GetCode()** | |
| **Zasięg** | public |
| **Zwracany typ** | int |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie kodu danego zdarzenia, identyfikującego je w obrębie PAPI. |
| **Wynik** | Zwraca aktualną wartość pola *\_code*. |

|  |  |
| --- | --- |
| **GetReturnDataType()** | |
| **Zasięg** | public |
| **Zwracany typ** | int |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie typu danych wyniku pomiaru danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_returnDataType*. |

|  |  |
| --- | --- |
| **GetEventType()** | |
| **Zasięg** | public |
| **Zwracany typ** | int |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie typu zdarzenia PAPI. |
| **Wynik** | Zwraca aktualną wartość pola *\_eventType*. |

|  |  |
| --- | --- |
| **GetResultClass()** | |
| **Zasięg** | public |
| **Zwracany typ** | CEventResult\* |
| **Parametry** | Brak. |
| **Opis** | Pozwala na stworzenie obiektu klasy CEventResult z wartościami pól odpowiadającymi danemu zdarzeniu. |
| **Wynik** | Zwraca obiekt klasy CEventResult\* z wartościami pól odpowiednimi dla obiektu, który ją wygenerował. |

### Klasa CEventResult

#### Opis

Klasa służąca do przekazywania wyniku pomiaru pojedynczego zdarzenia.

#### Pola

|  |  |
| --- | --- |
| **\_name** | |
| **Zasięg** | private |
| **Typ** | std::string |
| **Opis** | Nazwa zdarzenia. |

|  |  |
| --- | --- |
| **\_unit** | |
| **Zasięg** | private |
| **Typ** | std::string |
| **Opis** | Jednostka, w jakich wyrażana jest miara zdarzenia. |

|  |  |
| --- | --- |
| **\_result** | |
| **Zasięg** | private |
| **Typ** | long long |
| **Opis** | Wynik pomiaru. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **CEventResult(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | CEventResult\* |
| **Parametry** | * str::string name   Nazwa tworzonego zdarzenia.   * std::string unit   Jednostka, w jakich wyrażana jest miara tworzonego zdarzenia. |
| **Opis** | Konstruktor. |
| **Wynik** | Nowa instancja obiektu CEventResult. |

|  |  |
| --- | --- |
| **~ CEventResult()** | |
| **Zasięg** | public |
| **Zwracany typ** | Brak. |
| **Parametry** | Brak. |
| **Opis** | Destruktor. |
| **Wynik** | Zwolnienie zasobów. |

|  |  |
| --- | --- |
| **GetName()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie nazwy danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_name*. |

|  |  |
| --- | --- |
| **GetUnit()** | |
| **Zasięg** | public |
| **Zwracany typ** | std::string |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie jednostki, w jakiej wyrażana jest miara danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_unit*. |

|  |  |
| --- | --- |
| **GetResult()** | |
| **Zasięg** | public |
| **Zwracany typ** | long long |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie wyniku pomiaru danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_result*. |

|  |  |
| --- | --- |
| **SetResult(…) (można usunąc i zfriendować tę klasę z klasą CPapiMeasure)** | |
| **Zasięg** | public |
| **Zwracany typ** | void |
| **Parametry** | * long long result   Wartość, na jaką należy ustawić wynik danego pomiaru. |
| **Opis** | Pozwala na ustawienie wyniku pomiaru danego zdarzenia. |
| **Wynik** | Ustawia wartość pola *\_result*. |

### Klasa CLogger

#### Opis

Klasa umożliwiająca logowanie aktywności i zapisywanie jej do pliku PapiManagerLib.log, który tworzony jest w working directory biblioteki PapiManager.

#### Typy wyliczeniowe

|  |  |
| --- | --- |
| **LogType** | |
| **Zasięg** | public |
| **Wartości** | * Information – wartość związana z logiem informacyjnym. * Warning – wartość związana z logiem ostrzegawczym. * Error – wartość związana z logiem o błędzie. |
| **Opis** | Typ informuje o wadze i znaczeniu logowanej informacji. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **StartLog()** | |
| **Zasięg** | public static |
| **Zwracany typ** | void |
| **Parametry** | Brak. |
| **Opis** | Rozpoczęcie nowego loga. |
| **Wynik** | Wpisanie linijki rozpoczynającej nowy log do pliku z logami. |

|  |  |
| --- | --- |
| **GetName()** | |
| **Zasięg** | public static |
| **Zwracany typ** | void |
| **Parametry** | Brak. |
| **Opis** | Zakończenie aktualnego loga. |
| **Wynik** | Wpisanie linijki kończącej aktualny log do pliku z logami. |

|  |  |
| --- | --- |
| **Log(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | CEventInfo\* |
| **Parametry** | * str::string componentName   Nazwa elementu (klasy, funkcji) biblioteki, z którego wysyłany jest log.   * std::string message   Wiadomość do zalogowania.   * int type   Typ logowanej wiadomości rzutowany na typ wyliczeniowy CLogger::LogType. |
| **Opis** | Zalogowanie wiadomości *message*. |
| **Wynik** | Wpisanie linijki z wiadomością *message* do pliku z logami. |

### Przykładowy kod w C++ korzystający z biblioteki PapiManager

|  |
| --- |
| #include <iostream>  #include <vector>  #include <unistd.h>  #include <cstdlib>  #include <CComponentInfo.h>  #include <CPapiManager.h>  using namespace std;  int main() {  CPapiManager\* papi = CPapiManager::GetInstance();  string choosenEventName = "rapl:::PP0\_ENERGY:PACKAGE0";  CEventInfo\* choosenEventInfo;  papi->Init();  vector<CComponentInfo\*>\* components = papi->GetComponentsInfo();  for(unsigned int i = 0; i < components->size(); i++)  {  cout<<(\*components)[i]->GetName()<<"\n";  vector<CEventInfo\*> events = (\*components)[i]->GetEvents();  for(unsigned int j = 0; j < events.size(); j++)  {  cout<<"\t"<<events[j]->GetName()<<"\n";  if(events[j]->GetName().compare(choosenEventName) == 0)  {  choosenEventInfo = events[j];  }  }  }  cout<<"Creating vectors to measure vector"<<endl;  vector<CEventInfo\*>\* eventsToMeasure = new vector<CEventInfo\*>();  eventsToMeasure->push\_back(choosenEventInfo);  cout<<"Starting measure - Idle 1"<<endl;  if(!papi->StartMeasure(eventsToMeasure))  {  cout<<"Measure start error. Aborting..."<<endl;  return 0;  }  //Do your calculations here  cout<<"Stopping measure"<<endl;  vector<CEventResult\*> results = \*(papi->StopMeasure());  cout<<"Printing results"<<endl;  for(unsigned int e = 0; e < results.size(); e++)  {  cout<<results[e]->GetName()<<" ";  cout<<results[e]->GetResult()<<results[e]->GetUnit()<<endl;  }  papi->Close();    return 0;  } |

## Java Native Access

Java Native Access (JNA) to jedno z rozwiązań pozwalających na wywoływanie funkcji natywnych w języku C z poziomu aplikacji napisanej z Java.

