UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**

Specjalność: **Informatyka medyczna i telemedycyna**

Łukasz Świerczewski

nr albumu: 260172

Testowanie przypuszczenia Beal'a z wykorzystaniem superkomputerów oraz platformy do obliczeń rozproszonych BOINC

Testing Beal conjecture using supercomputers and distributed computing platform BOINC

Praca magisterska

wykonana w Zakładzie Technologii Informatycznych

pod kierunkiem prof. dr hab. Pawła Mikołajczaka

Lublin rok 2015

Abstrakt

Praca prezentuje aspekt możliwości testowania przypuszczenia Beal'a (ang. *Beal conjecture*) z wykorzystaniem zasobów superkomputerowych oraz platformy BOINC (ang. *Berkeley Open Infrastructure for Network Computing*). Opracowano algorytm poszukujący kontrprzykładu analizowanego problemu matematycznego oraz jego implementację w języku ANSI C. Wykonano testy na takich platformach sprzętowych jak m. in. węzły obliczeniowe superkomputerów IBM BlueGene/P, IBM BlueGene/Q oraz IBM Power 775. Na końcu stworzono projekt rozproszony ***BealF@Home*** w oparciu o BOINC (dostępny pod adresem ***http://bealf.pl***). Zwieńczeniem prac było uruchomienie obliczeń zarówno na superkomputerze ICM, UW *„Hydra”*, jak i komputerach wolontariuszy.

Podziękowania

Obliczenia wykonano w Interdyscyplinarnym Centrum Modelowania Matematycznego i Komputerowego (ICM) Uniwersytetu Warszawskiego w ramach grantu obliczeniowego nr G55-11.

Praca została wykonana z wykorzystaniem Infrastruktury PL-Grid.

PROGRAM
REGIONALNY
NARODOWA STRATEGIA SPÓJNOSCI

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ
ROZWOJU REGIONALNEGO

Spis treści

1. Wprowadzenie

Przypuszczenie Beal'a jest nieudowodnionym twierdzeniem matematycznym z teorii liczb. Mówi ono, że jeśli:

$$x^m + y^n = z^r$$

gdzie x, y, z, m, n oraz r są dodatnimi liczbami całkowitymi, oraz m, n, $r > 2$, to x, y, z mają wspólny dzielnik będący liczbą pierwszą. Z powyższego wynika, że nie znajdziemy rozwiązania powyższego równania dla wartości x, y, z, które są parami względnie pierwsze.

Przypuszczenie zostało sformułowane w roku 1993 przez Andrew'a Beal'a podczas jego prac nad uogólnieniami twierdzenia Fermata. Ufundował on w roku 1997 nagrodę w wysokości $5000 za dostarczenie dowodu lub kontrprzykładu dla swojej teorii. Na przestrzeni lat nagroda była kilkukrotnie podnoszona i w tej chwili (rok 2015) wynosi $1 000 000.

a) Cel pracy

Celem pracy jest komputerowa weryfikacja poprawności przypuszczenia Beal'a w zadanych przedziałach. Szczegółowo tezę pracy można postawić następująco:

Przypuszczenie Beal'a jest prawidłowe dla $0 < \boldsymbol{i} < 5\,000$[1] oraz $9\,999 < \boldsymbol{i} < 10\,601$, gdzie zmienna $\boldsymbol{i}$ definiuje zweryfikowany przedział w jakim mogą znajdować się wartości podstaw $\boldsymbol{x}$, $\boldsymbol{y}$ i $\boldsymbol{z}$ jako:

$$x, y, z \in [\, i \cdot 2000 \,;\, (i + 1) \cdot 2000 \,)$$

Wartości wykładników zawsze są zdefiniowane następująco:

$$m, n, r \in [\, 3 \,;\, 1000 \,]$$

1 Przy $i < 10\,000$ nie zastosowano dwukrotnej weryfikacji danych (zakresy przeliczono tylko raz). W przypadku gdy obliczenia z powodów technicznych zakończyły się błędem (błąd sprzętowy węzła) mogło dojść z pewnym prawdopodobieństwem do pominięcia możliwych rozwiązań, pomimo że teoretycznie po błędzie zadanie powinno zostać jeszcze raz, na nowo załadowane do systemu kolejkowania. Na klastrze zdefiniowany był przez administratorów ICM UW limit 7 dni na wykonanie pojedynczego zadania – sporadyczna część zadań nie zmieściła się w tych ograniczeniach czasowych, chociaż były także i zadania liczące w granicach dwóch dni. Bez podwójnej weryfikacji nie można mieć całkowitej pewności czy nie istnieje w tych zakresach kontrprzykład dla przypuszczenia Beal'a.

i	Ograniczenie dolne przedziału dla podstaw (włącznie)	Ograniczenie górne przedziału dla podstaw (bez)	Ograniczenie dolne przedziału dla wykładników (włącznie)	Ograniczenie górne przedziału dla wykładników (włącznie)
0	0	2 000	3	1 000
1	2 000	4 000		
2	4 000	6 000		
3	6 000	8 000		
… (włącznie)				
5 000	10 000 000	10 002 000	3	1 000
… (bez)				
10 000	20 000 000	20 002 000	3	1 000
… (włącznie)				
10 600	21 200 000	21 202 000	3	1 000

Tab. 1. Tabela przedstawiająca zestawienie zweryfikowanych przedziałów.

Pierwszych 5 000 przedziałów zbadano z wykorzystaniem zasobów superkomputerowych ICM UW[2] (zajęło to w przybliżeniu 600 000 godzin obliczeniowych[3]). Ostatnich 601 przedziałów ($9\,999 < i < 10\,601$) sprawdzono wykorzystując zarówno platformę BOINC, jak i podłączone do niej zasoby superkomputerowe[4] za pomocą specjalnie skonfigurowanego środowiska. Daje to w przybliżeniu kolejnych 21 636 godzin obliczeniowych.

2 Do weryfikacji wykorzystano superkomputer HP BladeSystem/Actina – Hydra.

3 Wykorzystano część superkomputera Hydra zawierającą 120 węzłów złożonych z dwóch procesorów klasy Intel Xeon X5660 oraz 24 GB pamięci RAM. Obliczenia trwały dwa miesiące. W tym okresie jednak nie były wykorzystywane wszystkie zasoby danej części superkomputera (korzystały z niego także inni użytkownicy).

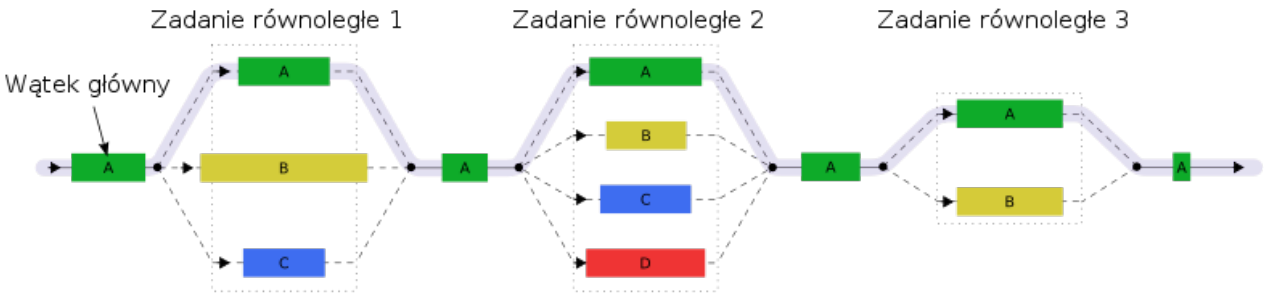
4 W tym etapie obliczeń wzięło udział maksymalnie jedynie 9 węzłów z procesorami Intel Xeon X5660.

2. Programowanie równoległe w standardzie OpenMP

Na potrzeby pracy wykonano analizę przyśpieszenia działania oprogramowania po zrównolegleniu jego za pomocą standardu OpenMP. Standard OpenMP umożliwia jedynie zrównoleglenie z wykorzystaniem możliwości jakie dają komputery z pamięcią wspólną. Aby aplikacje efektywnie realizować na klastrach komputerowych w postaci pojedynczych dużych zadań, należałoby wykorzystać standard MPI. Taka implementacja w dostępnym środowisku superkomputerowym nie miałaby jednak zbyt dużego sensu, ponieważ znacznie prościej jest podzielić obliczenia na kilka mniejszych zadań i wykonywać je za pomocą OpenMP (lub nawet sekwencyjnie) niż tworzyć dedykowane rozwiązanie MPI. W wybranej analizie przypuszczenia Beal'a można było zdecydować się na wrzucanie wielu pojedynczych zadań do systemu kolejkowego. Takie zadania dużo szybciej przeszłyby do fazy wykonywania do procesorach, niż np. instancje wymagające jednocześnie 256 procesorów (z powodu, że zawsze szybciej znajdą się dostępne pojedyncze procesory nich ich bardzo duża pula). Z tych też powodów zdecydowano się w pracy na dwie implementacje:

- sekwencyjną (roboczą),
- zrównoleglonąą w OpenMP (eksperymentalną).

Istota przetwarzania równoległego danych w OpenMP jest bardzo prosta. Wątek główny w określonych sekcjach rozdziela się na zdefiniowaną ilość wątków potomnych (oczywiście definiuje to programista). Ideę tą przedstawiono na Ryc. 1.



Ryc. 1. Przetwarzanie równoległe danych w OpenMP. *Źródło: Wikipedia.pl. Wolna licencja.*

Programista może zrównoleglić określony program (napisany w języku C/C++ lub też Fortran) za pomocą określonych dyrektyw kompilatora. Dyrektywy te mają postać:

#pragma omp <"reszta pragmy">

Najprostszy program jaki można napisać z wykorzystaniem OpenMP może wyglądać następująco:

```
1  int main(int argc, char* argv[])
2  {
3      #pragma omp parallel
4      printf("Witaj Swiecie!\n");
5      return 0;
6  }
```

Listing 1. Pierwszy, najprostszy program z wykorzystaniem OpenMP. *Źródło: Opracowanie własne.*

Jak widać na Listingu 1 w kodzie znajduje się *#pragma* przed instrukcją *printf()*. Pragma ta ma na celu zrównoleglenie instrukcji *printf()*. Oczywiście zrównaleglane mogą być nie tylko pojedyncze instrukcje, a całe ich bloki lub pętle (co może okazać się szczególnie przydatne).

Kolejny przykład (Listing 2) prezentuje właśnie zrównoleglenie pętli za pomocą *#pragma omp parallel for.*

```
1  int main(int argc, char **argv)
2  {
3      const long int SIZE;
4      N = 2500000;
5
6      long int i;
7      long int table[SIZE];
8
9      #pragma omp parallel for
10     for (i = 0; i < SIZE; i++)
11     {
12         table[i] = 5 * i - 1;
13     }
14
15     return 0;
16 }
```

Listing 2. Proste zrównoleglenie pętli z wykorzystaniem *#pragma omp parallel for. Źródło: Opracowanie własne.*

Podczas realizacji wyżej przedstawionego kodu do każdego z wątków zostanie przypisana pewna pula iteracji, które ma on zrealizować. Dzięki temu uruchamiając program na komputerach wielordzeniowych lub nawet wieloprocesorowych możemy uzyskać przyśpieszenie.

Inną często niezbędną funkcjonalnością podczas tworzenia aplikacji równoległych jest sekcja krytyczna, która ma w OpenMP postać *#pragma omp barrier*. Przykładowy kod wykorzystujący sekcję krytyczną zaprezentowano na Listingu 3.

```
1  int main(int argc, char **argv)
2  {
3      long int number_of_primes;
4      number_of_primes = 0;
5
6      long int i;
7
8      #pragma omp parallel for
9      for (i = 2; i < 10000000; i++)
10     {
11         if(isprime(i) == 1)
12         {
13             #pragma omp barrier
14             number_of_primes++;
15         }
16     }
17
18     printf("Ilosc liczb pierwszych: %ld\n", number_of_primes);
19
20     return 0;
21 }
```

Listing 3. Rozwiązanie OpenMP zliczające ilość liczb pierwszych w danym przedziale. *Źródło: Opracowanie własne.*

Gdyby nie zastosowanie pragmy w powyższym rozwiązaniu aplikacja mogłaby zwracać nieprawidłową ilość liczb pierwszych. Pragma gwarantuje, że daną instrukcję (w tym przypadku inkrementację zmiennej) wykona jednocześnie tylko jeden wątek.

Oczywiście to nie wszystkie z możliwości OpenMP, które jest bardzo bogatą biblioteką stosowaną do zrównoleglania w wielu pakietach naukowych (np. GROMACS [18]).

3. Obliczenia rozproszone i platforma BOINC (Berkeley Open Infrastructure for Network Computing)

Obliczenia rozproszone umożliwiają współdzielenie zasobów, które są często rozproszone nawet pod względem geograficznym. O tego typu obliczeniach będziemy mówić jednak nawet wtedy, gdy zasoby nie będą rozproszone pod względem geograficznym, a gdy posiadają architekturę heterogeniczną. Wśród Internautów dużą popularnością cieszą się projekty rozproszone, które umożliwiają bezpłatne udostępnienie mocy obliczeniowych domowych komputerów na potrzeby rozwiązywania bardzo złożonych problemów naukowych. Jedną z takich platform jest BOINC.

a) Obliczenia rozproszone

Obliczenia rozproszone są pewnym podzbiorem obliczeń równoległych. Nie uruchomimy więc efektywnie wszystkich algorytmów równoległych w systemie rozproszonym. Jak się jednak okazuje takie rozwiązanie umożliwia nam dobrą analizę dużej ilości problemów. Systemy rozproszone bazują zazwyczaj na architekturze klient – serwer, której zobrazowanie przedstawiono na Ryc. 2.

Ryc. 2. Architektura klient – serwer. W centrum umieszczony serwer, do którego są podłączeni klienci. *Źródło: Opracowanie własne.*

Architektura klient – serwer, a co za tym idzie także obliczenia rozproszone mają swoje wady. Jedną z nich jest to, że przesyłanie informacji możliwe jest głównie jedynie na liniach klient – serwer oraz w drugą stronę (serwer – klient). Nie ma możliwości samodzielnej łączności pomiędzy klientami, co czasem może bardzo ograniczać programistę.

Do zalet systemów rozproszonych należy jednak to, że umożliwiają one na dostęp do relatywnie dużych zasobów obliczeniowych (w porównaniu do klastrów komputerowych i komputerów z pamięcią wspólną).

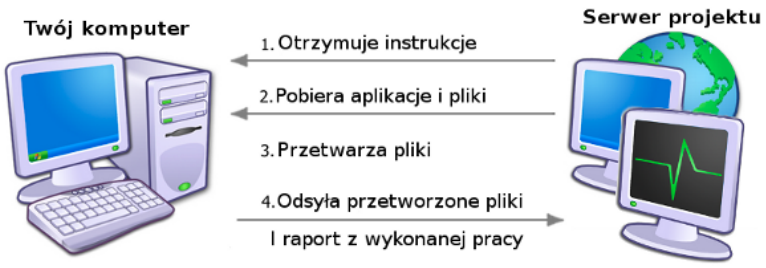
b) Platforma BOINC

Początkowo platforma BOINC była wykorzystywana jedynie przez projekt poszukiwania cywilizacji pozaziemskich SETI@Home. Z czasem jednak zaczęło z niej korzystać wiele ośrodków naukowych na całym świecie. W Tab. 2 zaprezentowano zestawienie przykładowych projektów BOINC.

Nazwa	Kategoria	Dziedzina	Właściciel
Asteroids@Home	Nauki fizyczne	Astrofizyka	Charles University in Prague
ATLAS@Home	Nauki fizyczne	Fizyka	CERN (European Organization for Nuclear Research)
Climateprediction.net	Nauki o Ziemi	Klimatologia	Oxford University
Cosmology@Home	Nauki fizyczne	Astronomia	University of Illinois at Urbana-Champaign
Einstein@Home	Nauki fizyczne	Astrofizyka	Univ. of Wisconsin - Milwaukee, Max Planck Institute
FightNeglectedDiseases@Home	Biologia i medycyna	Poszukiwanie leków przeciw malarii	University College Dublin
GPUGrid.net	Biologia i medycyna	Modelowanie molekularne protein	Barcelona Biomedical Research Park (PRBB)
Malariacontrol.net	Biologia i medycyna	Epidemiologia	The Swiss Tropical Institute
MindModeling@Home	Nauki kognitywne i sztuczna inteligencja	Nauki kognitywne	University of Dayton and Wright State University

Tab. 2. Przykładowe projekty działające na platformie BOINC. *Źródło: Opracowanie własne w oparciu o stronę Uniwersytetu Kalifornijskiego w Berkeley.*

Jak widać w Tab. 2 możliwości BOINC wykorzystuje wiele wiodących instytucji (od CERN po Oxford). Idea tego systemu jest bardzo podobna do innych platform rozproszonych. Schemat działania każdego projektu BOINC zaprezentowano na Ryc. 3.



Ryc. 3. Przepływ danych/instrukcji pomiędzy serwerem BOINC, a klientem. *Źródło: Strona*

zespołu BOINC@Poland.

Pierwszym etapem w komunikacji jest połączenie się klienta z tak zwanym *schedulerem* projektu. To on na podstawie informacji o hoście (rodzaj systemu operacyjnego, typ procesora, ilość pamięci RAM itp.) może wysłać do niego aplikację/aplikacje liczące oraz dane wejściowe. Po pobraniu niezbędnych plików klient przechodzi do obliczeń. W momencie gdy skończy on wykonywać operacje dla danego zadania odsyła przetworzone pliki do serwera. Razem z wynikami wygenerowanymi przez aplikację wysyłany jest także m.in. plik *stderr.txt*, w którym znajduje się zapis wyjścia diagnostycznego aplikacji liczącej.

Należy także dodać, że aplikacje, które wykorzystują projekty BOINC Można podzielić na te które używają intensywnie CPU i takie co nie zajmują czasu CPU praktycznie wcale. Do projektów *Non-CPU-intensive* należy m.in. WUProp@Home, który ma na celu analizę wykorzystania zasobów komputerów klientów przez inne projekty BOINC.

4. Algorytmy testujące przypuszczenie Beal'a i ich implementacje

a) Rozwiązanie Peter'a Norvig'a

Peter Norvig (Dyrektor ds. badawczych Google) na swojej stronie [4] zaprezentował rozwiązanie w języku Python analizujące przypuszczenie Beal'a. Jeden z dwóch kodów źródłowych (ten bardziej zoptymalizowany) zaprezentowano na Listingu 4.

```
1  def beal(max_base, max_power):
2      bases, powers, table, pow = initial_data(max_base, max_power)
3      for x in bases:
4          powx = pow[x]
5          for y in bases:
6              if y > x or gcd(x,y) > 1: continue
7              powy = pow[y]
8              for m in powers:
9                  xm = powx[m]
10                 for n in powers:
11                     sum = xm + powy[n]
12                     r = table.get(sum)
13                     if r: report(x, m, y, n, nth_root(sum, r), r)
```

Listing. 4. Rozwiązanie w języku Python zaproponowane przez Peter'a Norvig'a. *Źródło: [4]*.

W Tab. 3. zaprezentowano główne wyniki jakie uzyskał Peter Norvig [4] realizując kod przedstawiony na Listingu 1. Podczas obliczeń wykorzystano interpreter języka Python w wersji 1.5 oraz procesor o częstotliwości taktowania 400 MHz. Pomiary te wykonano w granicach 2000 roku.

	max_power=7	max_power=10	max_power=30	max_power=100	max_power=1000
max_base=100	-	-	-	0,6	19,0
max_base = 1000	-	-	-	6,2	?
max_base=10000	-	-	52,0	993,0	?
max_base=100000	-	109,0	?	?	?
max_base=250000	1323,0	?	?	?	?