Biblioteka jest stworzona tak, aby za jej pomocą korzystanie z funkcji udostępnianych przez biblioteki natywne było jak najbardziej naturalne i łatwe do zaimplementowania.

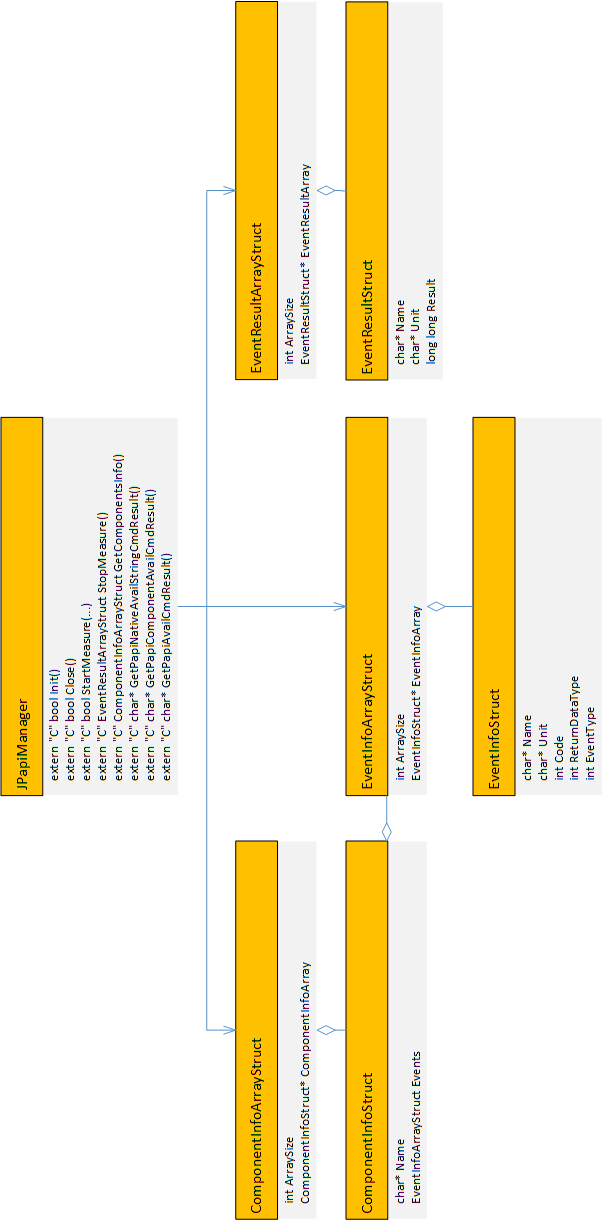
JNA jest projektem Open-Source. Kod rozwiązania dostępny jest pod adresem <https://github.com/java-native-access/jna>. W ramach projektu skorzystano z wersji 4.1.0. Można ją znaleźć wraz z projektem w folderze *external\_dependencies*.

## JPapiManager - C **NOTE: Możnaby zmienić nazwę – jest myląca (J oznacza, ze jest w JAVA, a to C)**

### Opis

JPapiManager to zbiór funkcji i struktur opakowujących klasy C++ w język C. Taki interface może zostać potem wykorzystany do współpracy z Java za pomocą JNA.

### Diagram



### Struktura EventInfoStruct

#### Opis

Struktura reprezentująca klasę CEventInfo.

#### Pola

|  |  |
| --- | --- |
| **Name** | |
| **Typ** | char\* |
| **Opis** | Nazwa zdarzenia. |

|  |  |
| --- | --- |
| **Unit** | |
| **Typ** | char\* |
| **Opis** | Jednostka, w jakich wyrażana jest miara zdarzenia. |

|  |  |
| --- | --- |
| **Code** | |
| **Typ** | int |
| **Opis** | Kod zdarzenia, identyfikujący je w obrębie PAPI. |

|  |  |
| --- | --- |
| **ReturnDataType** | |
| **Typ** | int |
| **Opis** | Typ danych wyniku pomiaru danego zdarzenia. Rzutowane na typ wyliczeniowy CEventInfo::ReturnValueType. |

|  |  |
| --- | --- |
| **EventType** | |
| **Typ** | int |
| **Opis** | Typ zdarzenia PAPI. Rzutowane na typ wyliczeniowy CEventInfo::EventType. |

### Struktura EventInfoArrayStruct

#### Opis

Struktura reprezentująca wektor obiektów klasy CEventInfo.

#### Pola

|  |  |
| --- | --- |
| **ArraySize** | |
| **Typ** | int |
| **Opis** | Rozmiar wektora. |

|  |  |
| --- | --- |
| **EventInfoArray** | |
| **Typ** | EventInfoStruct\* |
| **Opis** | Wskaźnik na pierwszy element wektora. |

### Struktura EventResultStruct

#### Opis

Struktura reprezentująca klasę CEventResult.

#### Pola

|  |  |
| --- | --- |
| **Name** | |
| **Typ** | char\* |
| **Opis** | Nazwa zdarzenia. |

|  |  |
| --- | --- |
| **Unit** | |
| **Typ** | char\* |
| **Opis** | Jednostka, w jakich wyrażana jest miara zdarzenia. |

|  |  |
| --- | --- |
| **Result** | |
| **Typ** | long long |
| **Opis** | Wynik pomiaru. |

### Struktura EventResultArrayStruct

#### Opis

Struktura reprezentująca wektor obiektów klasy CEventResult.

#### Pola

|  |  |
| --- | --- |
| **ArraySize** | |
| **Typ** | int |
| **Opis** | Rozmiar wektora. |

|  |  |
| --- | --- |
| **EventResultArray** | |
| **Typ** | EventResultStruct\* |
| **Opis** | Wskaźnik na pierwszy element wektora. |

### Struktura ComponentInfoStruct

#### Opis

Struktura reprezentująca klasę CComponentInfo.

### Pola

|  |  |
| --- | --- |
| **Name** | |
| **Typ** | char\* |
| **Opis** | Nazwa komponentu. |

|  |  |
| --- | --- |
| **Events** | |
| **Typ** | EventInfoArrayStruct |
| **Opis** | Struktura reprezentująca wektor obiektów klasy CEventInfo niosących informacje o zdarzeniach wchodzących w skład danego komponentu. |

### Struktura ComponentInfoArrayStruct

#### Opis

Struktura reprezentująca wektor obiektów klasy CComponentInfo.

#### Pola

|  |  |
| --- | --- |
| **ArraySize** | |
| **Typ** | int |
| **Opis** | Rozmiar wektora. |

|  |  |
| --- | --- |
| **ComponentInfoArray** | |
| **Typ** | ComponentInfoStruct\* |
| **Opis** | Wskaźnik na pierwszy element wektora. |

### Funkcje i metody

|  |  |
| --- | --- |
| **Init()** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | bool |
| **Parametry** | Brak. |
| **Opis** | Inicjalizuje PAPI poprzez wywołanie funkcji CPapiManager::Init(). |
| **Wynik** | Informacja, czy inicjalizacja zakończyła się powodzeniem:   * true – gdy inicjalizacja PAPI powiodła się. * false – gdy inicjalizacja PAPI się nie powiodła. |

|  |  |
| --- | --- |
| **Close()** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | bool |
| **Parametry** | Brak. |
| **Opis** | Zamyka PAPI poprzez wywołanie funkcji CPapiManager::Close(). |
| **Wynik** | Informacja, czy zamknięcie zakończyła się powodzeniem:   * true – gdy zamknięcie PAPI powiodło się. * false – gdy zamknięcie PAPI się nie powiodło. |

|  |  |
| --- | --- |
| **StartMeasure(…)** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | bool |
| **Parametry** | * EventInfoArrayStruct\* eventsArray   Struktura ze wskaźnikami na struktury EventInfoStruct, zawierającymi opisy zdarzeń udostępnianych przez PAPI. |
| **Opis** | Uruchamia pomiar zdarzeń, których opis znajduje się w strukturze *eventsArray* poprzez przekazanie ich do funkcji CPapiManager::Start(…), wcześniej konwertując dane ze stuktury EventInfoArrayStruct do obiektu std::vector<CEventInfo\*>\* . |
| **Wynik** | Informacja o poprawności uruchomienia pomiarów:   * true – gdy pomiary zostały uruchomione prawidłowo. * false – gdy podczas uruchamiania pomiarów wystąpił błąd lub gdy jakiś pomiar jest już prowadzony. |

|  |  |
| --- | --- |
| **StopMeasure()** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | EventResultArrayStruct |
| **Parametry** | Brak. |
| **Opis** | Kończy pomiar zdarzeń, dla których poprzednio wykonana została funkcja Start(…), poprzez wywołanie funkcji CPapiManager::Stop(). |
| **Wynik** | Struktura EventResultArrayStruct zawierająca wyniki przeprowadzonych pomiarów po wcześniejszym jej przekonwertowaniu z obiektu std::vector<CEventResult\*>\* . |

|  |  |
| --- | --- |
| **GetComponentsInfo()** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | ComponentInfoArrayStruct |
| **Parametry** | Brak. |
| **Opis** | Wyszukuje wszystkie komponenty PAPI poprzez wywołanie funkcji CPapiManager::GetComponentsInfo(). |
| **Wynik** | Struktura ComponentInfoArrayStruct zawierająca opisy wszystkich komponentów PAPI dostępnych w systemie. |

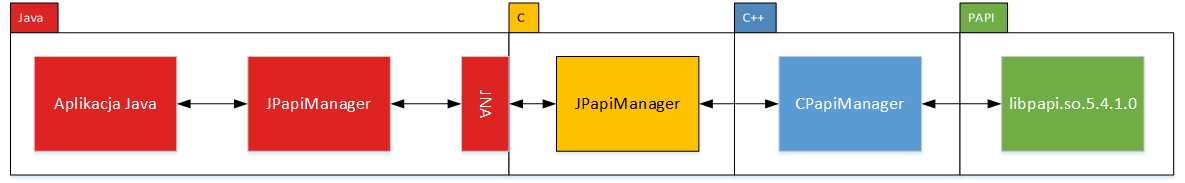
|  |  |
| --- | --- |
| **GetPapiNativeAvailStringCmdResult()** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | char\* |
| **Parametry** | Brak. |
| **Opis** | Wywołuje komendę *papi\_native\_avail*, która zwraca informację o wszystkich dostępnych w systemie zdarzeniach typu Native. Do wywołania komendy wykorzystywana jest funkcja CPapiManager:: GetPapiNativeAvailStringCmdResult(). |
| **Wynik** | Wynik działania komendy *papi\_native\_avail*. |

|  |  |
| --- | --- |
| **GetPapiComponentAvailCmdResult()** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | char\* |
| **Parametry** | Brak. |
| **Opis** | Wywołuje komendę *papi\_component\_avail*, która zwraca informację o wszystkich dostępnych w systemie komponentach oraz ich aktualnym stanie. Do wywołania komendy wykorzystywana jest funkcja CPapiManager:: GetPapiComponentAvailCmdResult(). |
| **Wynik** | Wynik działania komendy *papi\_component\_avail*. |

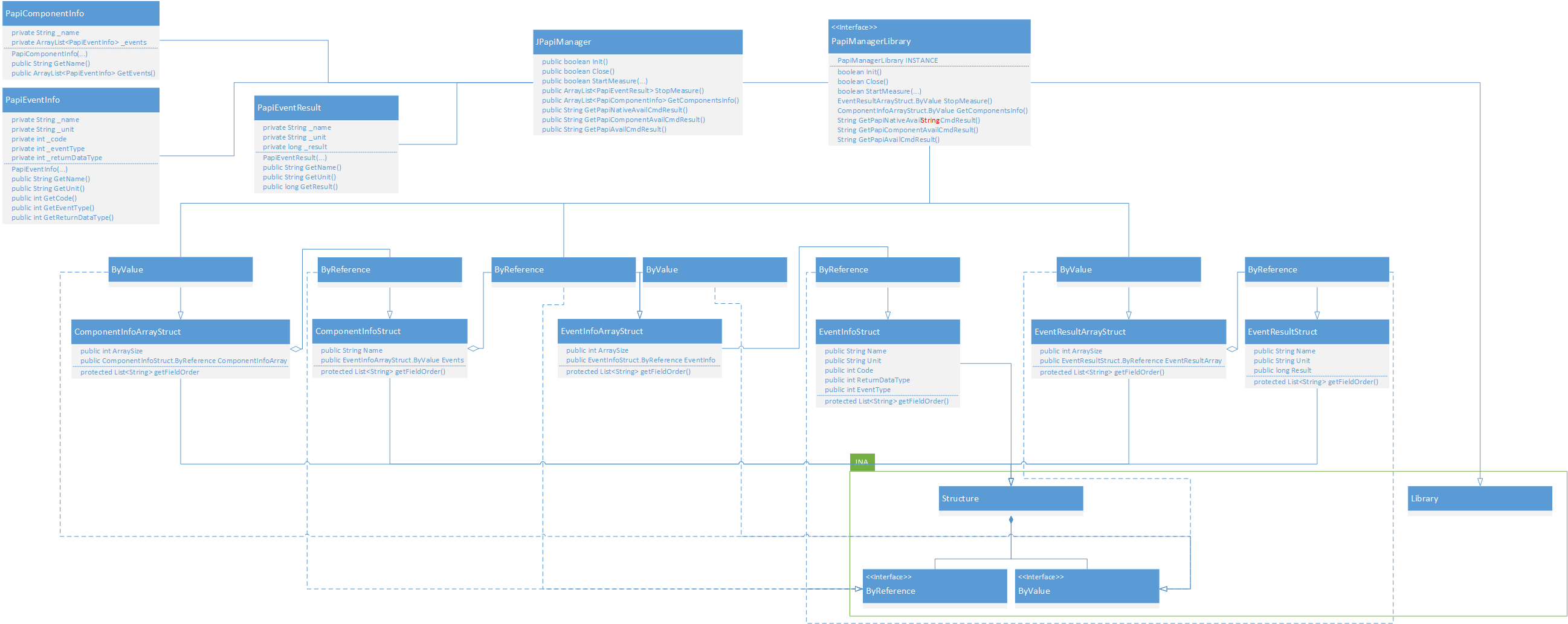
|  |  |
| --- | --- |
| **GetPapiAvailCmdResult()** | |
| **Zasięg** | extern „C” |
| **Zwracany typ** | char\* |
| **Parametry** | Brak. |
| **Opis** | Wywołuje komendę *papi\_avail*, która zwraca informację o wszystkich dostępnych w systemie zdarzeniach typu Preset. Do wywołania komendy wykorzystywana jest funkcja CPapiManager:: GetPapiAvailCmdResult(). |
| **Wynik** | Wynik działania komendy *papi\_avail*. |

## JPapiManager – Java

### Opis

Klasa zawierająca interface komunikacji z biblioteką PapiManager. Komunikacja między aplikacją w Java a biblioteką PAPI odbywa przebiega następująco:

### Diagram klas



### Interface PapiManagerLibrary

#### Opis

Interface rozszerzający klasę JNA Library. Interface pozwala na załadowanie i komunikację z udostępnianymi przez bibliotekę funkcjami oraz korzystanie z udostępnianych struktur.