Tab. 3. Czasy realizacji rozwiązania Peter'a Norvig'a przedstawione w [4]. Czasy mierzone w godzinach. *Źródło: [4]*.

Własne wyniki czasowe dla tego samego kodu realizowanego jednak na procesorze Intel Xeon X5660 z wykorzystaniem interpretera Python w wersji 2.6.6 zaprezentowano w Tab. 4.

	max_power=7	max_power=10	max_power=30	max_power=100	max_power=1000
max_base=100	-	-	-	0,003	0,527
max_base = 1000	-	-	-	0,968	?
max_base=10000	-	-	42,257	> 168,000	?
max_base=100000	-	> 168,00	?	?	?
max_base=250000	> 168,00	?	?	?	?

Tab. 4. Czasy realizacji rozwiązania Peter'a Norvig'a z wykorzystaniem procesora Intel Xeon X5660 i Pythona w wersji 2.6.6. Czasy mierzone w godzinach. *Źródło: Opracowanie własne.*

Jak widać uzyskane przyśpieszenie jest w dużym stopniu zależne od parametrów z jakimi był uruchamiany program. Na procesorze 400 MHz i Pythonie 1.5 dla parametrów max_power=100 oraz max_base=100 czas wykonywania wyniósł 0.6 godziny. Po zastosowaniu nowszej wersji Pythona i procesora Intel Xeon X5660 czas ten spadł do 12.86 sekundy (0.003 godziny). Daje to więc przyśpieszenie równe aż równo 200. Jeżeli jednak uwzględnimy większy przedział dla podstaw (max_base=1000) to wartość przyśpieszenia spadnie do 6.404.

b) Implementacja pierwsza (naiwna)

Zaimplementowano funkcję o poniższym nagłówku:

```
unsigned int beal_conjecture_test(
        unsigned long long int min_base,
        unsigned long long int max_base,
        unsigned long long int min_power,
        unsigned long long int max_power,
        beal_test_result_t** results);
```

Gdzie *beal_test_result_t* jest strukturą, w której będą ewentualnie przechowywane odnalezione kontrprzykłady. Struktura ta zdefiniowana jest następująco:

```
typedef struct beal_test_result {
    unsigned long long int x;
    unsigned long long int y;
    unsigned long long int z;
    unsigned long long int m;
    unsigned long long int n;
    unsigned long long int r;
} beal_test_result_t;
```

Grupuje ona wartości wszystkich podstaw i potęg równania $x^m + y^n = z^r$.

Funkcja *beal_conjecture_test* wykonuje zasadniczą część analizy *Przypuszczenia Beala*. Dla podanych przedziałów wartości podstaw (*min_base*, *max_base*) i potęg (*min_power*, *max_power*) generowane są wszystkie możliwe kombinacje wartości *x*, *y*, *z*, *m*, *n*, *r* z równania $x^m + y^n = z^r$. Jeśli po podstawieniu wartości równanie jest spełnione, oznacza to, że został znaleziony kontrprzykład. Wartości wszystkich zmiennych równania zapisywane są wtedy w strukturze *beal_test_result_t* i umieszczane w tablicy takich struktur. Po zapisaniu danych znalezionego rozwiązania, funkcja kontynuuje poszukiwania kolejnych rozwiązań. Po zbadaniu wszystkich kombinacji, tablica struktur jest zwracana przez jeden z parametrów funkcji, a wartość będąca wynikiem funkcji określa ilość znalezionych kontrprzykładów.

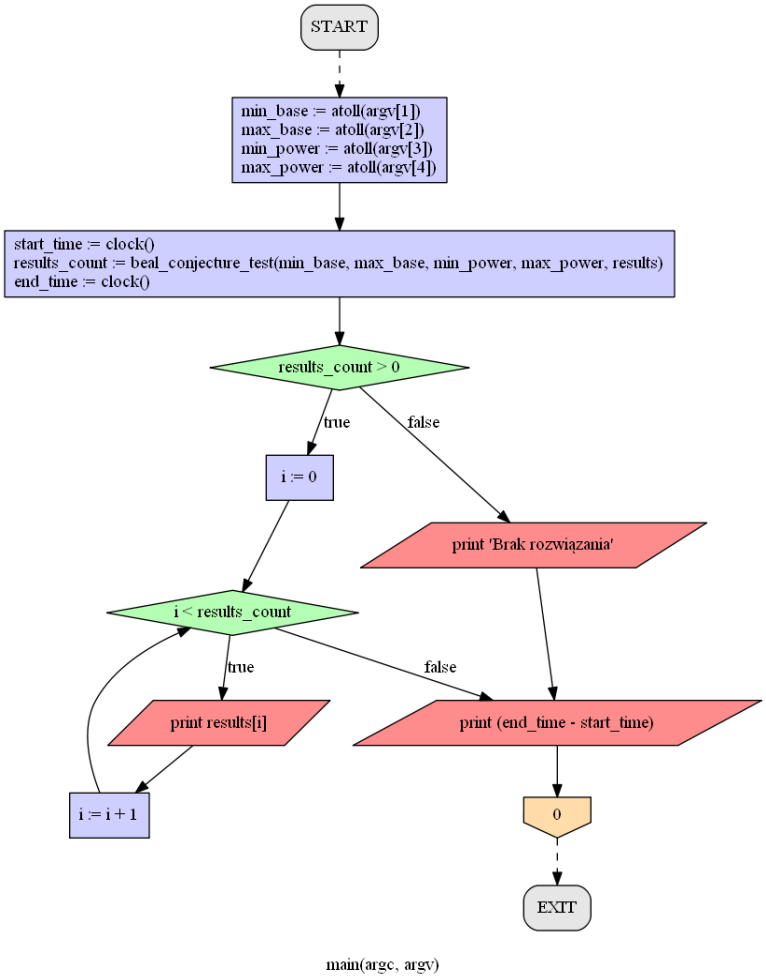
W pierwszej kolejności rozpatrywane są wszystkie kombinacje wartości podstaw *x* i *y* (dwie najbardziej zewnętrzne pętle *for*). Dla tych par, które są względnie pierwsze (*gcd(x,y)==1*), sprawdzane są następnie wszystkie możliwe wartości podstawy *z*. Jeśli *z* jest względnie pierwsze zarówno z *x* jak i z *y*, dla trójki *x*, *y*, *z*, sprawdzane są wszystkie możliwe kombinacje potęg z zadanego przedziału (trzy najbardziej wewnętrzne pętle *for*).

Program operuje na 64-bitowych dodatnich liczbach całkowitych. Wszelkie obliczenia są wykonywane modulo 2^{64}, co stwarza to ryzyko fałszywych trafień. Takie przypadki powinny być wyeliminowane już ręcznie (poza programem). Wydajność implementacji pierwszej (naiwnej) przedstawiono w Tab. 5.

	max_power=7	max_power=10	max_power=30	max_power=100	max_power=1000
max_base=100	0,000	0,000	0,000	0,032	38,223
max_base = 1000	0,016	0,030	0,654	34,780	> 168,000
max_base=10000	19,356	33,189	> 168,000		
max_base=100000	> 168,000				
max_base=250000	> 168,000				

Tab. 5. Czasy realizacji pierwszego rozwiązania napisanego w języku ANSI C na procesorze Intel Xeon X5660 (kompilacja kompilatorem GCC 4.4.7 z optymalizacją trzeciego stopnia O3). Czasy mierzone w godzinach. *Źródło: Opracowanie własne.*

Podstawowy schemat blokowy przedstawiający działanie programu przedstawiono na Ryc. 4.



Ryc. 4. Uproszczony schemat blokowy przedstawiający działanie programu. *Źródło: Opracowanie własne.*

c) Implementacja druga (z dodatkową tablicą)

W celu przyspieszenia wykonywania obliczeń, zoptymalizowana funkcja wykorzystuje dodatkową pamięć (tablicę), w której zapisywane są wszystkie możliwe wyniki potęgowania dla podstaw z przedziału [*min_base*, *max_base*] i potęg z przedziału [*min_power*, *max_power*]. Pierwszy indeks utworzonej tablicy odpowiada wartościom podstawy, spośród tych analizowanych, a drugi indeks – kolejność potęgi. Dla przykładu, w komórce o indeksach [0][0], znajdzie się wartość $min\_base^{min\_power}$, w komórce o indeksach [0][1] – $min\_base^{(min\_power+1)}$, [1][0] – $(min\_base+1)^{min\_power}$, itd. Dzięki takiemu podejściu, rozmiar wymaganej tablicy zależy jedynie od szerokości przedziałów wartości podstaw i potęg. Czasy realizacji analiz danych przedziałów przez oprogramowanie wykorzystujące drugą implementację przedstawiono w Tab. 6.

	max_power=7	max_power=10	max_power=30	max_power=100	max_power=1000
max_base=100	0,000	0,000	0,000	0,026	25,798
max_base = 1000	0,016	0,030	0,662	27,136	> 168,000
max_base=10000	19,851	33,693	> 168,000		
max_base=100000	> 168,000				
max_base=250000	> 168,000				

Tab. 6. Czasy realizacji drugiego rozwiązania napisanego w języku ANSI C na procesorze Intel Xeon X5660 (kompilacja kompilatorem GCC 4.4.7 z optymalizacją trzeciego stopnia O3). Czasy mierzone w godzinach. *Źródło: Opracowanie własne.*

d) Implementacja trzecia (z dwiema dodatkowymi tablicami)

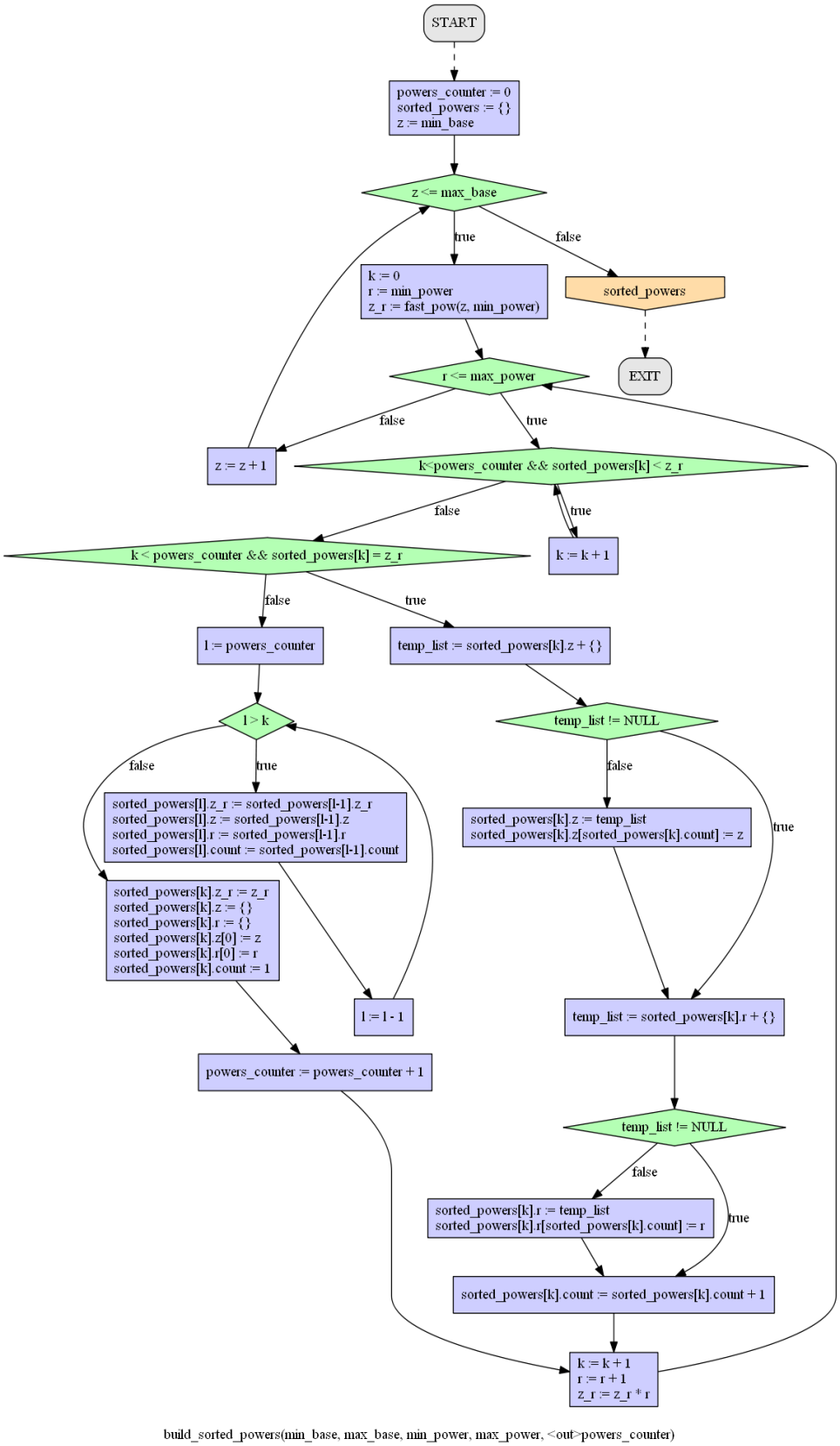
W trzecim algorytmie zastosowanie znajduje również druga tablica (*sorted_powers*), przechowująca struktury *powers_list_t*. Dla każdego możliwego wyniku potęgowania (końcowej wartości), gdzie podstawa jest z przedziału [*min_base, max_base*], a potęga z przedziału [*min_power, max_power*], w tablicy znajduje się dokładnie jeden rekord. Oprócz wyniku potęgowania, przechowywane są również podstawa i wykładnik (potęga). W przypadku, gdy więcej niż jedna kombinacja podstawy i wykładnika daje taki sam wynik (np. $10^4 = 100^2$), w strukturze na danej pozycji zapisane będą wszystkie wartości podstaw i wykładników dających taki wynik.

W tablicy *sorted_powers* będziemy szukać wyników potęgowania po wartości. Aby można było robić to efektywnie, tablica musi być posortowana. Tablica jest sortowana „w locie" podczas tworzenia. Wykorzystujemy fakt, że kolejne potęgi zadanej podstawy tworzą ciąg rosnący, dzięki czemu dodawanie elementów z zachowaniem sortowania jest szybsze, niż gdybyśmy sortowali tablicę po wypełnieniu.

Na Listingu 5 przedstawiono część główną kodu maksymalnie zoptymalizowanej funkcji *beal_conjecture_test* odpowiedzialna za poszukiwanie kontrprzykładu. Schemat blokowy procedury *build_sorted_powers* zaprezentowano na Ryc. 5. Wydajność ostatecznej wersji aplikacji zapisano w Tab. 7.

```
for (xi=0; xi<=max_base_index; ++xi)
{
    x = min_base + xi;

    for (yi=xi+1; yi<=max_base_index; ++yi)
    {
        y = min_base + yi;

        if (gcd(x, y) == 1ULL)
        {
            for (mi=0; mi<=max_power_index; ++mi)
            {
                x_m = powers[xi][mi];

                for (ni=0; ni<=max_power_index; ++ni)
                {
                    y_n = powers[yi][ni];
                    i = binary_search(sorted_powers, 0, powers_counter-1, x_m + y_n);

                    if (i != -1)
                    {
                        for (j=0; j<sorted_powers[i].count; ++j)
                        {
                            z = sorted_powers[i].z[j];

                            if (z != x && z != y && gcd(z, x) == 1ULL && gcd(z, y) == 1ULL)
                            {
                                ++results_count;

                                temp_results = (beal_test_result_t*)
                                    realloc(*results, results_count * sizeof(beal_test_result_t));

                                if (temp_results)
                                {
                                    *results = temp_results;
                                    (*results)[results_count-1].x = x;
                                    (*results)[results_count-1].y = y;
                                    (*results)[results_count-1].z = z;
                                    (*results)[results_count-1].m = min_power+mi;
                                    (*results)[results_count-1].n = min_power+ni;
                                    (*results)[results_count-1].r = sorted_powers[i].r[j];
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Listing 5. Część główna kodu maksymalnie zoptymalizowanej funkcji *beal_conjecture_test* odpowiedzialna za poszukiwanie kontrprzykładu. *Źródło: Opracowanie własne.*



Ryc. 5. Schemat blokowy procedury *build_sorted_powers*. *Źródło: Opracowanie własne.*

	max_power=7	max_power=10	max_power=30	max_power=100	max_power=1000
max_base=100	0,000	0,000	0,000	0,000	0,050
max_base = 1000	0,000	0,000	0,003	0,042	5,921
max_base=10000	0,018	0,046	0,561	5,780	> 168,000
max_base=100000	1,906	5,348	79,925	> 168,000	
max_base=250000	17,543	43,025	> 168,000		

Tab. 7. Czasy realizacji trzeciego rozwiązania napisanego w języku ANSI C na procesorze Intel Xeon X5660 (kompilacja kompilatorem GCC 4.4.7 z optymalizacją trzeciego stopnia O3). Czasy mierzone w godzinach. *Źródło: Opracowanie własne.*

e) Wykorzystane algorytmy elementarne

W funkcji głównej *beal_conjecture_test* byłem zmuszony do zastosowania kilku elementarnych algorytmów. W tym miejscu warto jest poświęcić chwilę na ich analizę.

Szybkie potęgowanie

```
unsigned long long int fast_pow(unsigned long long int p, unsigned long long int q);
```

Funkcja implementuje algorytm szybkiego obliczania potęgi o wykładniku naturalnym. Potęgowanie odbywa się poprzez serię operacji mnożenia. Dzięki wykorzystaniu właściwości binarnej reprezentacji wykładnika, liczba potrzebnych operacji mnożenia jest rzędu $log_2 q$ (metoda naiwna potrzebuje q operacji mnożenia).

Wykorzystujemy fakt, że podnoszenie do potęgi, która sama jest potęgą dwójki, można wykonać przy pomocy serii operacji podniesienia do kwadratu (podnosimy do kwadratu wynik poprzedniej operacji). Jedynki w binarnej reprezentacji liczby naturalnej określają, jakie potęgi dwójki wchodzą w jej skład. Każde potęgowanie możemy rozbić na iloczyn potęg, z których każda ma wykładnik będący potęgą dwójki.

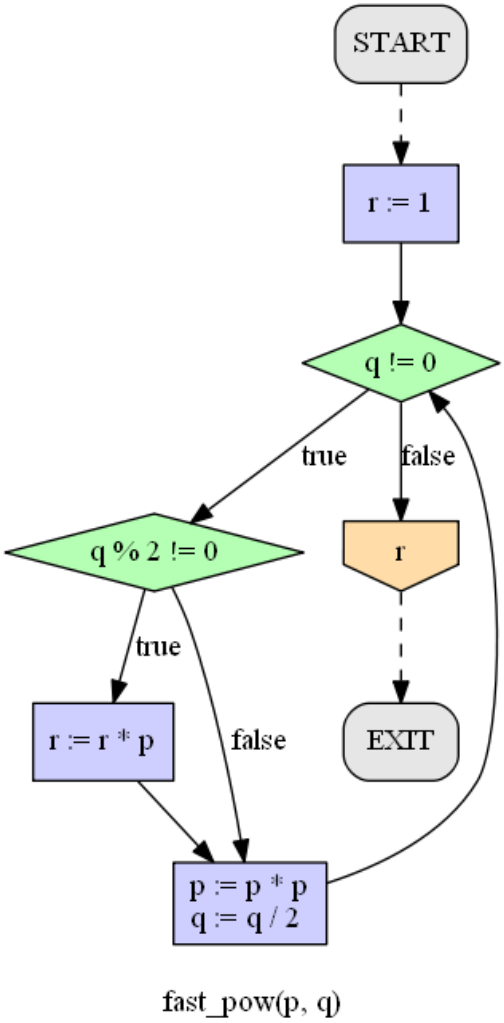
Dla przykładu, rozważmy obliczanie x^{10}. Naiwny sposób obliczania potęgi wymaga w tym przypadku 10 operacji mnożenia ($x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x$). Jeśli natomiast zauważymy, że 10 to w zapisie dwójkowym 1010_2, czyli 2+8, możemy całą operację zapisać jako:
$x^{10} = x^{2+8} = x^2 \cdot x^8 = x^2 \cdot ((x^2)^2)^2$
W takim przypadku wystarczy, że wykonamy cztery mnożenia (trzy podniesienia do kwadratu i mnożenie $x^2 \cdot x^8$).

W naszej implementacji zaczynamy od początkowego wyniku równego 1. Dla każdego bitu wykładnika (zaczynając od najmniej znaczących):

- jeśli bit jest równy 1, przemnóż wynik przez podstawę,
- podnieś podstawę do kwadratu (ten krok wykonujemy niezależnie od wartości bitu).

Operację kończymy po przetworzeniu ostatniego niezerowego bitu wykładnika i zwracamy wyliczony wynik.

Schemat blokowy algorytmu szybkiego potęgowania został przedstawiony na Ryc. 6.



Ryc. 6. Schemat blokowy algorytmu szybkiego potęgowania. *Źródło: Opracowanie własne.*

Największy wspólny dzielnik