#### Pola

|  |  |
| --- | --- |
| **INSTANCE** | |
| **Typ** | PapiManagerLibrary |
| **Opis** | Instancja biblioteki PapiManager zrzutowana na interface PapiManagerLibrary. |

#### Zagnieżdżone klasy

|  |  |
| --- | --- |
| **EventInfoStruct** | |
| **Zasięg** | public |
| **Opis** | Klasa reprezentująca strukturę JPapiManager::EventInfoStruct. |

|  |  |
| --- | --- |
| **EventInfoArrayStruct** | |
| **Zasięg** | public |
| **Opis** | Klasa reprezentująca strukturę JPapiManager:: EventInfoArrayStruct. |

|  |  |
| --- | --- |
| **EventResultStruct** | |
| **Zasięg** | public |
| **Opis** | Klasa reprezentująca strukturę JPapiManager:: EventResultStruct. |

|  |  |
| --- | --- |
| **EventResultArrayStruct** | |
| **Zasięg** | public |
| **Opis** | Klasa reprezentująca strukturę JPapiManager:: EventResultArrayStruct. |

|  |  |
| --- | --- |
| **ComponentInfoStruct** | |
| **Zasięg** | public |
| **Opis** | Klasa reprezentująca strukturę JPapiManager:: ComponentInfoStruct. |

|  |  |
| --- | --- |
| **ComponentInfoArrayStruct** | |
| **Zasięg** | public |
| **Opis** | Klasa reprezentująca strukturę JPapiManager:: ComponentInfoArrayStruct. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **Init()** | |
| **Zwracany typ** | boolean |
| **Parametry** | Brak |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::Init(). |

|  |  |
| --- | --- |
| **Close()** | |
| **Zwracany typ** | boolean |
| **Parametry** | Brak |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::Close(). |

|  |  |
| --- | --- |
| **StartMeasure(…)** | |
| **Zwracany typ** | boolean |
| **Parametry** | * EventInfoArrayStruct.ByReference   Obiekt zawierający opisy zdarzeń udostępnianych przez PAPI |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::StartMeasure(…). |

|  |  |
| --- | --- |
| **StopMeasure()** | |
| **Zwracany typ** | EventResultArrayStruct.ByValue |
| **Parametry** | Brak |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::StopMeasure(). |

|  |  |
| --- | --- |
| **GetComponentsInfo()** | |
| **Zwracany typ** | ComponentInfoArrayStruct.ByValue |
| **Parametry** | Brak |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::GetComponentsInfo(). |

|  |  |
| --- | --- |
| **GetPapiNativeAvailStringCmdResult()** | |
| **Zwracany typ** | String |
| **Parametry** | Brak |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::GetPapiNativeAvailStringCmdResult(). |

|  |  |
| --- | --- |
| **GetPapiComponentAvailCmdResult()** | |
| **Zwracany typ** | String |
| **Parametry** | Brak |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::GetPapiComponentAvailCmdResult(). |

|  |  |
| --- | --- |
| **GetPapiAvailCmdResult()** | |
| **Zwracany typ** | String |
| **Parametry** | Brak |
| **Opis** | Klasa reprezentująca funkcję JPapiManager::GetPapiAvailCmdResult(). |

### Klasa JPapiManager

#### Opis

Klasa JPapiManager jest główną klasą PapiManagera po stronie Java. Jest ona wrapperem na funkcje Interface’u PapiManagerLibrary, dzięki czemu pozostałe elementy aplikacji nie muszą być świadome korzystania z JNA oraz jej rozszerzeń pozwalających na raczenie sobie z obiektami w C (obiekty ByReference i ByValue). Wewnątrz klasy znajduje się również sama implementacja interface’u PapiManagerLibrary. Zapewnia ona transparentną dla całej aplikacji możliwość komunikacji z biblioteką PAPI.

#### Interface’u

|  |  |
| --- | --- |
| **PapiManagerLibrary** | |
| **Zasięg** | private |
| **Opis** | Interface JNA pozwalający na komunikację z biblioteką libPapiManager.so. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **Init()** | |
| **Zwracany typ** | boolean |
| **Parametry** | Brak |
| **Opis** | Wrapper na funkcję PapiManagerLibrary::Init(). |

|  |  |
| --- | --- |
| **Close()** | |
| **Zwracany typ** | boolean |
| **Parametry** | Brak |
| **Opis** | Wrapper na funkcję PapiManagerLibrary::Close(). |

|  |  |
| --- | --- |
| **StartMeasure(…)** | |
| **Zwracany typ** | boolean |
| **Parametry** | * ArrayList<PapiEventInfo> jEventsInfoArray   Lista zawierająca opisy zdarzeń udostępnianych przez PAPI |
| **Opis** | * Wrapper na funkcję PapiManagerLibrary::StartMeasure(…). Wykonuje rzutowanie jEventInfoArray na obiekt typu EventInfoArrayStruct.ByReference. |

|  |  |
| --- | --- |
| **StopMeasure()** | |
| **Zwracany typ** | ArrayList<PapiEventResult> |
| **Parametry** | Brak |
| **Opis** | Wrapper na funkcję PapiManagerLibrary::StopMeasure().Wykonuje rzutowanie wyniku działania funkcji na obiekt typu ArrayList<PapiEventResult>. |

|  |  |
| --- | --- |
| **GetComponentsInfo()** | |
| **Zwracany typ** | ArrayList< PapiComponentInfo> |
| **Parametry** | Brak |
| **Opis** | Wrapper na funkcję PapiManagerLibrary::GetComponentsInfo(). Wykonuje rzutowanie wyniku działania funkcji na obiekt typu ArrayList< PapiComponentInfo>. |

|  |  |
| --- | --- |
| **GetPapiNativeAvailStringCmdResult()** | |
| **Zwracany typ** | String |
| **Parametry** | Brak |
| **Opis** | Wrapper na funkcję PapiManagerLibrary::GetPapiNativeAvailStringCmdResult(). |

|  |  |
| --- | --- |
| **GetPapiComponentAvailCmdResult()** | |
| **Zwracany typ** | String |
| **Parametry** | Brak |
| **Opis** | Wrapper na funkcję PapiManagerLibrary::GetPapiComponentAvailCmdResult(). |

|  |  |
| --- | --- |
| **GetPapiAvailCmdResult()** | |
| **Zwracany typ** | String |
| **Parametry** | Brak |
| **Opis** | Wrapper na funkcję PapiManagerLibrary::GetPapiAvailCmdResult(). |

### Klasa PapiEventInfo

#### Opis

Odpowiednik klasy CEventInfo biblioteki PapiManager w Java.