```
unsigned long long int gcd(unsigned long long int a, unsigned long long int b);
```

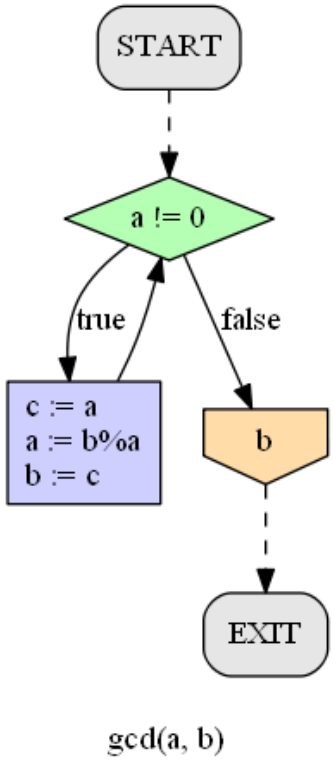
Funkcja oblicza największy wspólnego dzielnik (*nwd*) dwóch liczb naturalnych, wykorzystując algorytm Euklidesa. Algorytm oparty jest na następujących zależnościach:

nwd(b, 0) = b
nwd(b, a) = nwd(a, b mod a)

W naszej implementacji, wartość zmiennych *a* i *b* w kolejnych iteracjach wyliczane są jako:

$a_{t+1} = b_t \bmod a_t$
$b_{t+1} = a_t$
Zatrzymujemy się w momencie, gdy nowa wartość *a* jest równa 0. Wynikiem, czyli największym wspólnym dzielnikiem, jest wartość *b*.

Schemat blokowy algorytmu NWD został przedstawiony na Ryc. 7.



Ryc. 7. Schemat blokowy algorytmu NWD. *Źródło: Opracowanie własne.*

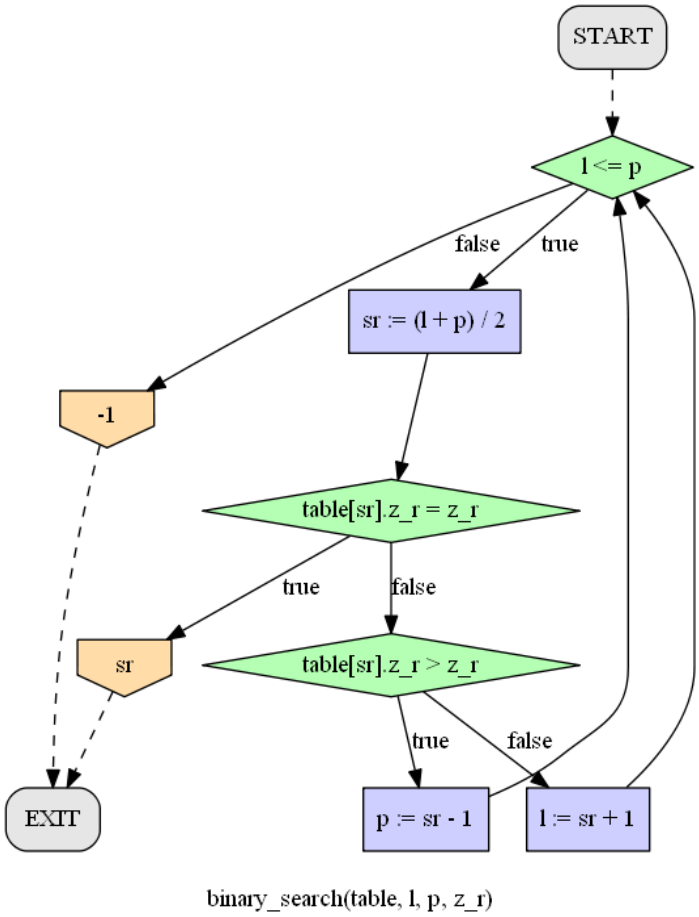
Przeszukiwanie tablicy metodą podziałów

```
int binary_search(powers_list_t* table, int l, int p, unsigned long long int z_r);
```

Funkcja sprawdza, czy podany element znajduje się w przekazanej tablicy. Tablica musi być posortowana, aby algorytm działał poprawnie. Na początku sprawdzany jest środkowy element tablicy. Jeśli jest taki sam jak szukany, algorytm się kończy. Jeśli środkowy element jest większy od szukanego, powtarzamy procedurę dla lewej połówki tablicy. Jeśli jest mniejszy – powtarzamy procedurę dla prawej połówki tablicy.

Jeśli uda się znaleźć szukany element, funkcja zwraca jego indeks w tablicy. Jeśli elementu nie ma, funkcja zwraca wartość -1.

Schemat blokowy algorytmu przeszukiwania tablicy metodą podziałów został przedstawiony na Ryc. 8.



Ryc. 8. Schemat blokowy algorytmu przeszukiwania tablicy metodą podziałów. *Źródło: Opracowanie własne.*

f) Porównanie wydajności najszybszego z zaimplementowanych rozwiązań z implementacją CUDA GPU zrealizowaną przez Jeet'a Chauhan'a

Porównano wydajność własnych rozwiązań z implementacją zaprezentowaną przez Jeet'a Chauhan'a [8]. Jak się okazało oprogramowanie Chauhan'a pomimo że wykorzystywało GPU rozwiązywało problem w sposób wysoce niezadowalający pod względem wydajnościowym. Nie realne w tym przypadku okazały się testy dla *max_base = 500* oraz *max_powers = 500* – zajęłyby relatywnie dużo czasu. Dlatego też w tym przypadku dla celów porównawczych zmniejszono badany przedział do *max_base = 100* oraz *max_powers = 100*. Wyniki zaprezentowano w Tab. 8.

	Procesor CPU	RAM	Procesor GPU	Kompilator	Czas realizacji
Jeet Chauhan Implementation	Intel i7 920 @ 2.80 GHz	24 GB DDR3	nVidia Tesla C2050	Cuda compilation tools, release 3.2, V0.2.1221	953,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K10	Cuda compilation tools, release 4.2, V0.2.1221	1116,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K10	Cuda compilation tools, release 5.0, V0.2.1221	1160,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K10	Cuda compilation tools, release 6.0, V6.0.1	1160,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K20	Cuda compilation tools, release 4.2, V0.2.1221	585,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K20	Cuda compilation tools, release 5.0, V0.2.1221	585,000 s
Jeet Chauhan Implementation	Intel Xeon E5-2670 @ 2.60GHz	512 GB DDR3	nVidia Tesla K20	Cuda compilation tools, release 6.0, V6.0.1	588,000 s
Łukasz Świerczewski Implementation	Intel i7 920 @ 2.80 GHz – 1 wątek	24 GB DDR3	Nie dotyczy	gcc version 4.1.2 20080704 (Red Hat 4.1.2-52)	1,790 s
Łukasz Świerczewski Implementation	Intel i7 920 @ 2.80 GHz – 8 wątków	24 GB DDR3	Nie dotyczy	gcc version 4.1.2 20080704 (Red Hat 4.1.2-52)	0,472 s
Łukasz Świerczewski Implementation	Intel Xeon E5-2670 @ 2.60GHz – 1 wątek	512 GB RAM	Nie dotyczy	gcc version 4.4.7 20120313 (Red Hat 4.4.7-11) (GCC)	1,100 s
Łukasz Świerczewski Implementation	Intel Xeon E5-2670 @ 2.60GHz – 16 wątków	512 GB RAM	Nie dotyczy	gcc version 4.4.7 20120313 (Red Hat 4.4.7-11) (GCC)	0,298 s

Tab. 8. Porównanie wydajności programu GPGPU Jeet'a Chauhan'a z oprogramowaniem własnym na kilku różnych platformach sprzętowych. *Źródło: Opracowanie własne.*

5. Zrównoleglenie kodu w OpenMP i wyniki czasowe realizacji równoległej algorytmów

Program można bardzo prosto zrównoleglić z wykorzystaniem środowiska OpenMP [5] [6]. Do trzeciej wersji kodu wystarczy dodać dwie pragmy – jedną odpowiedzialną za zrównoleglenie najbardziej zewnętrznej pętli for i drugiej odpowiedzialnej za wyodrębnienie sekcji krytycznej.

Pierwsza pragma powinna wyglądać następująco:

```
#pragma omp parallel for private(xi, x, yi, y, mi, x_m, ni, y_n, i, j, z) num_threads(number_of_threads)
```

Druga pragma to zwykłe ujęcie bloku instrukcji warunkowej:

```
if (z != x && z != y && gcd(z, x) == 1ULL && gcd(z, y) == 1ULL)
```

w

```
#pragma omp critical
```

Dodatkowo należało także w analogiczny sposób zrównoleglić pozostałe pętle wykorzystywane przy wstępnych obliczeniach (m.in. przy generowaniu tablicy *sorted_powers*).

a) Analiza wyników zrównoleglnie na platformach IBM Blue Gene/P, IBM Blue Gene/Q, IBM Power 775 oraz 2x Intel Xeon X5660

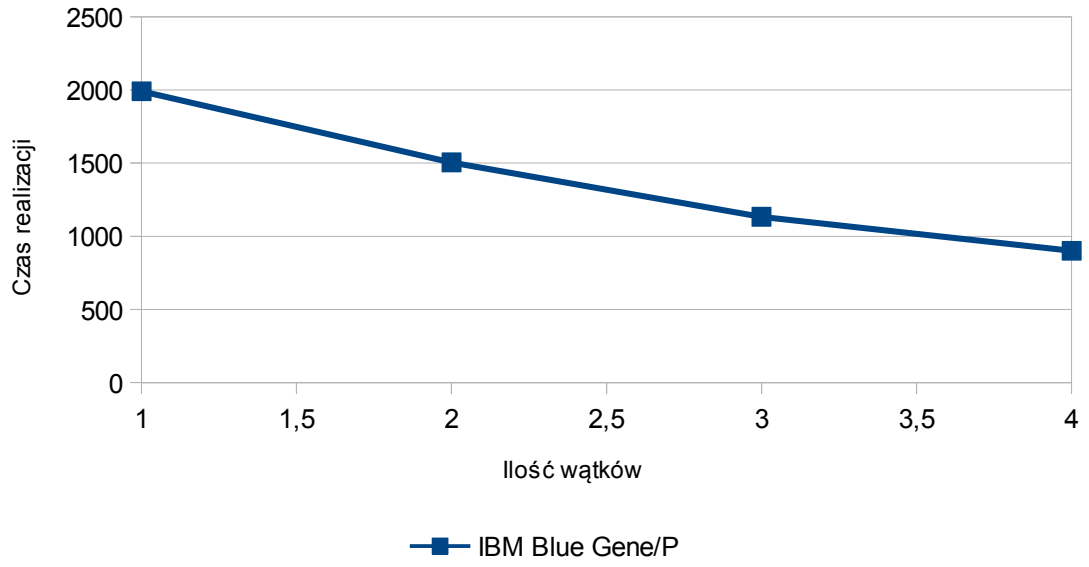
Wyniki czasowe zrównoleglenia uzyskane na czterech różnych platformach sprzętowych (IBM Blue Gene/P, IBM Blue Gene/Q, IBM Power 775 oraz 2x Intel Xeon X5660) przedstawiono w Tab. 6.

Ilość wątków	IBM Blue Gene/P	IBM Blue Gene/Q	IBM Power 775	2x Intel Xeon X5660
1	1991	1305	1734	1612
2	1505	996	1309	1229
3	1133	755	986	924
4	901	609	784	729
6	-	445	562	533
8	-	356	443	421
10	-	298	364	349
12	-	262	314	302
16	-	218	253	-
24	-	166	181	-
32	-	143	147	-

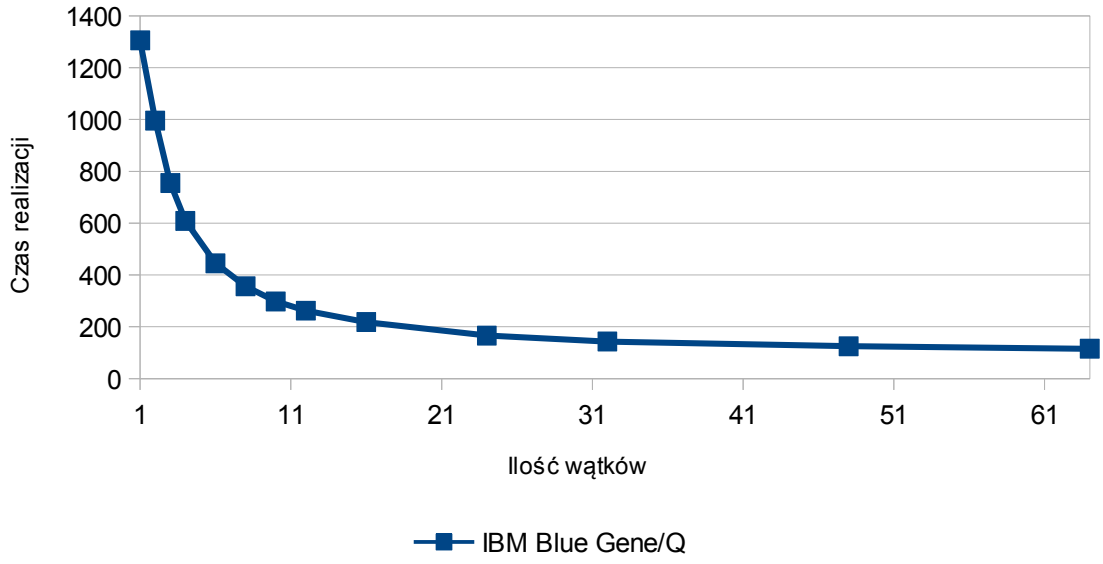
48	-	125	-	-
64	-	115	-	-

Tab. 9. Czasy realizacji rozwiązania zrównoleglonego w środowisku OpenMP wykonywanego z parametrami max_base = 500, max_power = 500. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*

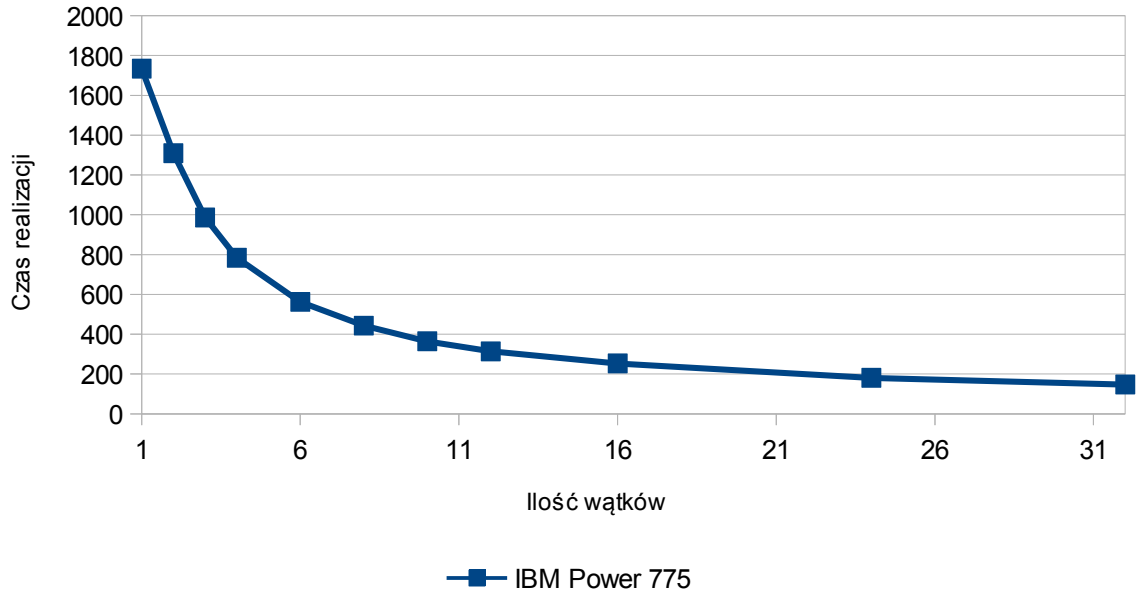
Wykresy przedstawiające spadek czasu realizacji programu dzięki zrównolegleniu dla badanych platform sprzętowych zaprezentowano na Ryc. 9 (IBM Blue Gene/P), Ryc. 10 (IBM Blue Gene/Q), Ryc. 11 (IBM Power 775) oraz Ryc. 12 (2x Intel Xeon X5660).



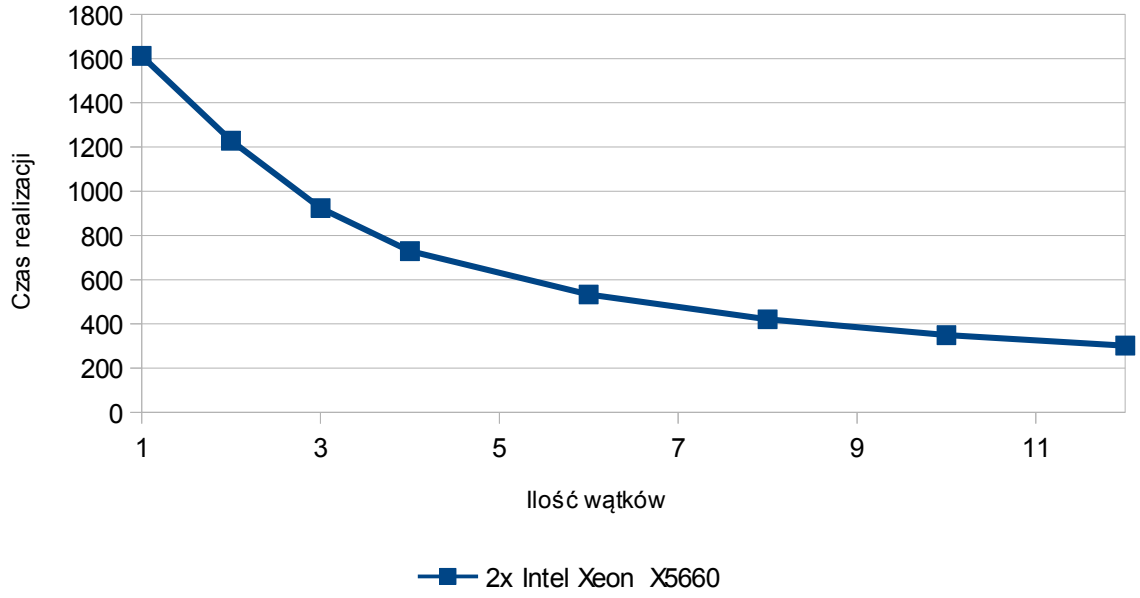
Ryc. 9. Redukcja czasu wykonywania programu na Blue Gene/P dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*



Ryc. 10. Redukcja czasu wykonywania programu na Blue Gene/Q dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*



Ryc. 11. Redukcja czasu wykonywania programu na IBM Power 775 dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*



Ryc. 12. Redukcja czasu wykonywania programu na 2x Intel Xeon X5660 dzięki zrównolegleniu w środowisku OpenMP. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*

Zestawienie maksymalnych przyśpieszeń jakie udało się uzyskać na badanych czterech platformach sprzętowych przedstawiono w Tab. 10.

	IBM Blue Gene/P	IBM Blue Gene/Q	IBM Power 775	2x Intel Xeon X5660
Maksymalne uzyskane przyśpieszenie:	2,209	11,347	11,795	5,337

Tab. 10. Uzyskane maksymalne przyśpieszenie dzięki zastosowaniu rozwiązania zrównoleglonego w środowisku OpenMP na różnych platformach sprzętowych. *Źródło: Opracowanie własne.*

Jak widać w Tab. 10 najwyższe przyśpieszenie uzyskano na węźle z procesorami IBM Power 775 i wynosiło ono 11,795.

b) Analiza wyników zrównoleglenia na platformie złożonej dodatkowo z akceleratora Intel Xeon Phi 5110P

Dość niedawno firma Intel wprowadziła na rynek nowe akceleratory Xeon Phi mające spełniać oczekiwania użytkowników wymagających bardzo dużych mocy obliczeniowych. Tego typu dwa akceleratory (5110P) zostały zainstalowane w klastrze Moss dostępnym w Poznańskim Centrum Superkomuterowo-Sieciowym. Do sprzętu tej klasy miałem dostęp w ramach narodowej inicjatywy PL-Grid. Specyfikację techniczną wybranych akceleratorów z rodziny Intel Xeon Phi przedstawiono w Tab. 11.

Typ akceleratora:	3120A	3120P	5110P	5120D	7120X	7120P
TDP (WAT):	300	300	225	245	300	300
Ilość rdzeni:	57	57	60	60	61	61
Częstotliwość zegara (GHz):	1,100	1,100	1,053	1,053	1,238	1,238
Wydajność teoretyczna (GFLOPS):	1003	1003	1011	1011	1208	1208
Wydajność pamięci (GT/s):	5,0	5,0	5,0	5,5	5,5	5,5
Przepustowość pamięci:	240	240	320	352	352	352
Rozmiar pamięci (GB):	6	6	8	8	16	16
Rozmiar pamięci cache (MB):	28,5	28,5	30,0	30,0	30,5	30,5
Tryb Turbo:	NIE	NIE	NIE	NIE	TAK	TAK

Częstotliwość trybu Turbo (GHz):	-	-	-	-	1,333	1,333

Tab. 11. Tabela przedstawiająca specyfikację techniczną wybranych akceleratorów z rodziny Intel Xeon Phi. *Źródło: Opracowanie własne na podstawie danych technicznych firmy Intel.*

Na klastrze Moss dostępne są dwa akceleratory Intel Xeon Phi. Na potrzeby tej pracy przetestowano jednak możliwości zrównoleglenia aplikacji jedynie na jednym z nich (mic0). Istnieje oczywiście możliwość wykorzystania dwóch z nich jednak należałoby w takim przypadku skorzystać ze środowiska MPI. Uzyskane czasy realizacji oraz przyśpieszenie dla trybu natywnego przedstawiono w Tab. 12. Analogiczną tabelą dla trybu offload (środowisko OpenMP) jest Tab. 13.

Ilość wątków	Czas realizacji	Przyśpieszenie
1	3721	-
2	2111	1,762
3	1408	2,642
4	1072	3,469
6	642	5,793
8	515	7,215
10	380	9,783
12	338	10,987
16	239	15,542
20	192	19,375
24	161	22,994
32	120	31,008
48	81	45,753
50	79	46,965
60	69	53,274

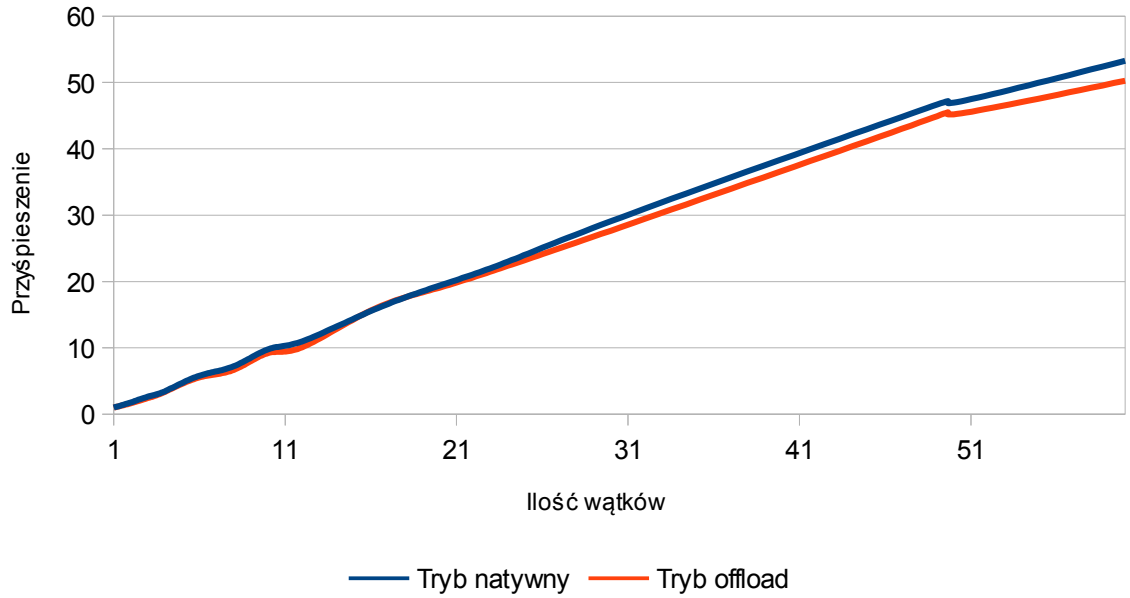
Tab. 12. Czasy realizacji oraz uzyskane przyśpieszenia rozwiązania zrównoleglonego w środowisku OpenMP (na akceleratorze Intel Xeon Phi 5110P w trybie natywnym) wykonywanego z parametrami max_base = 500, max_power = 500. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*

Ilość wątków	Czas realizacji	Przyśpieszenie
1	3729	-
2	2269	1,643
3	1530	2,436

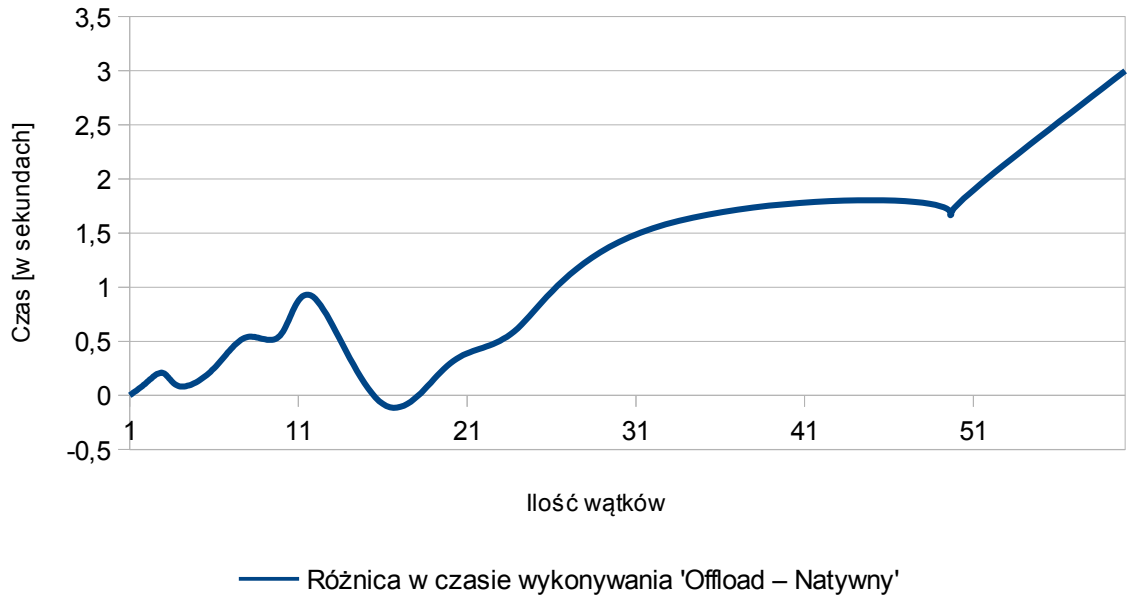
4	1101	3,386
6	672	5,543
8	558	6,675
10	404	9,214
12	369	10,094
16	238	15,621
20	195	19,078
24	166	22,374
32	126	29,465
48	84	43,973
50	82	45,212
60	74	50,277

Tab. 13. Czasy realizacji oraz uzyskane przyśpieszenia rozwiązania zrównoleglonego w środowisku OpenMP (na akceleratorze Intel Xeon Phi 5110P w trybie *offload*) wykonywanego z parametrami max_base = 500, max_power = 500. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*

Jak widać w powyższych tabelach minimalnie lepsze rezultaty (tj. wyższe przyśpieszenia) uzyskano zgodnie z oczekiwaniami w trybie natywnym. Na Ryc. 13 oraz Ryc. 14 można zobaczyć graficzne zobrazowanie na wykresie tych wyników.



Ryc. 13. Porównanie przyśpieszeń uzyskanych na Intel Xeon Phi w trybie *offload* oraz w trybie natywnym. Źródło: Opracowanie własne. *Źródło: Opracowanie własne.*



Ryc. 14. Przedstawienie graficzne różnicy w czasie wykonywania pomiędzy trybem *offload*, a natywnym. *Źródło: Opracowanie własne.*

Na Ryc. 14 dobrze widać, że praktycznie zawsze (nie licząc jednego przypadku dla 16 wątków) realizacja natywna jest szybsza od tej w OpenMP (*offload*). Jeżeli więc mamy bezproblemową możliwość skorzystania z tego trybu to warto jego wykorzystać, chociaż różnica z *offload* nie jest wielka. Należy także zauważyć, że różnica pomiędzy skrajnymi punktami tego wykresu jest bardzo mała (wynosi niecałe 3,5 sekundy). Dlatego też ponowne wykonanie pomiarów może dać dość zasadniczo inny przebieg.

c) Analiza wyników zrównoleglenia na platformie wykorzystującej Versatile SMP Foundation Advanced Platform firmy ScaleMP

Przetestowano możliwość zrównoleglenia oprogramowania z wykorzystaniem standardu OpenMP oraz platformy wykorzystującej Versatile SMP Foundation Advanced Platform firmy ScaleMP. Platforma ta pozwala agregować dziesiątki zwyczajnych serwerów w jedną dużą maszynę SMP w oparciu o szybką sieć InfniBand. Rozwiązanie tworzy komputer z współdzieloną pamięcią (dziesiątki TB pamięci operacyjnej) i setkami rdzeni obliczeniowych.

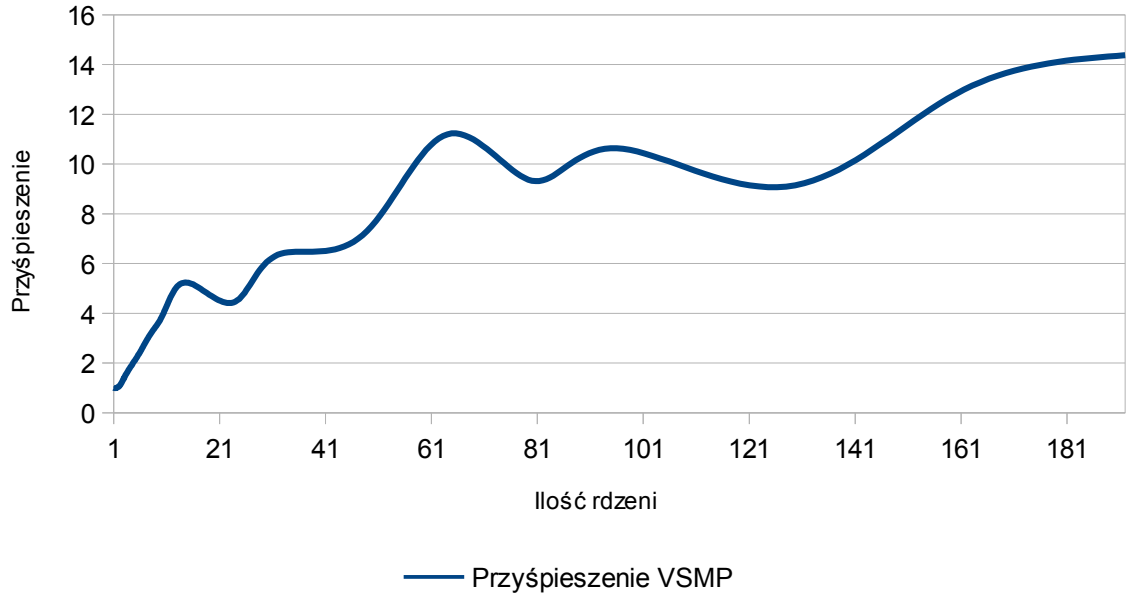
Specyfikacja dostępnego sprzętu:

Obudowy typu blade HP C7000 10U, serwery: typu blade HP ProLiant BL490 G6 (16 płyt głównych na obudowę blade), procesory Intel Xeon X5670 @ 2.93GHz, 12MB Cache, architektura EM64T, liczba procesorów: 32, Liczba rdzeni obliczeniowych: 192, Moc obliczeniowa: 2,25 TFlops (całość), całkowita pamięć operacyjna: 1152 GB (16 nodów po 72 GB), Sieć InfiniBand (40Gb/s) oraz Gigabit Ethernet (10Gb/s) .

Wyniki analizy wydajnościowej przedstawiono w Tab. 14 oraz Ryc. 15.

Ilość wątków	Czas realizacji	Przyśpieszenie
1	2128	1,000
2	2003	1,062
3	1502	1,416
4	1195	1,780
6	870	2,445
8	678	3,185
10	557	3,820
12	488	4,750
16	412	5,165
24	476	4,470
32	335	6,352
48	298	7,140
64	190	11,200
80	228	9,333
96	200	10,640
128	234	9,094
160	166	12,819
192	148	14,378

Tab. 14. Wyniki czasowe uzyskane podczas analizy przypuszczenia Beal'a zrównoleglonego w środowisku OpenMP wykonywanego z parametrami max_base = 500, max_power = 500. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*



Ryc. 15. Przyśpieszenia uzyskane na platformie wykorzystującej Versatile SMP Foundation Advanced Platform firmy ScaleMP. *Źródło: Opracowanie własne.*

Jak wynika z powyższej Tabeli oraz powyższego wykresu obliczenia na platformie vSMP wykorzystujące dużą ilość rdzeni są bardzo mało opłacalne. Wykorzystanie więcej niż 64 rdzeni nie daje już relatywnego przyśpieszenia. Jest to spowodowane najprawdopodobniej nie skalowaniem się w tym przypadku całej platformy i ograniczeniem przepustowości (lub też opóźnień) pamięci.

6. Adaptacja oprogramowania na potrzeby projektu rozproszonego

Aby uruchomić własny system rozproszony należało dostosować do jego potrzeb zarówno samą aplikację generującą wyniki, jak i sam *rdzeń* BOINC. Ten rozdział skrótowo opisuje przeprowadzone prace.

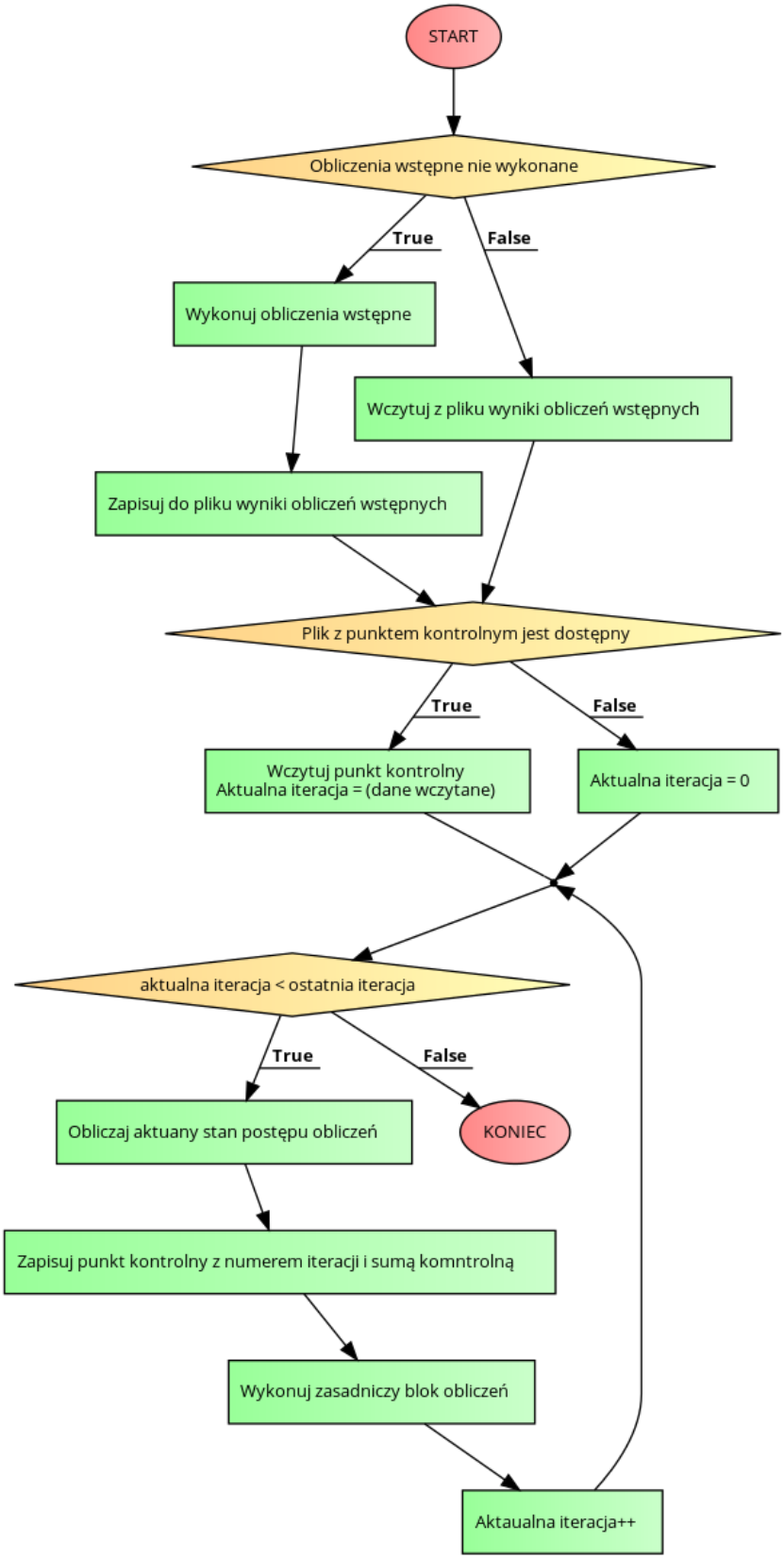
a) Dostosowanie do potrzeb silnika analizującego przypuszczenie Beal'a

Praktycznie każdą uruchamianą niezależnie przez klienta BOINC aplikację należy dostosować do jego wymogów. Wyjątkiem tutaj mogą być programy obliczeniowe uruchamiane za pomocą tak zwanego *wrappera* (pośrednika). W projekcie BealF@Home zastosowano jednak bezpośrednie wykorzystanie BOINC API z pominięciem funkcjonalności jakie daje wrapper, gdyż jest on całkowicie zbędny[5]. Na Listingu 6 przedstawiono uproszczony schemat zastosowanego rozwiązania. Na Ryc. 11. zaprezentowano natomiast analogiczny schemat blokowy.