#### Pola

|  |  |
| --- | --- |
| **\_name** | |
| **Zasięg** | private |
| **Typ** | String |
| **Opis** | Nazwa zdarzenia. |

|  |  |
| --- | --- |
| **\_unit** | |
| **Zasięg** | private |
| **Typ** | String |
| **Opis** | Jednostka, w jakich wyrażana jest miara zdarzenia. |

|  |  |
| --- | --- |
| **\_code** | |
| **Zasięg** | private |
| **Typ** | int |
| **Opis** | Kod zdarzenia, identyfikujący je w obrębie PAPI. |

|  |  |
| --- | --- |
| **\_returnDataType** | |
| **Zasięg** | private |
| **Typ** | int |
| **Opis** | Typ danych wyniku pomiaru danego zdarzenia. Rzutowane na typ wyliczeniowy CEventInfo::ReturnValueType. |

|  |  |
| --- | --- |
| **\_eventType** | |
| **Zasięg** | private |
| **Typ** | int |
| **Opis** | Typ zdarzenia PAPI. Rzutowane na typ wyliczeniowy CEventInfo::EventType. |

#### Typy wyliczeniowe

**DODAC**

#### Funkcje i metody

|  |  |
| --- | --- |
| **PapiEventInfo(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | PapiEventInfo |
| **Parametry** | * String name   Nazwa tworzonego zdarzenia.   * String unit   Jednostka, w jakich wyrażana jest miara tworzonego zdarzenia.   * int code   Kod zdarzenia, identyfikujący je w obrębie PAPI.   * int returnDataType   Typ danych wyniku pomiaru danego zdarzenia. Rzutowane na typ wyliczeniowy CEventInfo::ReturnValueType.   * int eventType   Typ zdarzenia PAPI. Rzutowane na typ wyliczeniowy CEventInfo::EventType. |
| **Opis** | Konstruktor. |
| **Wynik** | Nowa instancja obiektu PapiEventInfo. |

|  |  |
| --- | --- |
| **GetName()** | |
| **Zasięg** | public |
| **Zwracany typ** | String |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie nazwy danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_name*. |

|  |  |
| --- | --- |
| **GetUnit()** | |
| **Zasięg** | public |
| **Zwracany typ** | String |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie jednostki, w jakiej wyrażana jest miara danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_unit*. |

|  |  |
| --- | --- |
| **GetCode()** | |
| **Zasięg** | public |
| **Zwracany typ** | int |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie kodu danego zdarzenia, identyfikującego je w obrębie PAPI. |
| **Wynik** | Zwraca aktualną wartość pola *\_code*. |

|  |  |
| --- | --- |
| **GetReturnDataType()** | |
| **Zasięg** | public |
| **Zwracany typ** | int |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie typu danych wyniku pomiaru danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_returnDataType*. |

|  |  |
| --- | --- |
| **GetEventType()** | |
| **Zasięg** | public |
| **Zwracany typ** | int |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie typu zdarzenia PAPI. |
| **Wynik** | Zwraca aktualną wartość pola *\_eventType*. |

### Klasa PapiEventResult

#### Opis

Odpowiednik klasy CEventResult biblioteki PapiManager w Java.

#### Pola

|  |  |
| --- | --- |
| **\_name** | |
| **Zasięg** | private |
| **Typ** | String |
| **Opis** | Nazwa zdarzenia. |

|  |  |
| --- | --- |
| **\_unit** | |
| **Zasięg** | private |
| **Typ** | String |
| **Opis** | Jednostka, w jakich wyrażana jest miara zdarzenia. |

|  |  |
| --- | --- |
| **\_result** | |
| **Zasięg** | private |
| **Typ** | long |
| **Opis** | Wynik pomiaru. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **PapiEventResult(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | PapiEventResult |
| **Parametry** | * String name   Nazwa tworzonego zdarzenia.   * String unit   Jednostka, w jakich wyrażana jest miara tworzonego zdarzenia. |
| **Opis** | Konstruktor. |
| **Wynik** | Nowa instancja obiektu PapiEventResult. |

|  |  |
| --- | --- |
| **GetName()** | |
| **Zasięg** | public |
| **Zwracany typ** | String |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie nazwy danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_name*. |

|  |  |
| --- | --- |
| **GetUnit()** | |
| **Zasięg** | public |
| **Zwracany typ** | String |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie jednostki, w jakiej wyrażana jest miara danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_unit*. |

|  |  |
| --- | --- |
| **GetResult()** | |
| **Zasięg** | public |
| **Zwracany typ** | long |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie wyniku pomiaru danego zdarzenia. |
| **Wynik** | Zwraca aktualną wartość pola *\_result*. |

### Klasa PapiComponentInfo

#### Opis

Odpowiednik klasy CComponentInfo biblioteki PapiManager w Java.

#### Pola

|  |  |
| --- | --- |
| **\_name** | |
| **Zasięg** | private |
| **Typ** | String |
| **Opis** | Nazwa komponentu. |

|  |  |
| --- | --- |
| **\_events** | |
| **Zasięg** | private |
| **Typ** | ArrayList<PapiEventInfo> |
| **Opis** | Lista zawierająca obiekty niosące informacje o zdarzeniach wchodzących w skład danego komponentu. |

#### Funkcje i metody

|  |  |
| --- | --- |
| **PapiComponentInfo(…)** | |
| **Zasięg** | public |
| **Zwracany typ** | PapiComponentInfo |
| **Parametry** | * String name   Nazwa tworzonego komponentu.   * ArrayList<PapiEventInfo> events   Lista zawierająca obiekty niosące informacje o zdarzeniach wchodzących w skład tworzonego komponentu. |
| **Opis** | Konstruktor. |
| **Wynik** | Nowa instancja obiektu PapiComponentInfo. |

|  |  |
| --- | --- |
| **GetName()** | |
| **Zasięg** | public |
| **Zwracany typ** | String |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie nazwy danego komponentu. |
| **Wynik** | Zwraca aktualną wartość pola *\_name*. |

|  |  |
| --- | --- |
| **GetEvents()** | |
| **Zasięg** | public |
| **Zwracany typ** | ArrayList<PapiEventInfo> |
| **Parametry** | Brak. |
| **Opis** | Pozwala na pobranie wektora zawierającego wskaźniki na obiekty niosące informacje o zdarzeniach wchodzących w skład danego komponentu.. |
| **Wynik** | Zwraca aktualną wartość pola *\_events*. |