```
START;

if(Obliczenia wstępne nie wykonane)
{
   Wykonuj obliczenia wstępne;
   Zapisuj do pliku wyniki obliczeń wstępnych;
}
else
{
   Wczytuj z pliku wyniki obliczeń wstępnych;
}

if(Plik z punktem kontrolnym jest dostępny)
{
   Wczytuj punkt kontrolny
   Aktualna iteracja = (dane wczytane);
}
else
{
   Aktualna iteracja = 0;
}

while(aktualna iteracja < ostatnia iteracja)
{
   Obliczaj aktuany stan postępu obliczeń;
   Zapisuj punkt kontrolny z numerem iteracji i sumą komntrolną;
   Wykonuj zasadniczy blok obliczeń;

   if(Znaleziono prawdopodobne rozwiązanie)
   {
     Zapisuj rozwiązanie do pliku;
   }

   Aktaualna iteracja++;
}

KONIEC;
```

Listing 6. Uproszczony pseudokod prezentujący możliwość adaptacji programu na potrzeby projektu BOINC. *Źródło: Opracowanie własne.*

5 Wrapper ma sens np. gdy aplikacja licząca jest napisana w innym języku niż C/C++ i nie ma możliwości z jego poziomu wywoływania funkcji BOINC API.



Ryc. 16. Uproszczony schemat blokowy prezentujący możliwość adaptacji programu na potrzeby projektu BOINC. *Źródło: Opracowanie własne.*

Na początku programu oczywiście należy załączyć (include) plik *"boinc_api.h"*. Następnie na na samym początku funkcji głównej *main* zostało zainicjowane środowisko BOINC API za pomocą funkcji *boinc_init()*. Jej obsługę zaprezentowano na Listingu 7.

```
1  int rc = boinc_init();
2  if (rc)
3  {
4      fprintf(stderr, "APP: boinc_init() failed. rc=%d\n", rc);
5      exit(rc);
6  }
7
8  boinc_fraction_done(0.0);
```

Listing 7. Obsługa funkcji *boinc_init()* oraz ustawienie postępu obliczeń na 0% za pomocą funkcji *boinc_fraction_done()*. *Źródło: Opracowanie własne.*

Kolejnym etapem, który warto opisać są kroki zaznaczone na Listingu 4 i Ryc. 11 jako *„Obliczaj aktualny stan postępu obliczeń”* oraz *„Zapisuj punkt kontrolny z numerem iteracji i sumą kontrolną”*. Są one realizowane już w funkcji głównej poszukującej kontrprzykładu dla przypuszczenia Beal'a. Operacje te przedstawiono na Listingu 8.

```
1  progress_value = (float) (xi - min_index) / (float) (max_index - min_index);
2
3  fp_progress=fopen("progress", "w");
4  fprintf (fp_progress, "%f", progress_value);
5  fclose(fp_progress);
6
7  boinc_fraction_done(progress_value);
8
9  fp_checkpoint=fopen("checkpoint", "w");
10 fprintf (fp_checkpoint, "%lld %lld", xi, checksum);
11 fclose(fp_checkpoint);
12
13 boinc_checkpoint_completed();
```

Listing 8. Kod realizujący obliczanie aktualnego stanu postępu obliczeń, aktualizowanie paska postępu oraz zapis do pliku punktu kontrolnego i sumy kontrolnej. *Źródło: Opracowanie własne.*

Na Listingu 6 wykorzystano dwie funkcje z BOINC API: wspomnianą już wcześniej *boinc_fraction_done()*, która przyjmuje za argument ułamek określający postęp obliczeń (0 → 0%, 0.5 → 50%, 1.0 → 100%) oraz *boinc_checkpoint_completed()*, która jak nazwa informuje wysyła informację do BOINC API, że punkt kontrolny został zapisany poprawnie.

Obliczanie stanu postępu obliczeń zostało zaprezentowane w pierwszej linijce Listingu 6. Do obliczania stanu wykorzystywane są zmienne: *xi*, *min_index* oraz *max_index*. Określają one odpowiednio numer aktualnej iteracji, numer iteracji startowej oraz numer iteracji końcowej. Dla przykładu przy wartościach: *xi = 620*, *min_index = 600*, *max_index = 800* wartość wyświetlona przy odpowiednim zadaniu na pasku postępu będzie wynosiła *(620-600) / (800-600) = 20 / 200 = 0.1 => 10%*. Zapisany punkt kontrolny oraz sumę kontrolną należy przy wznawianiu obliczeń odpowiednio wczytać z pliku. Kod odpowiedzialny za obsługę tego działania zaprezentowano na Listingu 9.

```
1   unsigned long long int checksum = 0;
2
3   if ( (fp_checkpoint=fopen("checkpoint", "r"))==NULL )
4   {
5       xi = min_index;
6   }
7   else
8   {
9       fscanf(fp_checkpoint, "%lld %lld", &xi, &checksum);
10      fclose(fp_checkpoint);
11  }
```

Listing 9. Kod w funkcji poszukującej kontrprzykładu dla przypuszczenia Beal'a odpowiedzialny za wznawianie obliczeń z zapisanego punktu kontrolnego (wczytaniu podlega także zapisana suma kontrolna). *Źródło: Opracowanie własne.*

W tym miejscu warto także napisać czym jest wcześniej wspomniana suma kontrolna. Suma kontrolna jest zawsze dopisywana na końcu pliku *„result”* z ewentualnymi rezultatami. Znaczna część zadań (ponad 99%) kończy się nie odnalezieniem w danym przedziale nawet rozwiązania prawidłowego modulo 2^{64}. Oznacza to, że gdyby nie było sumy kontrolnej zwracany byłby jedynie pusty plik. Nie byłoby więc możliwości zweryfikowania przez serwer, czy faktycznie dany komputer klienta wykonał wszystkie wymagane obliczenia. Można więc wyobrazić sobie sytuację, gdy użytkownik zatrzymuje aplikację liczącą, podmienia jedynie wartość w pliku gdzie zapisany jest ostatni punkt kontrolny na wyższą (bliżą końcu obliczeń) i wznawia obliczenia ale już od nowego, dalszego punktu kontrolnego. W ten sposób teoretycznie możliwe byłoby oszukanie systemu, co jednak zostało wyeliminowane przez zastosowanie sumy kontrolnej.

Suma kontrolna na samym starcie aplikacji wynosi 0 (Linia pierwsza w Listingu 7). Z czasem działania funkcji głównej wyszukującej kontrprzykłady jest zwiększana o wartość bezwzględną pewnej liczby zwróconej na danym kroku iteracji przez *binary_search()*. Jest to procedura całkowicie deterministyczna i na różnych komputerach, w różnym czasie obliczeń będzie dawała ten sam wynik końcowy (sumę kontrolną). Kod prezentujący obliczenie końcowe sumy kontrolnej oraz operację jej zapisu do pliku *„result”* przedstawiono na Listingu 10.

```
1   checksum = 1000 + checksum % 1000;
2
3   fp_results=fopen("result", "a");
4   fprintf (fp_results, "CHECKSUM=%ld", checksum);
5   fclose(fp_results);
```

Listing 10. Kod prezentujący obliczenie końcowe sumy kontrolnej oraz operację jej zapisu do pliku *„result”. Źródło: Opracowanie własne.*

Jak widać na Listingu 8 suma kontrolna podlega ostatecznie pewnemu „okrojeniu”. Dzięki operacji w linii pierwszej będzie miała ona jednak zawsze cztery cyfry (od 1000 do 1999). Jest to istotne z pewnych względów i aspekt ten został wykorzystany podczas programowania walidatora po stronie serwera projektu BOINC.

Na samym końcu programu można wywołać ostatecznie *boinc_fraction_done(1.0)* oraz funkcję *boinc_finish(0)*, która poinformuje BOINC API o zakończeniu obliczeń.

b) Dostosowanie do potrzeb platformy BOINC

Pierwszą rzeczą jaką powinniśmy zrobić na zainstalowanym już serwerze BOINC jest dodanie aplikacji. Pierwszą rzeczą jaką powinniśmy wykonać jest dodanie nazwy naszej aplikacji do pliku *project.xml* znajdującego się w katalogu głównym projektu. Wpis ten będzie wyglądać następująco:

<app>

<name>beal_engine</name>

<user_friendly_name>Beal Engine</user_friendly_name>

</app>

Kolejnym etapem jest stworzenie odpowiedniej struktury katalogów (od *./apps*) oraz umieszczenie w nim odpowiednich plików (w tym oczywiście także aplikacji liczącej). Przykładowo dla wersji aplikacji 1.00 i platformy Linux x86_64 (*x86_64-pc-linux-gnu*) będzie to:

./apps/beal_engine/1.00/x86_64-pc-linux-gnu/

Dla aplikacji *beal_engine* w wersji 1.01 i platformy Microsoft Windows x86_64 będzie to:

./apps/beal_engine/1.01/windows_x86_64/

Więcej etykiet określających różne platformy odnajdziemy w pliku *project.xml*. Należy zaznaczyć, że możemy zdefiniować aplikacje dla różnych, bardzo egzotycznych platform takich jak konsola do gier PlayStation 3 lub system Android działający na procesorach ARM.

W powyżej zdefiniowanych katalogach umieszczamy aplikację liczącą. W naszym przypadku dla wersji 1.00 i Linux x86_64 (*x86_64-pc-linux-gnu*) jej nazwa będzie wyglądać następująco:

beal_engine_1.00_x86_64-pc-linux-gnu

Zasadniczo można powiedzieć, że szablon nazwy aplikacji można zdefiniować jako:

nazwa_aplikacji_number_wersji_platforma

Dodatkowo w tym samym katalogu co aplikacja musimy umieścić także dwa pliki: *job.xml* oraz *version.xml*. Zawartość pliku *job.xml* przedstawiono na Listingu 11. Na Listingu 12 przedstawiono natomiast zawartość pliku *version.xml*.

```
1  <job_desc>
2      <task>
3          <application>beal_enigne</application>
4      </task>
5  </job_desc>
```

Listing 11. Zawartość pliku *job.xml*. *Źródło: Opracowanie własne.*

```xml
<version>
   <file>
      <physical_name>beal_engine_1.00_x86_64-pc-linux-gnu</physical_name>
      <needs_network/>
      <copy_file/>
      <main_program/>
    </file>
    <file>
      <physical_name>job.xml</physical_name>
      <logical_name>job.xml</logical_name>
   </file>
</version>
```

Listing 12. Zawartość pliku *version.xml*. *Źródło: Opracowanie własne.*

Kolejnym krokiem jaki powinniśmy wykonać jest stworzenie odpowiednich szablonów i zapisanie ich w katalogu *./templates/*. Ja szablon wejściowy nazwałem *input.xml* (*./templates/input.xml*), a szablon wyjściowy *output.xml* (*./templates/output.xml*). Przykładowy szablon wejściowy przedstawiono na Listingu 13, a przykładowy szablon wyjściowy na Listingu 11.

```xml
<file_info>
        <number>0</number>
</file_info>
<workunit>
        <file_ref>
                <file_number>0</file_number>
                <open_name>input</open_name>
                <copy_file/>
        </file_ref>

        <target_nresults>2</target_nresults>
        <min_quorum>2</min_quorum>
        <rsc_fpops_bound>99999999999999999</rsc_fpops_bound>
        <rsc_fpops_est>575705506285715</rsc_fpops_est>
        <rsc_memory_bound>10000000.000000</rsc_memory_bound>
        <delay_bound>604800</delay_bound>
</workunit>

```

Listing 13. Przykładowy szablon wejściowy. *Źródło: Opracowanie własne.*

Najbardziej rozwinięty jest w szablonie wejściowym tag *<workunit>*. To w nim są zawarte informacje niezbędne do wygenerowania nowego zadania na platformie BOINC. W tagu *<file_ref>* znajdują się informacje o pliku lub też plikach wejściowych, z których korzysta nasza aplikacja licząca. Tag *<delay_bound>* określa czas *deadline* – czas (wyrażony w sekundach) w jakim klient musi wykonać wszystkie obliczenia i odesłać je do serwera. Jeżeli w tym okresie serwer nie otrzyma odpowiedzi to przydzieli wykonanie określonej porcji obliczeń innemu klientowi. *<rsc_memory_bound>* określa górny limit pamięci (wyrażony w Bajtach), który może być wykorzystany przez aplikację liczącą. *<rsc_fpops_est>* to przeciętna ilość operacji

zmiennoprzecinkowych, która jest wymagana do zakończenia zadania, a *<rsc_fpops_bound>* maksymalna, graniczna ilość operacji zmiennoprzecinkowych po przekroczeniu, które zadanie zostanie zabite.

```
<output_template>
    <file_info>
        <name><OUTFILE_0/></name>
        <generated_locally/>
        <upload_when_present/>
        <max_nbytes>32768</max_nbytes>
        <url><UPLOAD_URL/></url>
    </file_info>
    <result>
        <file_ref>
            <file_name><OUTFILE_0/></file_name>
            <open_name>result</open_name>
            <copy_file>1</copy_file>
            <no_delete/>
        </file_ref>
    </result>
</output_template>
```

Listing 14. Przykładowy szablon wyjściowy. *Źródło: Opracowanie własne.*

Szablon wyjściowy może się wydawać trochę prostszy od wejściowego. Najistotniejsza informacja jest zawarta w tagu *<result>* i *<file_ref>* - jest to nazwa pliku wynikowego wygenerowanego przez aplikację liczącą – w tym przypadku *result*. Oczywiście plików wynikowych może być wiele. W przypadku plików, które mogą zajmować dużo miejsca należy także pamiętać o prawidłowym ustaleniu wartości w tagu *<max_nbytes>*.

Po stworzeniu odpowiedniej struktury katalogów w *./apps/* oraz umieszczeniu w niej niezbędnych plików i utworzeniu odpowiednich szablonów można przejść do właściwego dodawania aplikacji do serwera BOINC.

Najpierw wydajemy polecenie:

./bin/xadd

Dzięki temu zostanie dodana wstępnie aplikacja. Teraz musimy zaktualizować jej wersję wydając polecenie:

./bin/update_versions

Wynik wywołania tego polecenia przedstawiono na Listing. 15.

```
boincadm@beal:~/projects/bealf$ ./bin/update_versions

Found app version directory for: beal_engine 1.00 x86_64-pc-linux-gnu

    NOTICE: You have not provided a signature file for job_2.0.xml,
    and your project's code-signing private key is on your server.

    IF YOUR PROJECT IS PUBLICLY ACCESSABLE, THIS IS A SECURITY VULNERABILITY.
    PLEASE STOP YOUR PROJECT IMMEDIATELY AND READ:
    http://boinc.berkeley.edu/trac/wiki/CodeSigning

    Continue (y/n)? y
cp apps/beal_engine/1.00/x86_64-pc-linux-gnu/job_2.0.xml \
/home/boincadm/projects/bealf/download/job_2.0.xml
cp apps/beal_engine/1.00/x86_64-pc-linux-gnu/beal_engine_1.00_x86_64-pc-linux-gnu \
/home/boincadm/projects/bealf/download/beal_engine_1.00_x86_64-pc-linux-gnu
    Files:
        beal_engine_1.00_x86_64-pc-linux-gnu (main program)
        job_2.0.xml
    Flags:
        API version: 7.1.0
    Do you want to add this app version (y/n)? y
    App version added successfully; ID=1
```

Listing. 15. Wywołanie polecenia *./bin/update_versions* podczas pierwszej aktualizacji wersji aplikacji. *Źródło: Opracowanie własne.*

Ostatnim już etapem będzie wygenerowanie WU (ang. *workunit*), które będą mogli pobrać klienci i rozpocząć obliczenia.

Możemy zrobić to wprowadzając:

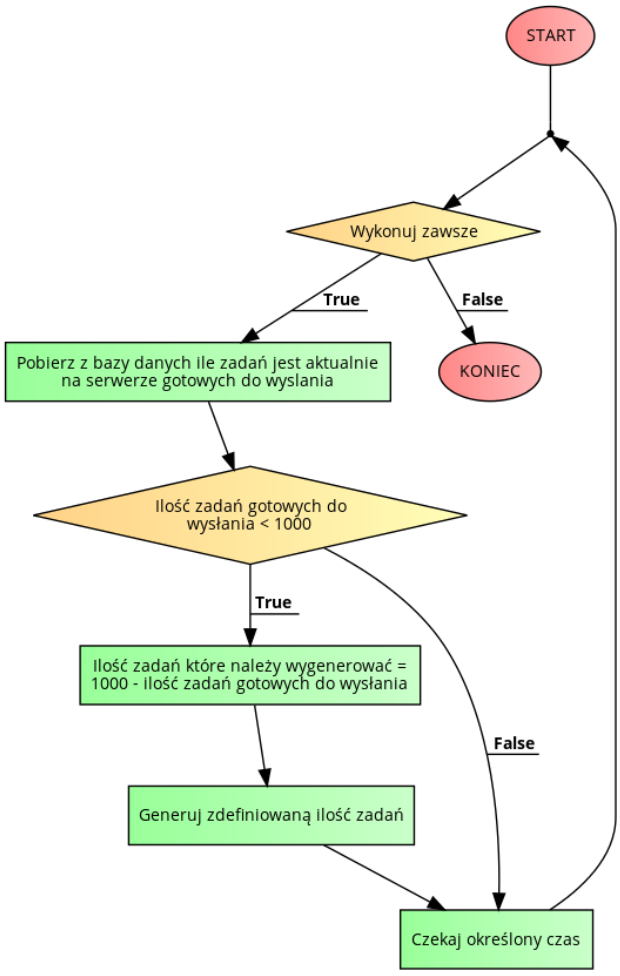
./bin/create_work -appname beal_engine -wu_name NAZWA-WU -wu_template templates/input.xml -result_template templates/output.xml plik_wejściowy

gdzie *plik_wejściowy* jest nazwą pliku, który znajduje się w katalogu *./download/*. Po wykonaniu tej operacji możemy połączyć się BOINC Manager'em (lub samym BOINC klientem) z naszym projektem i pobrać pierwsze zadanie obliczeniowe.

Jako walidator zadań na serwerze BOINC zastosowano zmodyfikowany *sample_bitwise_validator*. Jedno takie samo zadanie jest zawsze wysyłane co najmniej do dwóch klientów (min_quorum = 2). Odsyłają oni wynik obliczeń, który powinien być taki sam. *Sample_bitwise_validator* porównuje ze sobą nadesłane wyniki. Jeżeli są identyczne to zatwierdza zadanie jako zrealizowane. Jeżeli natomiast jest pomiędzy nimi jakakolwiek różnica, to wysyła to samo zadanie do trzeciego klienta i na podstawie wyniku, który on zwróci waliduje ostatecznie daną porcję obliczeń.

Główną modyfikacją walidatora jest dodanie do jego możliwości wyłuskiwania z plików wynikowych potencjalnych rozwiązań. W prosty sposób wykorzystano fakt, że waga pliku, w którym nie ma żadnego rozwiązania wynosić będzie zawsze 13 Bajtów (jest to jedynie zapis sumy kontrolnej, która ma postać: *CHECKSUM=XXXX*). W przypadku gdy rozmiar ten jest większy to znaczy, że w pliku znajdują się potencjalne rozwiązania przypuszczenia Beal'a i zostają one umieszczone w dedykowanej do tego celu bazie danych.

Jednym z kluczowych zadań na platformie BOINC jest także odpowiednie generowanie masowych zadań (jak wygenerować jedno zadanie zaprezentowano wcześniej). Można oczywiście funkcjonalność tą realizować w sposób trywialny – np. za pomocą specjalnego skryptu Bash'a wywoływać BOINC'owy program *create_work* i w ten sposób generować kolejne próbki zadań. Można także opracować bardziej wyrafinowane metody. W projekcie BealF@Home początkowo wykorzystywano właśnie prosty skrypt Bash (dodany do pracy jako *załącznik D*). Na późniejszym etapie zastosowano bardziej zaawansowany generator napisany w języku C, który wywoływał polecenia systemowe i łączył się z bazą danych projektu (dodany do pracy jako *załącznik E*). Uproszczony blokowy tego generatora zaprezentowano na Ryc. 17.



Ryc. 17. Uproszczony schemat blokowy bardziej zaawansowanego generatora zadań wykorzystanego na platformie BealF@Home. *Źródło: Opracowanie własne.*

Jak widać powyższy generator zadań jest wykonywany w nieskończonej pętli (może zostać przerwany tylko przez użytkownika). Przy każdej iteracji (które dzielą odpowiedni okres czasu) jest sprawdzana ilość zadań na serwerze, i jeżeli jest ona mniejsza niż 1000 to automatycznie generowane są nowe próbki. Dzięki takiemu rozwiązaniu zadania są generowane zawsze na bieżąco, bez chwilowego, znacznego obciążenia serwera jak to było w przypadku prostego rozwiązania napisanego w Bash'u.

7. Podsumowanie

Podczas obliczeń ograniczono się jedynie do klasycznych procesorów CPU. Można jednak dodatkowo wykorzystać akceleratory graficzne co byłoby bardzo ciekawym podejściem do problemu. Dzięki takiemu rozwiązaniu najprawdopodobniej możliwe byłoby uzyskanie znacznie lepszych wyników końcowych. Oczywiście istnieją już zaprogramowane implementacje w CUDA [7] – należy do nich m.in. rozwiązanie zaprezentowane w pracy magisterskiej Jeet'a Chauhan'a [8] napisanej w San Diego State University. Jak jednak wynika z porównań tej implementacji z moim kodem, system opracowany przez Chauhan'a znacznie odbiega od ideału – pomimo że wykorzystuje dodatkowy akcelerator graficzny to zadania realizuje ponad 500 razy wolniej.

Istnieje także implementacja testująca zagadnienie przypuszczenia Beal'a w języku Brook+ [9], której jednak ze względu na brak odpowiedniego środowiska już nie przetestowano (najprawdopodobniej jednak wydajnościowo wyglądałaby podobnie do rozwiązania Jeet'a Chauhan'a). Pomysły te mają jednak swoje ograniczenia i nie nadają się do zastosowania na wielką skalę.

Połączeniem możliwości jakie daje środowisko MPI [10] z wykorzystaniem akceleratorów graficznych zajął się także Naveen Kumar Reddy Nandipati [11]. W tym miejscu należy także wspomnieć o standardzie OpenCL [12], który jest jednolitym środowiskiem umożliwiającym programowanie kart graficznych (zarówno AMD jak i nVidia) oraz np. procesorów IBM Cell [13].

Do obliczeń wykorzystano także wykorzystać platformę BOINC [14] [15] (ang. *Berkeley Open Infrastructure for Network Computing*), która umożliwia połączenie mocy obliczeniowych tysięcy komputerów połączonych za pomocą sieci Internet. Inne aspekty badawcze, które można wykorzystać w dalszej pracy zaprezentowali polscy badacze [16] [17].

Opracowana na potrzeby tej pracy metoda testowania przypuszczenia Beal'a nie jest oczywiście doskonała. Analizy wykonane dla określonych przedziałów mają tą wadę, że nie znajdą rozwiązania, którego zmienne x, y, z, n, m, r leżą w różnych z nich. Dla przykładu weźmy analizowane dwa analizowane przedziały:

Pierwszy z nich dla podstaw: [2000, 4000]; Dla wykładników: [3, 1000].
Drugi z nich dla podstaw: [4000, 6000]; Dla wykładników: [3, 1000].

Algorytm nie odnajdzie rozwiązania jeżeli takowe znajduje się w sumie zbioru podstaw: [2000, 6000]. Niedogodność tą można rozwiązać przynajmniej na kilka sposobów, kosztem jednak znacznego zwiększenia zapotrzebowania na moc obliczeniową. Rozwiązanie tego problemu będzie jednak zagadnieniem początkowym dla przyszłych badań.

Ostatecznie w ramach tej pracy nie udało się odnaleźć kontrprzykładu dla przypuszczenia Beal'a. Nie oznacza to oczywiście jak wcześniej zaznaczono, że takie rozwiązanie nie istnieje. Rezultatem przeprowadzonych obliczeń są rozwiązania prawidłowe modulo 2^{64}. Takich przypadków do dnia 07.06.2015 r. odnaleziono w sumie 47 (zostały przedstawione w załącznikach H i J).

Literatura

1. Mauldin, R. Daniel. "A generalization of Fermat's Last Theorem: the Beal conjecture and prize problem." *Notices of the American Mathematical Society* 44.11 (1997): 1436-1437.

2. Beal Prize - American Mathematical Society, URL: http://www.ams.org/profession/prizes-awards/ams-supported/beal-prize, Ostatni dostęp: 18.04.2014

3. Beal Conjecture, URL: http://www.bealconjecture.com/, Ostatni dostęp: 18.04.2014

4. Beal's Conjecture: A Search for Counterexamples, URL: http://norvig.com/beal.html, Ostatni dostęp: 18.04.2014

5. Dagum, Leonardo, and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming." *Computational Science & Engineering, IEEE* 5.1 (1998): 46-55.

6. Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*. Vol. 10. MIT press, 2008.

7. Sanders, Jason, and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

8. Chauhan, Jeet. *Nvidia GeForce 8400 GPGPU implementation of a CUDA C parallel algorithm to search for counterexamples to Beal's Conjecture*. Diss. San Diego State University, 2010.

9. Shah, Nirav. *AMID Firestream 9170 GPGPU implementation of a Brook+ parallel algorithm to search for counterexamples to Beal's Conjecture*. Diss. San Diego State University, 2010.

10. Gropp, William, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.

11. Nandipati, Naveen Kumar Reddy. "Heterogeneous NPACI-ROCKS/MPI/CUDA distributed multi-GPGPU application for seeking counterexamples to Beal's Conjecture; ROCKS/CUDA integration component." (2011).

12. Stone, John E., David Gohara, and Guochun Shi. "OpenCL: A parallel programming standard for heterogeneous computing systems." *Computing in science & engineering* 12.3 (2010): 66.

13. Chen, Thomas, et al. "Cell broadband engine architecture and its first implementation—a performance view." *IBM Journal of Research and Development* 51.5 (2007): 559-572.

14. Anderson, David P. "Boinc: A system for public-resource computing and storage." *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE, 2004.

15. Estrada, Trilce, Michela Taufer, and David P. Anderson. "Performance prediction and analysis of BOINC projects: An empirical study with EmBOINC." *Journal of Grid*

Computing 7.4 (2009): 537-554

16. H. Zarzycki H, J. Czerniak, *Development and Code Management of Large Software Systems*, PSZW, nr 27, s. 327-339, Bydgoszcz 2010.

17. H. Zarzycki, Application of the finite difference CN method to value derivatives, PSZW, nr 42, s. 267-277, Bydgoszcz 2011.

18. Berendsen, Herman JC, David van der Spoel, and Rudi van Drunen. "GROMACS: A message-passing parallel molecular dynamics implementation." *Computer Physics Communications* 91.1 (1995): 43-56.

Spis rysunków

Spis tablic

Tab. 14. Wyniki czasowe uzyskane podczas analizy przypuszczenia Beal'a zrównoleglonego w środowisku OpenMP wykonywanego z parametrami max_base = 500, max_power = 500. Czasy mierzone w sekundach. *Źródło: Opracowanie własne.*

Spis listingów

Załączniki

a) Schemat postępowania podczas instalacji serwera BOINC

1. Instalujemy na naszym serwerze system operacyjny Debian (w naszym przypadku będzie to wersja 7.7.0 64-bit).

2. Dodajemy w systemie użytkownika „boincadm”.

   adduser boincadm

3. Instalujemy niezbędne do działania serwera BOINC komponenty (pakiety).

   apt-get install m4 make dh-autoreconf pkg-config git libapache2-mod-php5 mysql-server libmysqlclient-dev php5-mysql php5-cli php5-gd phpmyadmin python python-mysqldb libssl-dev libcurl4-openssl-dev

   Instalator poprosi nas o zdefiniowanie hasła root'a oraz wybranie rodzaju serwera, z którym zostanie automatycznie skonfigurowany PHPMyAdmin (w naszym przypadku zaznaczamy apache2).

   Końcówka logów powinna wyglądać mniej więcej następująco:

   [....] Reloading web server config: apache2apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName
   . ok
   Setting up g++-4.7 (4.7.2-5) ...
   Setting up g++ (4:4.7.2-1) ...
   update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
   Setting up build-essential (11.5) ...
   Setting up libstdc++6-4.7-dev (4.7.2-5) ...
   Processing triggers for libapache2-mod-php5 ...
   [....] Reloading web server config: apache2apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName
   . ok

   Jeżeli proces instalacji zwrócił jakieś błędy oznacza to, że najprawdopodobniej są jakieś problemy z repozytorium. Rozwiązaniem problemu może być wykonanie polecenia: **apt-get update**.

   Po tym kroku wymagane do działania serwera BOINC oprogramowanie powinno już zostać poprawnie zainstalowane i funkcjonować. Serwer Apache możemy przetestować wchodząc na stronę http://IPSERWERA lub wpisując w przeglądarce bezpośrednio localhost (jeżeli mamy do serwera dostęp z poziomu X'ów oraz jest zainstalowana na nim przeglądarka internetowa).