### Przykładowy kod korzystający z JPapiManager – Java

|  |
| --- |
| **import** java.util.ArrayList;  **import** PapiManager.\*;  **public** **class** PapiManagerExample  {  **public** **static** **void** main(String[] args)  {  String choosenEventName = "rapl:::PP0\_ENERGY:PACKAGE0";  ArrayList<PapiEventInfo> choosenEvents = **new** ArrayList<PapiEventInfo>();  ArrayList<PapiEventResult> measurementResults = **null**;    /\* PAPI Initialization \*/  **if**(!JPapiManager.Init())  {  System.out.println("Papi not Initalized");  **return**;  }    /\* PAPI Printing PAPI Components and Events \*/  ArrayList<PapiComponentInfo> componentsInfo = JPapiManager.GetComponentsInfo();    **for** (**int** i = 0; i < componentsInfo.size(); i++) {    **for** (**int** j = 0; j < componentsInfo.get(i).GetEvents().size(); j++)  {  **if**(componentsInfo.get(i).GetEvents().get(j).GetName().equals(choosenEventName))  {  choosenEvents.add(componentsInfo.get(i).GetEvents().get(j));  }  }  }    System.out.println("\nStarting measure");  **if**(!JPapiManager.StartMeasure(choosenEvents))  {  System.out.println("Measure start error. Aborting...");  **return**;  }  //Do your calculations here  System.out.println("Stopping measure");  measurementResults = JPapiManager.StopMeasure();    System.out.println("Printing results");    **for** (**int** i = 0; i < measurementResults.size(); i++)  {  System.out.println("\t" + measurementResults.get(i).GetName() + " event result: " + measurementResults.get(i).GetResult() + measurementResults.get(i).GetUnit());  }  /\* PAPI DeInitialization \*/  **if**(!PapiManager.JPapiManager.Close())  {  System.out.println("Papi not Closed");  }    **return**;  }  } |

# Użyte protokoły i interface’y

## Universal Chess Interface (UCI)

### Krótki opis do czego służy protokół

### Jak to działa?

## Forsyth-Edwards Notation (FEN)

### Opis

Notacja FEN bierze swoją nazwę od jej twórcy – Steven’a James’a Edwards’a, amerykańskiego matematyka i programisty specjalizującego się m.in. w pisaniu programów szachowych.

Notacja FEN pozwala na opisanie pojedynczym ciągiem znaków o zadanej strukturze opisać aktualny stan rozgrywki: pozycje wszystkich figur, turę, ruch i półruch. Bazuje ona na opracowanym w XIX wieku systemie zapisu rozgrywki autorstwa David’a Forsyth’a.

FEN jest elementem notacji przebiegu partii szachowych PGN oraz EPD.

### Notacja

Pojedyńczy stan rozgrywki opisywany jest poprzez jednolinijkowy string podzielony na sześć sekcji oddzielonych od siebie spacjami:

|  |  |
| --- | --- |
| <FEN> ::= | <Piece Placement>  ' ' <Side to move>  ' ' <Castling ability>  ' ' <En passant target square>  ' ' <Halfmove clock>  ' ' <Fullmove counter> |

* Piece Placement:

Sekcja opisująca układ figur na szachownicy. Podzielona ona jest na osiem części rozdzielonych znakiem slash (/). Każda z części odpowiada pojedynczemu rankowi (rzędowi) na szachownicy, te zaś ułożone są w kolejności malejącej – począwszy od ranku 8, a na ranku 1 skończywszy.

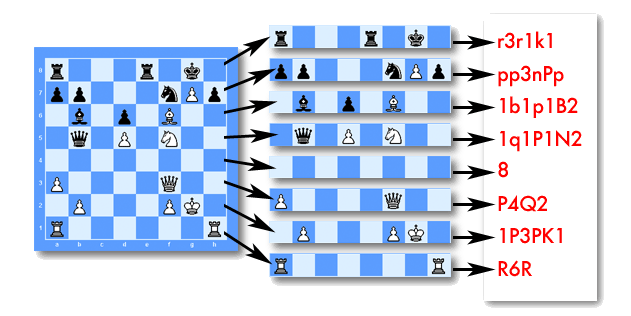
|  |
| --- |
| <Piece Placement> ::=  <rank8>'/'<rank7>'/'<rank6>'/'<rank5>'/'<rank4>'/'<rank3>'/'<rank2>'/'<rank1> |

Na opis każdego z ranków mogą składać się z góry ustalone małe i wielkie litery oraz cyfry:

|  |
| --- |
| <digits> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'  <white Piece> ::= 'P' | 'N' | 'B' | 'R' | 'Q' | 'K'  <black Piece> ::= 'p' | 'n' | 'b' | 'r' | 'q' | 'k' |

Cyfry oznaczają ilość pustych pól, litery z kolei kryją różne figury szachowe:

|  |
| --- |
| 'P' | 'p' => Pawn (pionek)  'N' | 'n' => Knight (skoczek)  'B' | 'b' => Bishop (goniec)  'R' | 'r' => Rook (wieża)  'Q' | 'q' => Queen (królowa)  'K' | 'k' => King (król) |

Poniżej przykład zakodowania stanu szachownicy:

**Obrazek 10.** Kodowanie układu figur w FEN.

**Żródło:** <http://www.chessgames.com/fenhelp.html>

* Side to move:

Określa graczka, który będzie wykonywał kolejny ruch:

|  |
| --- |
| <Side to move> ::= {'w' | 'b'} |

* Castling ability:

Określa roszady dostępne dla każdego z graczy. Sekcja ta może zawierać myślnik (-), oznaczający że żadna roszada nie jest już dostępna w tej rozgrywce, lub od jednej do czterech liter, jak poniżej:

|  |
| --- |
| <Castling ability> ::= '-' | ['K'] ['Q'] ['k'] ['q'] |

Rozmiar litery, tak samo jak poprzednio, określa gracza (wielkie – biały, małe – czarny). Same litery mają następujące znaczenia:

|  |
| --- |
| 'Q' | 'q' => Castling Queen’s Side (roszada długa)  'K' | 'k' => Castling King’s Side (roszada krótka) |

* En passant target square

Pole to niesie informacje o wykonaniu przez pionek przesunięcia o dwa pola w jego pierwszym ruchu. Wtedy, w pole *en passant target square* wpisywane są współrzędne pola przeskoczonego przez pionek – pola pozwalającego na bicie w przelocie (en passant). W przypadku dowolnego innego ruchu w poprzednim półruchu, pole wypełniane jest myślnikiem (-).

|  |
| --- |
| <En passant target square> ::= '-' | <squareCoords>  <squareCoords> ::= <file> <eprank>  <file> ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h'  <eprank> ::= '3' | '6' |

Ważnym jest, że podczas wypełniania tego pola nie jest sprawdzana fizyczna możliwość wykonania takiego bicia. Oznacza to, że pole zostanie wypełnione wartościami współrzędnych pola nawet jeżeli żaden z pionków przeciwnika nie może wykorzystać szansy na bicie w przelocie.

* Halfmove clock

Zegar półruchów. Pozwala na egzekwowanie zasady remisu po wykonaniu 50 półruchów bez bicia. Zegar jest zerowany po biciu lub wykonaniu ruchu pionkiem. We wszystkich innych wypadkach jego wartość jest inkrementowana o jeden.

* Fullmove clock

Zegar ruchów pełnych. Po każdym ruchu gracza grającego pionkami w kolorze czarnym jego wartość jest inkrementowana o jeden.

### Przykład

Poniższy przykład przedstawia notacje FEN dla czterech kolejnych ruchów na szachownicy wraz z jej układem i opisem:

|  |  |  |
| --- | --- | --- |
| **Ruch** | **Szachownica** | **Opis** |
| **Start** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR%20w%20KQkq%20-%200%201&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Stan inicjalny. |
| **Zapis w FEN:** | rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1 | |
| **Pionek**  **E2 -> E4** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR%20b%20KQkq%20e3%200%201&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Podwójny ruch białego pionka z pola E2 na pole E4. Możliwość bicia w przelocie na polu E3. |
| **Zapis w FEN:** | rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1 | |
| **Pionek**  **C7 -> C5** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR%20w%20KQkq%20c6%200%202&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Podwójny ruch czarnego pionka z pola C7 na pole C5. Możliwość bicia w przelocie na rpolu C6. |
| **Zapis w FEN:** | rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR w KQkq c6 0 2 | |
| **Koń**  **G1 -> F3** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R%20b%20KQkq%20-%201%202&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Ruch białego skoczna z pola G1 na pole E3. |
| **Zapis w FEN:** | rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2 | |

**Tabela 4.** Zapis notacji FEN dla kolenych ruchów.

**Generator obrazków:** <http://webchess.freehostia.com/diag/>

## Standard Algebraic Notation (SAN)

### Opis

Algebraic Chess Notation (AN) to zbiór notacji wykorzystujących do opisu ruchu graczy zestaw danych o polu początkowym (Origin Square) oraz końcowym (Target Square). Dane pole opisywane jest poprzez file, rank lub oba te parametry. Dokładny format zapisu zależy od konkretnej notacji AN jak i również figury czy typu ruchu.

Standard Algebraic Notation to notacja wchodząca w skład notacji AN oraz jest elementem zapisu przebiegu partii szachowej w notacji PGN.

### Notacja

Składnia notacji nieco różni się zapisem w zależności od figury i przypadku który opisuje. Zaczynając od najprostrzego:

* Ruch pionkiem:

|  |
| --- |
| <SAN> ::= <to square>[<promoted to>] |

Opis ruchu pionka składa się tylko z informacji o tym, na jakie pole został on przesunięty oraz, jeżeli pionek dotarł do ostatniego rzędu (oznaczonego liczbą 1 lub 8), figury na jaką został wypromowany:

|  |
| --- |
| <promoted to> ::= [ 'N' | 'B' | 'R' | 'Q' ] | [ 'n' | 'b' | 'r' | 'q' ] |

* Bicie pionkiem:

|  |
| --- |
| <SAN> ::= <from file>[<from rank>] 'x' <to square>[<promoted to>] |

W przypadku bicia pionkiem opis zaczyna się od współrzędnych początkowych. Następnie pojawia się znak 'x' informujący o biciu w opisywanych ruchu. Wszystkie kolejne elementy pokrywają się z notacją dla ruchu pionka w przód.

* Ruch dowolną inną figurą:

|  |
| --- |
| <SAN> ::= <Piece symbol>[<from file>|<from rank>|<from square>]['x']<to square> |

W przypadku ruchu dowolną inną figurą notacja rozpoczyna się od jej oznaczenia (literami zgodnie z notacją FEN). Następnie (zależnie do ruchu – pion, poziom, ukos) podawane są niezbędne współrzędne opisującej pole startowe (file, rank lub oba) po którym pojawiają się pełne współrzędne pola docelowego. W przypadku bicia, między danymi o polu począkowym i docelowym podawany jest znak 'x'.

* Roszady:

W notacji SAN wykrzystywany jest standardowy sposób oznaczania roszad dla programowania szachów, pomimo iż wyłamuje się ze standardu narzuconego przez AN:

* + Roszada długa: O-O-O
  + Roszada krótka: O-O

### Przykłady

Poniższy przykład przedstawia notacje SAN dla czterech kolejnych ruchów na szachownicy wraz z jej układem i opisem:

|  |  |  |
| --- | --- | --- |
| **Ruch** | **Szachownica** | **Opis** |
| **Start** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR%20w%20KQkq%20-%200%201&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Stan inicjalny. |
| **Pionek**  **E2 -> E4** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR%20b%20KQkq%20e3%200%201&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Podwójny ruch białego pionka z pola E2 na pole E4. Możliwość bicia w przelocie na polu E3. |
| **Zapis w SAN:** | e4 | |
| **Pionek**  **C7 -> C5** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR%20w%20KQkq%20c6%200%202&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Podwójny ruch czarnego pionka z pola C7 na pole C5. Możliwość bicia w przelocie na rpolu C6. |
| **Zapis w SAN:** | c7 | |
| **Koń**  **G1 -> F3** | http://webchess.freehostia.com/diag/chessdiag.php?fen=rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R%20b%20KQkq%20-%201%202&size=small&coord=yes&cap=no&stm=yes&fb=no&theme=smart&color1=BFBCB6&color2=615F5E&color3=000000 | Ruch białego skoczna z pola G1 na pole E3. |
| **Zapis w SAN:** | Nf3 | |

**Tabela 5.** Zapis notacji SAN dla kolenych ruchów.

**Generator obrazków:** <http://webchess.freehostia.com/diag/>

## Portable Game Notation (PGN)

### Opis

Notacja PGN to powszechnie stosowana przez wiele systemów szachowych metoda zapisu rozgrywki. Została opracowana przez Steven’a James’a Edwards’a (pomysłodawcę FEN) w 1993 roku. Głównymi celami postawionymi przed PGN było:

* Forma przystępna dla każdego człowieka (niezależnie od kraju i pochodzenia)
* Forma przystępna dla maszyny (aplikacji, bazy danych, itd)
* Umożliwienie łatwej komunikacji między graczami i aplikacjami

Pliki w notacji PGN posiadają rozszerzene \*.pgn i przechowywują dane w postaci czystego tekstu kodowanego znakami ASCII. Struktura takiego pliku jest dokładnie ustalona – wykorzystuje m.in. notację FEN.

### Notacja

Plik PGN składa się z dwóch łatwych do rozróżnienia między sobą sekcji:

* Tag Pairs:

Sekcja zawiera zbiór kluczy (tagów) wraz z przypisanymi im wartościami. Ich zadaniem jest przechowywanie informacji opisujących rozgrywkę, np.: w celu późniejszego łatwego wyszukiwania jej poprzez odpowiednie zapytania. Tag Pair zapisywany jest w następującym formacie:

[Nazwa "wartość"]

W ramach PGN zdefiniowanych jest wiele różnych kluczy. Poniżej wymieniono tylko najważniejsze z nich:

|  |  |
| --- | --- |
| **Klucz** | **Opis** |
| **Event** | Nazwa turnieju lub meczu. |
| **Site** | Lokalizacja, w której odbyła się rozgrywka zapisana w formacie Miasto, Region KRAJ, gdzie kraj zapisywany jest trzyliterowym kodem International Olympic Committee. |
| **Date** | Data meczu zapisana w formacie YYYY.MM.DD . |
| **Round** | Numer rundy w turnieju, w ramach której odbywa się rozgrywka. |
| **White** | Osoba grająca białymi pionkami. |
| **Black** | Osoba grająca czarnymi pionkami. |
| **Result** | Wynik gry zapisany w formacie W-B, gdzie:   * W – wynik gracza używającego białych pionków * B – wynik gracza używajączego czarnych pionków   Dopuszczalne są następujące wartości:   * 1 – wygrana * ½ - remis * 0 - przegrana |

* Movetext:

Sekcja w notacji przechowywująca wszystkie ruchy wykonane w trakcie rozgrywki przez obu graczy. Zapis prowadzony jest w delikanie zmienionym formacie FEN – do jego standardowych własności dodane zostały dwa oznaczenia:

* Znak '+' dodany na końcu opisu ruchu oznacza wykonanie szacha.
* Znak '#' dodany na końcu opisu ruchu oznacza wykonanie mata.