4. Dodajemy naszego użytkownika *boincadm* oraz *www-data* do odpowiednich grup.

   usermod -a -G www-data boincadm
   usermod -a -G boincadm www-data

5. Następnym krokiem będzie dodanie odpowiedniego użytkownika w naszym systemie bazodanowym MySQL.

   Wpisujemy:

   mysql -p

   System poprosi nas o podanie hasła do serwera MySQL (użytkownika root), które zdefiniowaliśmy w kroku trzecim tej instrukcji.

   Wprowadzamy:

   mysql> *CREATE USER 'boincadm'@'localhost' IDENTIFIED BY 'nasze_haslo';*

   System zwróci informację:

   Query OK, 0 rows affected (0.00 sec)

   Wychodzimy z MySQL wpisując:

   mysql> *quit*

6. W tym momencie powinniśmy przelogować się na użytkownika *boincadm* – wstępną konfigurację systemu za pomocą konta root'a mamy już za sobą.

7. Bezpośrednio w katalogu domowym użytkownika *boincadm* klonujemy kod źródłowy serwera BOINC.

   git clone git://boinc.berkeley.edu/boinc-v2.git nazwa-serwera

   Operacja ta może trochę potrwać, ponieważ system będzie pobierać wszystkie wymagane źródła z serwera Uniwersytetu Kalifornijskiego w Berkeley.

   W wyniku otrzymamy informację podobną do poniższej:

   Cloning into 'nazwa-serwera'...
   remote: Counting objects: 229790, done.
   remote: Compressing objects: 100% (44125/44125), done.
   remote: Total 229790 (delta 186678), reused 225238 (delta 182785)
   Receiving objects: 100% (229790/229790), 147.98 MiB | 2.10 MiB/s, done.
   Resolving deltas: 100% (186678/186678), done.
   Checking out files: 100% (3684/3684), done.

8. Kolejnym etapem jest kompilacja źródeł BOINC oraz instalacja.

   Wchodzimy do katalogu z naszymi źródłami:

   cd nazwa-serwera

   Wykonujemy skrypt autosetup:

   ./_autosetup

   W przypadku braku problemów powinien zwrócić on:

   Done, now run ./configure

   Postępujemy według wskazówki i uruchamiamy skrypt configure z tym, że dodajemy dodatkowe flagi: --disable-client --disable-manager dzięki, którym nie będziemy kompilować klienta oraz manager'a BOINC.

   ./configure --disable-client --disable-manager

   Proces ten na wirtualnej maszynie z procesorem równoważnym 1 GHz i 1 GB RAM zajął około 2 minut i zakończył się zwróceniem przez system informacji:

   --- Configuring BOINC 7.5.1 (Release) ---
   --- Build Components: (libraries server) ---

   Teraz należy wykonać:

   make

   oraz:

   make install

   Na tym etapie m.in. wszystkie wymagane biblioteki będące integralnymi składnikami serwera BOINC są zainstalowane w systemie.

9. Teraz możemy utworzyć bazę danych w której będą składowane informacje projektu.

   Wchodzimy do MySQL z poziomu konta root:

   mysql -u root -p

   Podajemy wymagane hasło oraz wprowadzamy następujące zapytanie:

   GRANT CREATE, DROP, SELECT, INSERT, UPDATE, DELETE, CREATE, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES ON `nazwa-serwera`. TO 'boincadm'@'localhost';*

10. Przechodzimy do właściwego utworzenia projektu za pomocą skryptu make_project:

    ./tools/make_project --delete_prev_inst --db_name kproject --db_user boincadm --db_passwd nasze_haslo --db_host localhost kproject 'Project@Home'

    Opcja *--delete_prev_inst* jest fakultatywna i służy do usunięcia poprzednich plików serwera (np. gdy coś nam nie wyszło i instalujemy serwer od nowa).

    W przypadku tworzonego projektu BealF@Home dostaniemy następującą informację:

    Creating project 'BealF@Home' (short name 'bealf'):
    PROJECT_ROOT = /home/boincadm/projects/bealf/
    URL_BASE = http://debian/
    HTML_USER_URL = http://debian/bealf/
    HTML_OPS_URL = http://debian/bealf_ops/
    KEY_DIR = /home/boincadm/projects/bealf/keys/
    DB_NAME = beal
    DB_HOST = localhost

    Continue? [Y/n]

    Potwierdzamy naciskając ***Y***.

    Jeżeli wszystko poszło po planie system powinien zwrócić informację:

    Done creating project. Please view
    /home/boincadm/projects/bealf/bealf.readme
    for important additional instructions.

11. Następnym krokiem powinno być wejście do pliku *nazwa-serwera.readme* (w naszym przypadku *bealf.readme*) i zapoznanie się z dalszymi instrukcjami –najistotniejszych z nich realizacja została przedstawiona w punktach 12 -17.

12. Nadpisujemy konfigurację serwera Apache, która od teraz ma wskazywać na właściwy katalog użytkownika boincadm.

    Standardowo jest to:

    cat /home/boincadm/projects/nazwa-serwera/nazwa-serwera.httpd.conf >> /etc/apache2/apache2.conf

    W naszym przypadku będzie to:

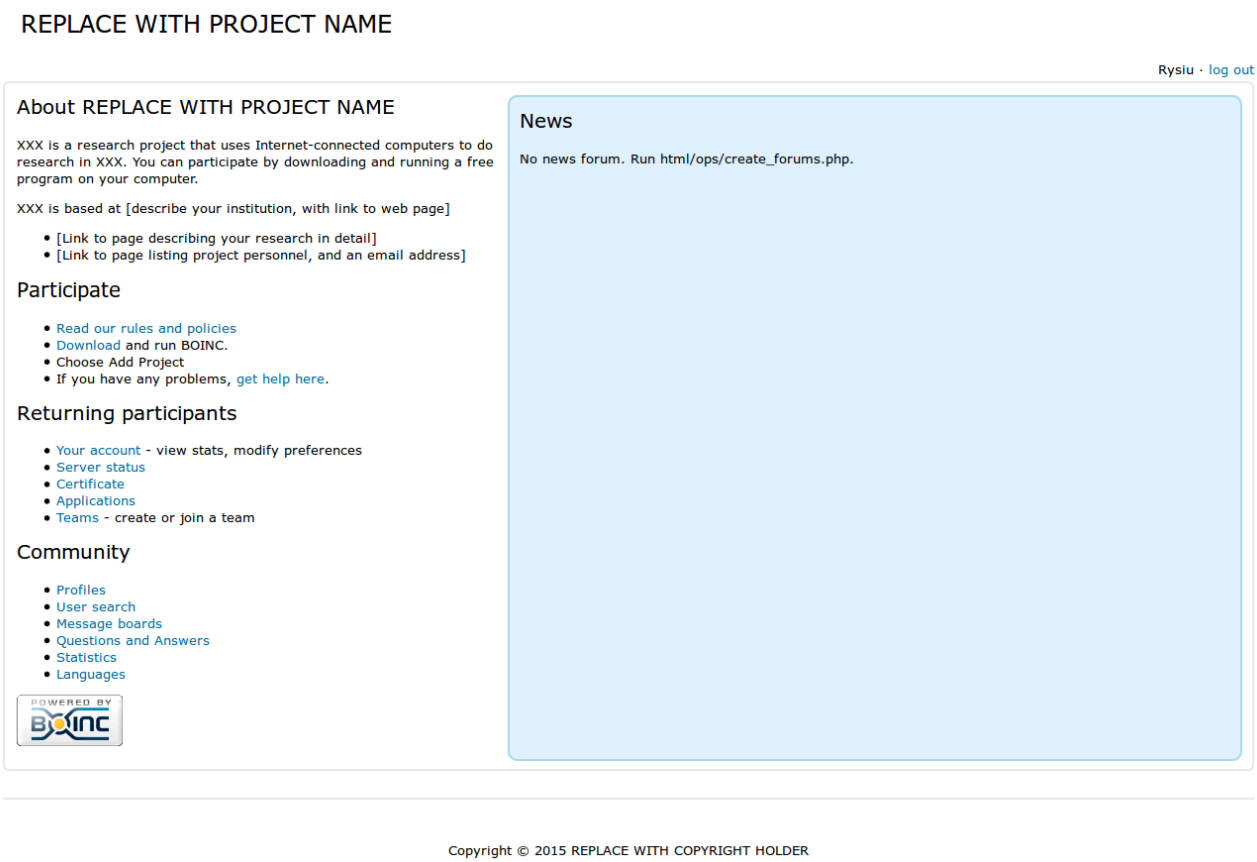
    cat /home/boincadm/projects/bealf/bealf.httpd.conf >> /etc/apache2/apache2.conf

13. Restartujemy serwer Apache (jako użytkownik root).

    service apache2 restart

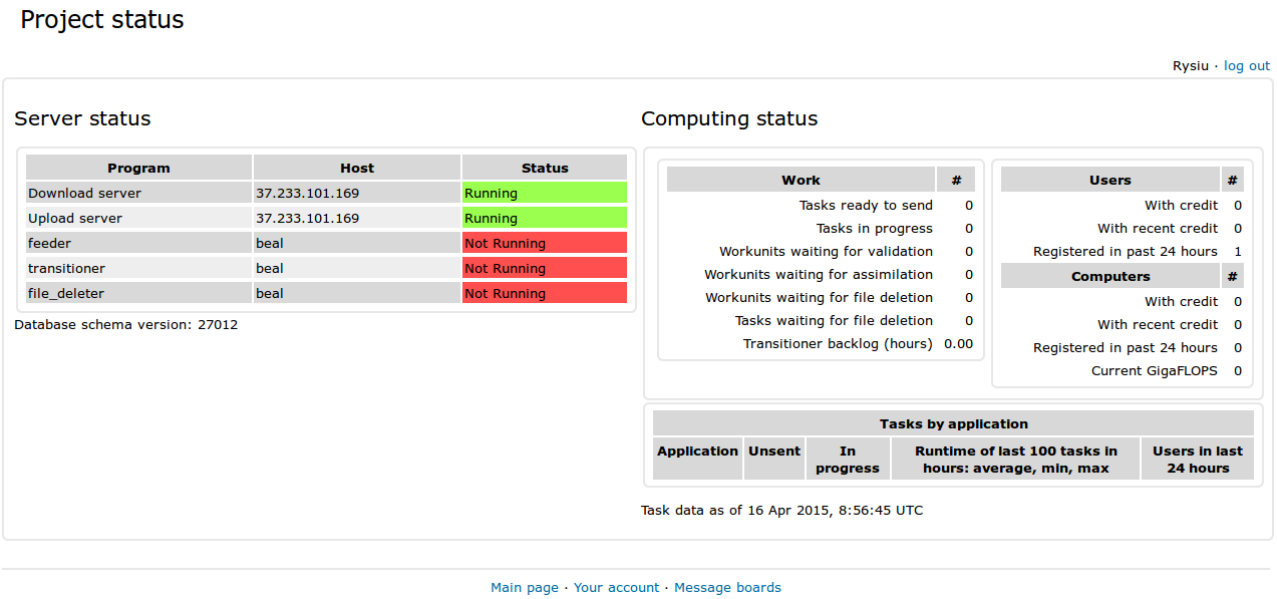
14. Na tym etapie działa już prawidłowo nasza strona internetowa projektu. Możemy wejść na adres http://IP_SERWERA/SKRÓT_PROJEKTU/.

Jeszcze nie skonfigurowana strona serwera wygląda tak jak przedstawiono na Ryc. 18.



Rys. 18. Początkowa strona projektu BOINC.

Inną dość istotną stroną (z której jako administratorzy projektu możemy dość często korzystać) jest *„Project status”*. Wejdziemy na nią klikając na odpowiednie łącze ze strony głównej. Strona ta została zaprezentowana na Ryc. 19.



Ryc. 19. Strona *„Project status”* serwera BOINC.

15. Kolejną czynnością jaką powinniśmy wykonać jest dodanie do Crontab'a (systemowego harmonogramu zadań) odpowiedniej linii uruchamiającej serwer BOINC.

    Wpisujemy polecenie z poziomu użytkownika boincadm:

    crontab -e

    Za pierwszym razem system może się nas spytać jakiego edytora należy użyć do edycji Crontab'a – wybieramy najprostszy *nano* po czym wklejamy do okna linię (dla mnie wygląda tak jak poniżej):

    0,5,10,15,20,25,30,35,40,45,50,55 * * * * cd /home/boincadm/projects/bealf ; /home/boincadm/projects/bealf/bin/start –cron

    Po tym zapisujemy harmonogram i wychodzimy z jego edycji.

16. Jednym ze składników serwera BOINC jest także panel www administratora, który standardowo jest dostępny pod adresem http://IP_SERWERA/SKRÓT_PROJEKTU_ops/. Panel ten jest jednak standardowo chroniony hasłem i aby jego zobaczyć musimy ustalić hasło dostępu.

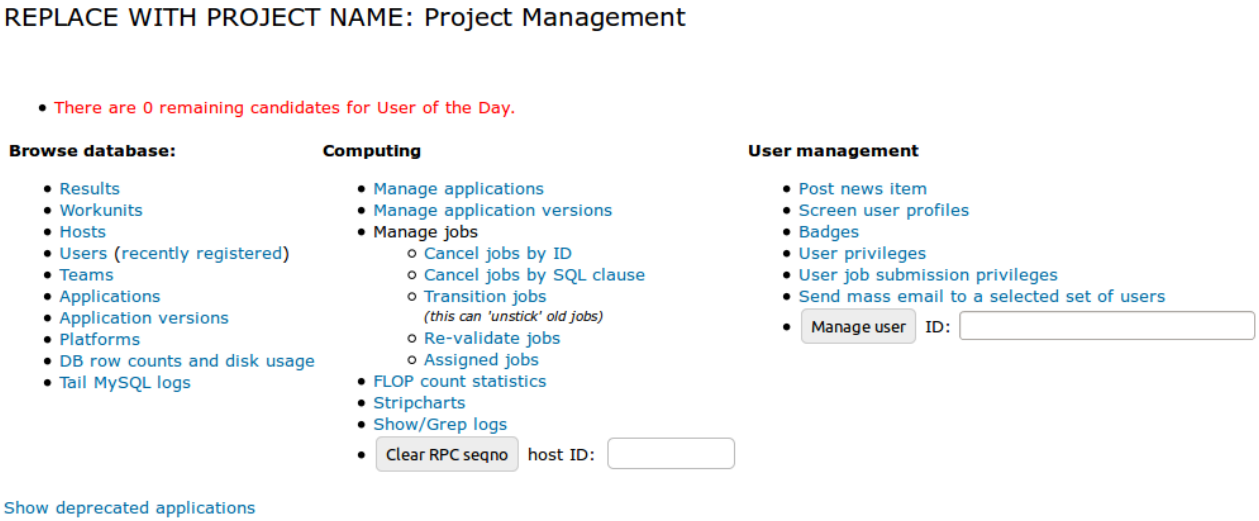
    W tym celu jako użytkownik boincadm wchodzimy do katalogu *~/projects/SKRÓT_PROJEKTU/html/ops/* (w moim przypadku będzie to dokładnie */projects/bealf/html/ops*).

    I wydajemy polecenie:

    htpasswd -c .htpasswd username

    gdzie *username* to oczywiście nazwa użytkownika, którą możemy zdefiniować. System ponadto spyta dwa razy o hasło dla nowego użytkownika.

    Teraz możemy już zalogować się do panelu administratora, który zaprezentowano na Ryc. 20.



Ryc. 20. Górna część panelu administratora serwera BOINC.

17. Na tym etapie możemy już skorzystać z odpowiednich poleceń serwera BOINC odpowiadających za jego uruchomienie, zatrzymanie oraz wyświetlenie statusów daemonów.

    Będąc w katalogu domowym projektu (w moim przypadku jest to dla użytkownika *boincadm*: *~/projects/bealf/*) wprowadzamy polecenie:

    ./bin/start

    Uruchomi to nasz serwer BOINC i zwróci następującą informację:

    Entering ENABLED mode
    Starting daemons
    Starting daemon: feeder -d 3
    Starting daemon: transitioner -d 3
    Starting daemon: file_deleter -d 3

    Aby zobaczyć status daemonów BOINC wystarczy wpisać:

    ./bin/status

    co zwróci odpowiednią informację:

    BOINC is ENABLED

    DAEMON pid status lockfile disabled commandline
    1 8560 running locked no feeder -d 3
    2 8562 running locked no transitioner -d 3
    3 8566 running locked no file_deleter -d 3

    Na koniec można zatrzymać wykonywanie serwera za pomocą.

    ./bin/stop

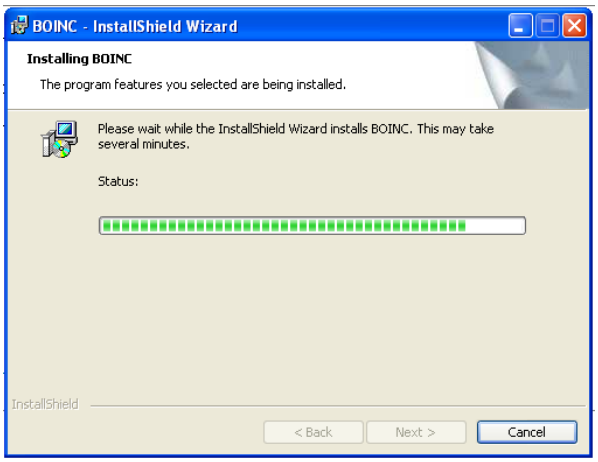
    Entering DISABLED mode
    Stopping all daemons
    Killed process 8566
    Killed process 8560
    Killed process 8562
    Waiting for process 8560 to end: . ok

b) Schemat postępowania podczas instalacji klienta BOINC na systemie Microsoft Windows XP

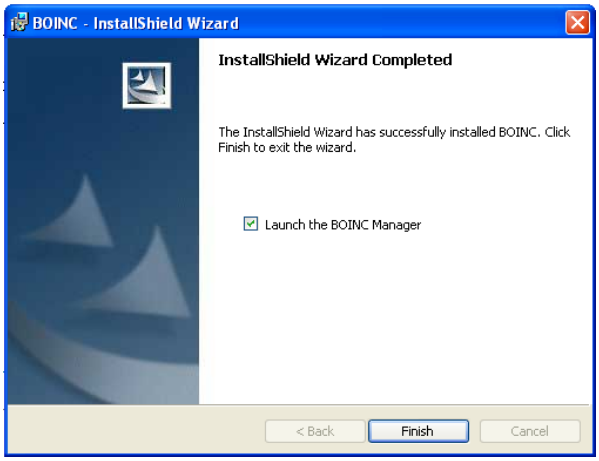
Instalacja klienta BOINC jest bardzo intuicyjna i prosta. Pomimo to postanowiłem ją razem z procesem dołączania do wybranego projektu przedstawić w odrębnym załączniku. Prezentowany proces instalacji dotyczy BOINC klienta w wersji 7.4.42 oraz instalacji z dnia 05.05.2015 r.

1. Aby pobrać klienta BOINC należy wejść na stronę internetową Uniwersytetu Kalifornijskiego w Berkeley: **https://boinc.berkeley.edu.pl/download.php** i kliknąć na odpowiednie łącze prowadzące do pliku instalatora.

2. Następnie uruchamiamy pobrany instalator i postępujemy według informacji. Będziemy musieli zaakceptować licencję oprogramowania oraz wybrać miejsce na dysku, gdzie zostanie zainstalowane – to standardowa procedura.



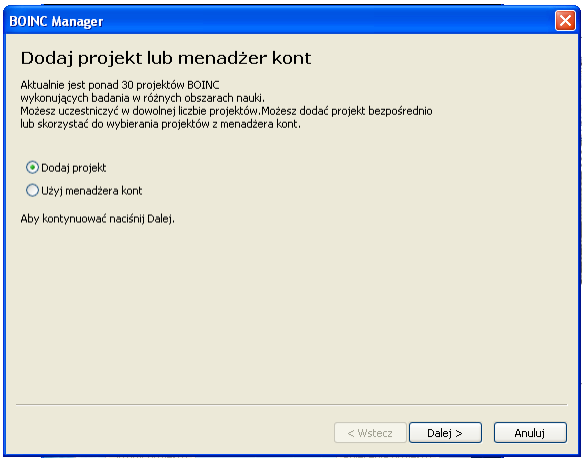
3. Po zakończeniu instalacji, instalator spyta się nas czy uruchomić BOINC. Pozostawiamy zaznaczone *„Launch the BOINC Manager”* i naciskamy *„Finish”*.



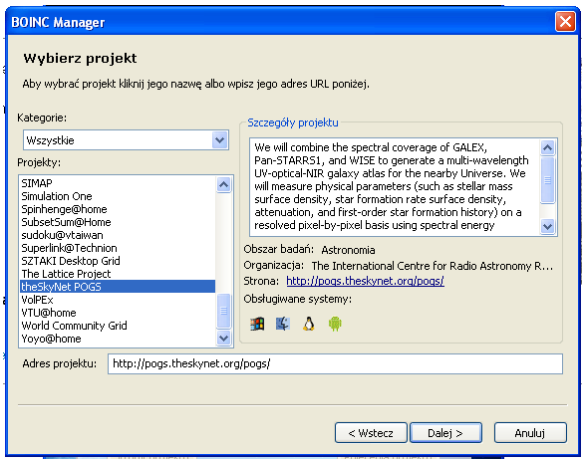
4. Następnie zostanie uruchomiony BOINC Manager w trybie widoku uproszczonego, którego wygląd zaprezentowano poniżej.

5. Kolejnym etapem jest dodanie do BOINC Manager'a nowego projektu. Aby to zrobić wybieramy odpowiednią opcję z górnego menu „Opcje”.

   Uwaga: Czasem BOINC Manager za pierwszym uruchomieniem automatycznie prosi o dodanie nowego projektu.

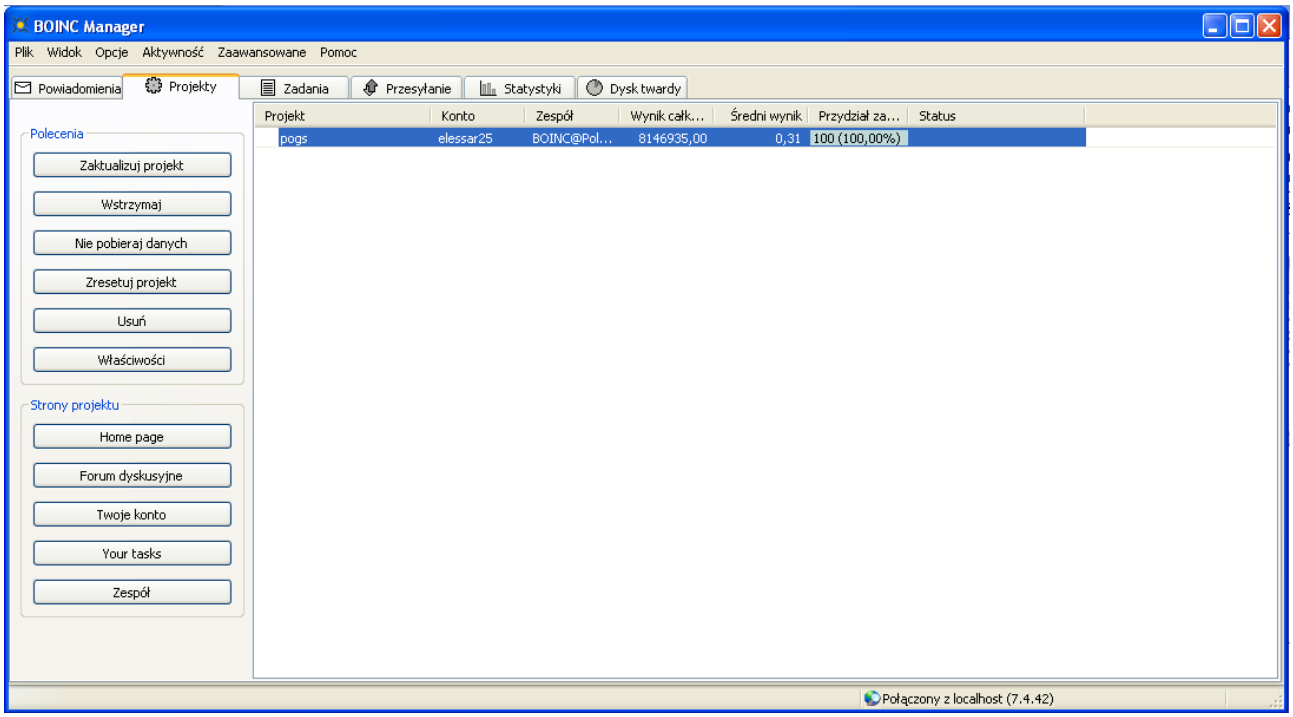


Wybieramy odpowiedni projekt z listy lub wpisujemy adres URL projektu, którego nie ma na liście i akceptujemy nasz wybór. Warto zapoznać się z opisami projektów (w języku angielskim) i platformami na jakie są one dostępne.

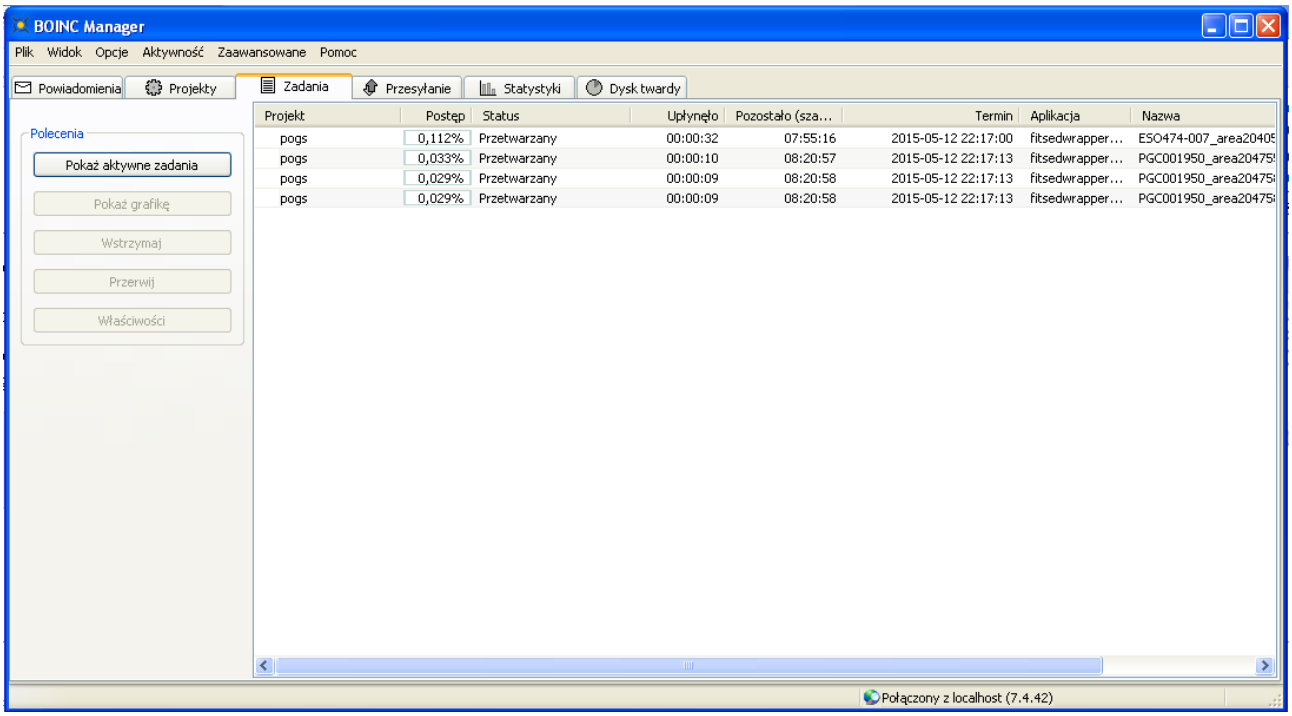


W kolejnym oknie Manager spyta nas o konto. Możemy założyć nowe konto lub wpisać dane już istniejącego. Postępując kilkukrotnie według tego punktu można dodać w identyczny sposób do naszego BOINC Manager'a kilka projektów, które będą wykorzystywać zasoby naszego komputera.

6. Widok z prostego możemy zmienić na zaawansowany – z jego poziomu będziemy mieli znacznie większe możliwości.



Po chwili zadania powinny się pobrać i zacząć liczyć. Ilość jednocześnie wykonywanych zadań intensywnie wykorzystujących CPU jest ograniczona ilością rdzeni/procesorów dostępnych w systemie lub preferencjami użytkownika (może on zdefiniować, że np. na maszynie czterordzeniowej będzie wykorzystywany maksymalnie tylko jeden rdzeń).


BOINC Manager
Plik Widok Opcje Aktywność Zaawansowane Pomoc
Powiadomienia
Projekty
Zadania
Przesyłanie
Statystyki
Dysk twardy
Polecenia
Pokaż aktywne zadania
Pokaż grafikę
Wstrzymaj
Przerwij
Właściwości
Projekt Postęp Status Upłynęło Pozostało (sza... Termin Aplikacja Nazwa
pogs 0,112% Przetwarzany 00:00:32 07:55:16 2015-05-12 22:17:00 fitsedwrapper... ESO474-007_area20405
pogs 0,033% Przetwarzany 00:00:10 08:20:57 2015-05-12 22:17:13 fitsedwrapper... PGC001950_area20475
pogs 0,029% Przetwarzany 00:00:09 08:20:58 2015-05-12 22:17:13 fitsedwrapper... PGC001950_area20475
pogs 0,029% Przetwarzany 00:00:09 08:20:58 2015-05-12 22:17:13 fitsedwrapper... PGC001950_area20475
Połączony z localhost (7.4.42)

c) Kod źródłowy zmodyfikowanej funkcji *init_result()* walidatora BOINC *sample_bitwise_validator*