Cała sekcja kończy się wynikiem zapisanym zgodnie z formatem taga Result.

### Przykład

Poniżej znajduje się przykładowa partia szachowa zapisana w formacie PGN:

|  |
| --- |
| [Event "F/S Return Match"]  [Site "Belgrade, Serbia Yugoslavia|JUG"]  [Date "1992.11.04"]  [Round "29"]  [White "Fischer, Robert J."]  [Black "Spassky, Boris V."]  [Result "1/2-1/2"]    1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 {This opening is called the Ruy Lopez.} 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5 35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6 1/2-1/2 |

# Aplikacja

## Opis aplikacji

## Nasza baza danych

### Struktura bazy danych

### Przykładowy wpis w bazie

### Przykładowe zapytanie używane w naszej aplikacji

## Klasteryzacja

## Uczenie algorytmem genetycznym

## Architektura

## Instrukcja użytkownika

### Konfiguracja systemu

#### Instalacja PAPI

Niniejsza praca pisana była z wykorzystaniem PAPI w wersji 5.4.1. Najnowsze wersje oprogramowania można pobrać ze strony <http://icl.cs.utk.edu/papi/software/index.html>. Paczka z wersją 5.4.1 została dołączona do pracy w katalogu *external\_dependencies*.

Pełna instrukcja konfiguracji i instalacji PAPI znajduje się w pliku *INSTALL.txt* w głównym katalogu paczki z PAPI. Jedynie w przypadku kroku pierwszego potrzebne są pewne modyfikacje. Do komendy *./configure* należy dodać listę komponentów PAPI, jakie mają zostać dodatkowo zainstalowane. W przypadku tego projektu komenda powinna wyglądać następująco:

./configure –-with-components=”rapl cuda nvml net”

Resztę kroków należy wykonać bez zmian.

Po pomyślnej konfiguracji i instalacji należy stworzyć link lub skopiować wybudowaną bibliotekę do katalogu */usr/lib* :

ln -s /usr/local/lib/libpapi.so.5.4.1.0 /usr/lib/libpapi.so.5.4.1.0

Po jej wykonaniu biblioteka powinna być widoczna dla wszystkich aplikacji.

Po samej instalacji warto sprawdzić, czy wszystkie komponenty działają prawidłowo. W tym celu w terminalu należy wywołać komendę:

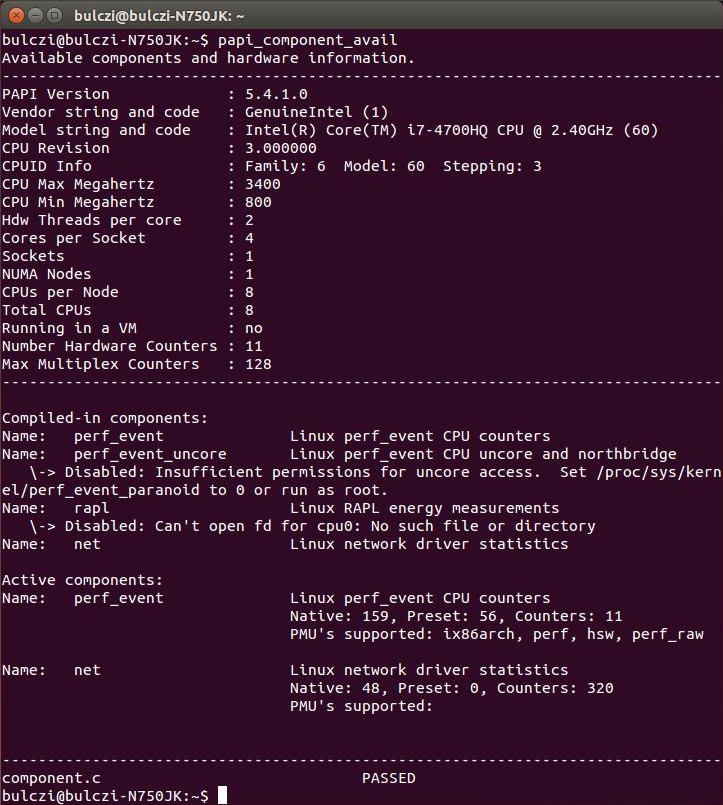
papi\_component\_avail

Podczas pisania pracy natrafiono na trzy przypadki nieprawidłowego działania zainstalowanego komponentu. Oto sposoby na przywrócenie ich do poprawnego działania:

|  |  |
| --- | --- |
| **Komponent** | rapl |
| **Błąd** | Can't open fd for cpu0: No such file or director |
| **Przyczyna** | Najczęściej niezaładowanie modułu kernela pozwalające na korzystanie z rejestrów MSR. |
| **Rozwiązanie** | Uruchom z prawami roota komendę:  sudo modprobe msr |

|  |  |
| --- | --- |
| **Komponent** | rapl |
| **Błąd** | Can't open fd for cpu0: Operation not permitted |
| **Przyczyna** | Brak odpowiednich uprawnień. |
| **Rozwiązanie** | Uruchom aplikację na prawach roota. |

|  |  |
| --- | --- |
| **Komponent** | perf\_event\_uncore |
| **Błąd** | Insufficient permissions for uncore access. Set /proc/sys/kernel/perf\_event\_paranoid to 0 or run as root. |
| **Przyczyna** | Brak odpowiednich uprawnień. |
| **Rozwiązanie** | Uruchom aplikację na prawach roota. |



#### Instalacja biblioteki PapiManager

Wybuduj bibliotekę PapiManager z projektu o tej samej nazwie dostarczonego w ramach niniejszej pracy. Powstałą bibliotekę skopiuj do katalogu */usr/lib*:

cp <eclipse\_workspace>/PapiManager/Release/libPapiManager.so /usr/lib/libPapiManager.so

, lub podlinkuj analogicznie jak w przypadku biblioteki *libpapi.so.5.4.1.0*:

ln -s <eclipse\_workspace>/PapiManager/Release/libPapiManager.so /usr/lib/libPapiManager.so

### Interface uzytkownika

### Gra z uzytkownikiem

### Gra aplikacji samej ze soba

### Gra energooszczędna

<http://www.chesscorner.com>

<https://www.chessprogramming.wikispaces.com>

<http://www.chessgames.com>

<http://webchess.freehostia.com/diag/>

<http://www6.chessclub.com/help/PGN-spec>

<http://www.computerchess.org.uk/>

<http://www.kaims.pl/~goluch/doc/mgr-two-sided.pdf>

<http://www2.cs.uni-paderborn.de/cs/ag-monien/PERSONAL/FLULO/publications/ipdps04.pdf>