```
long int file_size(const char *path)
{

    FILE *fp = NULL;
    fpos_t dlugosc;

    if ((fp=fopen(path, "rb"))==NULL)
    {
        printf ("Blad otwarcia pliku: %s!\n", path);
        return -1;
    }
    fseek (fp, 0, SEEK_END); /* ustawiamy wskaźnik na koniec pliku */
    fgetpos (fp, &dlugosc);
    fclose (fp);

    return dlugosc.__pos;

}

int init_result(RESULT& result, void*& data) {
    int retval;
    FILE_CKSUM_LIST* fcl = new FILE_CKSUM_LIST;
    vector<OUTPUT_FILE_INFO> files;
    char md5_buf[MD5_LEN];
    double nbytes;

    MYSQL *conn;
    FILE *pFile;

    char sql_buffer[512];
    char result_buffer[32768];


    if (first) {
        parse_cmdline();
        first = false;
    }

    retval = get_output_file_infos(result, files);
    if (retval) {
        log_messages.printf(MSG_CRITICAL,
            "[RESULT#%u %s] check_set: can't get output filenames\n",
            result.id, result.name
        );
        return retval;
    }

    for (unsigned int i=0; i<files.size(); i++) {
        OUTPUT_FILE_INFO& fi = files[i];
        if (fi.no_validate) continue;

    long int size_result_file;
    size_result_file = file_size(fi.path.c_str());

    if(size_result_file > 14)
    {
        pFile = fopen (fi.path.c_str(), "rb" );
        fread (result_buffer, 1, (size_result_file-15), pFile);
        fclose (pFile);

        conn = mysql_init(NULL);
        mysql_real_connect(conn, "localhost", "root", "40964096BEAL", "beal_final", 0, NULL, 0);

        sprintf (sql_buffer, "INSERT INTO final_results (content, \
        resultid, userid, hostid, teamid, validate_time, sent_time, received_time) \
        VALUES ('%s', '%d', '%d', '%d', '%d', now(), '%d', '%d')",
        result_buffer, result.id, result.userid, result.hostid, result.teamid,
        result.sent_time, result.received_time);

        mysql_query(conn, sql_buffer);

        mysql_close(conn);
    }
```

```
74
75          retval = md5_file(fi.path.c_str(), md5_buf, nbytes, is_gzip);
76          if (retval) {
77              if (fi.optional && retval == ERR_FOPEN) {
78                  strcpy(md5_buf, "");
79                      // indicate file is missing; not the same as md5("")
80              } else {
81                  log_messages.printf(MSG_CRITICAL,
82                      "[RESULT#%u %s] md5_file() failed for %s: %s\n",
83                      result.id, result.name, fi.path.c_str(), boincerror(retval)
84                  );
85                  return retval;
86              }
87          }
88          fcl->files.push_back(string(md5_buf));
89      }
90      data = (void*) fcl;
91      return 0;
92  }
```

d) Kod źródłowy trywialnego generatora zadań wykorzystywanego na platformie BOINC

```
1  #!/bin/bash
2
3  for (( i = 10000 ; i < 10100; i++ ))
4  do
5      for (( j = 0 ; j < 10; j++ ))
6      do
7          let bases_down=i*2000
8          let bases_up=i+1
9          let bases_up=bases_up*2000
10
11         let index_down=j*200
12         let index_up=j+1
13         let index_up=index_up*200
14
15         echo "$bases_down $bases_up 3 1000 $index_down $index_up" > ./download/B-$i-$j
16
17         ./bin/create_work -appname beal_engine -wu_name B-$i-$j -wu_template templates/input.xml
18             -result_template templates/output.xml B-$i-$j
19     done
20 done
```

e) Kod źródłowy zaawansowanego generatora zadań wykorzystywanego na platformie BOINC

```
// Sample command line:
// ./work_generator 5 beal_engine

# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <time.h>
# include <unistd.h>
# include <mysql/mysql.h>

int main(int argc, char **argv)
{
    if(argc < 3)
    {
        printf("Invalid argument. ARGC must be > 2\n");
        printf("For example:\n");
        printf("\t./work_generator 5 beal_engine\n");

        return 1;
    }

    // INFO BLOCK ********** START *********************************************

    if(thread_id == 0)
    {
        printf("================================================\n");
    }

    printf("Application: %s\n", argv[2]);

    if(thread_id == 0)
    {
        printf("================================================\n");
    }

    // INFO BLOCK ********** STOP **********************************************

    time_t ticks;

    char db_server[] = "server";
    char db_user[] = "user";
    char db_password[] = "password";
    char db_database[] = "database";

    int sleep_time;
    sleep_time = atoi(argv[1]);

    long int results_unsend;
    long int number_of_generate_wus;

    char sql_query[512];
    char checkpoint_filename[64];
    char system_query[512];

    MYSQL *conn;

    MYSQL_RES *res;
    MYSQL_ROW row;

    // **** START ************** Get app ID

    conn = mysql_init(NULL);

    if (!mysql_real_connect(conn, db_server, db_user, db_password, db_database, 0, NULL, 0))
    {
        fprintf(stderr, "%> %s\n", mysql_error(conn));
        return 1;
    }

    sprintf (sql_query, "SELECT ID FROM app WHERE name='%s'", argv[thread_id+2]);

    if (mysql_query(conn, sql_query))
    {
        fprintf(stderr, "%> %s\n", mysql error(conn));
```

```
75          mysql_close(conn);
76          return 1;
77      }
78
79      res = mysql_use_result(conn);
80      row = mysql_fetch_row(res);
81
82      int app_id;
83      app_id = atoi(row[0]);
84
85      mysql_free_result(res);
86      mysql_close(conn);
87
88      // **** STOP ************** Get app ID
89
90      sprintf (sql_query, "SELECT COUNT(ID) FROM result WHERE server_state='2' AND appid='%d'", app_id);
91      sprintf (checkpoint_filename, "work_generator_checkpoint_%s", argv[2]);
92
93      int beal_range_size;
94      beal_range_size = 2000;
95
96      int beal_index_range_size;
97      beal_index_range_size = 200;
98
99      while(1)
100     {
101
102         // **** START ************** Get number of unsend results
103
104         conn = mysql_init(NULL);
105
106         if (!mysql_real_connect(conn, db_server, db_user, db_password, db_database, 0, NULL, 0))
107         {
108             fprintf(stderr, "%> %s\n", mysql_error(conn));
109
110             continue;
111         }
112
113         if (mysql_query(conn, sql_query))
114         {
115             fprintf(stderr, "%> %s\n", mysql_error(conn));
116
117             mysql_close(conn);
118             continue;
119         }
120
121
122         res = mysql_use_result(conn);
123         row = mysql_fetch_row(res);
124         results_unsend = atol(row[0]);
125         mysql_free_result(res);
126         mysql_close(conn);
127
128         // **** START ************** Get number of unsend results
129
130         if(results_unsend < 1000)
131         {
132
133             number_of_generate_wus = (1000 - results_unsend) / 10;
134
135             ticks = time(NULL);
136
137             printf("[%s] [%.24s] UNSEND RESULTS: %ld :: GENERATE %ld NEW WU'S\n",
138             argv[2], ctime(&ticks), results_unsend, number_of_generate_wus);
139
140             unsigned long long long int i;
141
142             unsigned long long int checkpoint;
143             unsigned long long int down_range;
144
145             FILE *fp;
146
147             if ((fp=fopen(checkpoint_filename, "rt"))==NULL)
148             {
149                 if ((fp=fopen(checkpoint_filename, "wt"))==NULL)
150                 {
```

```
151                     printf("ERROR file: %s\n", checkpoint_filename);
152                     //return 1;
153                 }
154
155                 checkpoint = 1;
156                 fprintf (fp, "%lld", checkpoint);
157
158                 fclose(fp);
159             }
160             else
161             {
162                 fscanf(fp, "%lld", &checkpoint);
163                 fclose(fp);
164             }
165
166             for(i=0; i < number_of_generate_wus; i++)
167             {
168                 unsigned long long int j;
169                 for(j=0; j < 10; j++)
170                 {
171                     unsigned long long int bases_down;
172                     unsigned long long int bases_up;
173
174                     unsigned long long int index_down;
175                     unsigned long long int index_up;
176
177                     bases_down = checkpoint*beal_range_size;
178                     bases_up = (checkpoint+1)*beal_range_size;
179
180                     index_down = j*beal_index_range_size;
181                     index_up = (j+1)*beal_index_range_size;
182
183                     sprintf (system_query, "echo '%lld %lld 3 1000 %lld %lld' > B-%lld-%lld",
184                             bases_down, bases_up, index_down, index_up, checkpoint, j);
185                     system(system_query);
186
187                     sprintf (system_query, "./bin/create_work -appname beal_engine -wu_name \
188                     B-%lld-%lld -wu_template templates/input.xml -result_template \
189                     templates/output.xml B-%lld-%lld", checkpoint, j, checkpoint, j);
190                     system(system_query);
191                 }
192
193                 checkpoint++;
194
195                 if ((fp=fopen(checkpoint_filename, "wt"))==NULL)
196                 {
197                     printf("[%s] ERROR file: %s\n", argv[thread_id+2], checkpoint_filename);
198                 }
199
200                 fprintf (fp, "%lld", checkpoint);
201                 fclose(fp);
202             }
203         }
204
205         sleep(sleep_time);
206     }
207
208     return 0;
209 }
```

f) Kod źródłowy silnika analizującego przypuszczenie Beal'a

```
#define BOINC_VERSION

#include <inttypes.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

# ifdef BOINC_VERSION
    # include "boinc_api.h"
# endif


typedef struct beal_test_result
{
    unsigned long long int x;
    unsigned long long int y;
    unsigned long long int z;
    unsigned long long int m;
    unsigned long long int n;
    unsigned long long int r;
} beal_test_result_t;

typedef struct powers_list
{
    ;
    unsigned long long int z_r;
    unsigned long long int* z;
    unsigned long long int* r;
    int count;
} powers_list_t;

unsigned int beal_conjecture_test(unsigned long long int min_base, unsigned long long int max_base,
                                  unsigned long long int min_power, unsigned long long int max_power,
                                  beal_test_result_t** results);

unsigned int beal_conjecture_test_boinc(unsigned long long int min_base, unsigned long long int max_base,
                                        unsigned long long int min_power, unsigned long long int max_power,
                                        unsigned long long int min_index, unsigned long long int max_index,
                                        beal_test_result_t** results);

unsigned long long int fast_pow(unsigned long long int p, unsigned long long int q);
unsigned long long int gcd(unsigned long long int a, unsigned long long int b);

int main(int argc, char* argv[])
{

# ifdef BOINC_VERSION

    int rc = boinc_init();
    if (rc)
    {
        fprintf(stderr, "APP: boinc_init() failed. rc=%d\n", rc);
        exit(rc);
    }

    boinc_fraction_done(0.0);

# endif

    // ************* Start block - time variables

    clock_t time_start_total;
    clock_t time_end_total;
    double time_dif_total;

    time_t time_start_total_seconds;
    time_t time_end_total_seconds;

    // ************* Stop block - time variables

    time_start_total_seconds = time(NULL);
```

```
77      unsigned long long int min_base;
78      unsigned long long int max_base;
79      unsigned long long int min_power;
80      unsigned long long int max_power;
81      unsigned long long int min_index;
82      unsigned long long int max_index;
83
84  # ifdef BOINC_VERSION
85      FILE *fp_input;
86
87      fp_input=fopen("input", "rb");
88
89      if (fp_input==NULL)
90      {
91          fprintf (stderr, "\n------ Error read input file -----\n");
92          return 1;
93      }
94
95      fscanf (fp_input, "%lld %lld %lld %lld %lld %lld",
96      &min_base, &max_base, &min_power, &max_power, &min_index, &max_index);
97
98      fprintf (stderr, "\n------ Read input file -----\n");
99      fprintf (stderr, "\nmin_base: %lld; max_base: %lld\n", min_base, max_base);
100     fprintf (stderr, "min_power: %lld; max_power: %lld\n", min_power, max_power);
101
102     fclose (fp_input);
103 # else
104     min_base = atoll(argv[1]);
105     max_base = atoll(argv[2]);
106     min_power = atoll(argv[3]);
107     max_power = atoll(argv[4]);
108 # endif
109
110     int results_count;
111     beal_test_result_t* results;
112     int i;
113
114     clock_t start_time = clock();
115 # ifdef BOINC_VERSION
116     results_count = beal_conjecture_test_boinc(min_base, max_base,
117     min_power, max_power, min_index, max_index, &results);
118 # else
119     results_count = beal_conjecture_test(min_base, max_base, min_power, max_power, &results);
120 # endif
121
122     clock_t end_time = clock();
123
124     if (results_count > 0)
125     {
126         for (i=0; i<results_count; ++i)
127         {
128
129 # ifndef BOINC_VERSION
130             printf("System found example solution:\n");
131             printf("\tx = %lld\n", results[i].x);
132             printf("\ty = %lld\n", results[i].y);
133             printf("\tz = %lld\n", results[i].z);
134             printf("\tm = %lld\n", results[i].m);
135             printf("\tn = %lld\n", results[i].n);
136             printf("\tr = %lld\n", results[i].r);
137             printf(
138                 "\n%lld^%lld + %lld^%lld = %lld^%lld\n\n",
139                 results[i].x,
140                 results[i].m,
141                 results[i].y,
142                 results[i].n,
143                 results[i].z,
144                 results[i].r);
145 # endif
146
147         }
148         free(results);
149     }
150     else
```

```
151     {
152
153 # ifndef BOINC_VERSION
154         printf("System not found solutions of equation x^m + y^n = z^r\n");
155         printf("dla x, y, z in range [%lld,%lld] \n", min_base, max_base);
156         printf("and m, n, r in range [%lld,%lld].\n\n", min_power, max_power);
157 # endif
158     }
159
160
161     time_end_total = clock();
162     time_dif_total = (double)(time_end_total-time_start_total) / CLOCKS_PER_SEC;
163     time_end_total_seconds = time(NULL);
164
165
166     printf("\n\n***********************************************************************\n");
167     printf("Total Execution time:\n");
168     printf("\tReal time: %ld seconds\n", (time_end_total_seconds - time_start_total_seconds));
169     printf("\tSystem time: %lf seconds\n\n", time_dif_total);
170
171
172 # ifdef BOINC_VERSION
173
174     fprintf (stderr, "\nPROCESSING COMPLET\n\n");
175
176     boinc_fraction_done(1.0);
177     boinc_finish(0);
178
179 # endif
180
181     return 0;
182 }
183
184 // PL: l - lewy index tablicy, p - prawy index tablicy
185 // EN: l - left index array, p - right array index
186 long long int binary_search(powers_list_t* table, int l, int p, unsigned long long int z_r)
187 {
188     unsigned long long int sr;
189     while (l<=p)
190     {
191         sr = (l + p) / 2;
192
193         if (table[sr].z_r == z_r)
194             return sr;
195
196         if (table[sr].z_r > z_r)
197             p = sr - 1;
198         else
199             l = sr + 1;
200     }
201     // PL: zwracamy -1, gdy nie znajdziemy elementu
202     // EN: return -1 if we do not find an item in array
203     return -1;
204 }
205
206
207
208 unsigned int beal_conjecture_test_boinc(unsigned long long int min_base, unsigned long long int max_base,
209                                         unsigned long long int min_power, unsigned long long int max_power,
210                                         unsigned long long int min_index, unsigned long long int max_index,
211                                         beal_test_result_t** results)
212 {
213     int results_count = 0;
214     *results = NULL;
215
216     beal_test_result_t* temp_results;
217     unsigned long long int* temp_list;
218
219     int max_base_index = (int)(max_base - min_base);
220     int max_power_index = (int)(max_power - min_power);
221
222     int i, j, k, l, xi, yi, mi, ni;
223     unsigned long long int p, q, x, y, z;
```

```
224     unsigned long long int x_m, y_n;
225
226
227     FILE *fp_powers;
228
229     unsigned long long int** powers =
230     (unsigned long long int**)malloc((max_base_index+1) * sizeof(unsigned long long int*));
231
232     if ( (fp_powers=fopen("powers", "rb"))==NULL )
233     {
234         for (i=0; i<=max_base_index; ++i)
235         {
236             powers[i] = (unsigned long long int*)malloc((max_power_index+1) * sizeof(unsigned long long int));
237             p = min_base+i;
238             q = min_power;
239             powers[i][0] = fast_pow(p, q);
240             for (j=1; j<=max_power_index; ++j)
241             {
242                 powers[i][j] = powers[i][j-1] * p;
243             }
244         }
245         fp_powers=fopen("powers", "wb");
246
247         for (i=0; i<=max_base_index; ++i)
248         {
249             fwrite(powers[i], sizeof(unsigned long long int), sizeof(max_power_index+1), fp_powers);
250         }
251
252         fclose(fp_powers);
253     }
254     else
255     {
256         fp_powers=fopen("powers", "rb");
257
258         for (i=0; i<=max_base_index; ++i)
259         {
260             powers[i] = (unsigned long long int*)malloc((max_power_index+1) * sizeof(unsigned long long int));
261             fread (powers[i], sizeof(unsigned long long int), sizeof(max_power_index+1), fp_powers);
262         }
263
264         fclose(fp_powers);
265     }
266
267     int max_powers_count = (max_base_index+1) * (max_power_index+1);
268     int powers_counter = 0;
269     powers_list_t* sorted_powers = (powers_list_t*) malloc(max_powers_count * sizeof(powers_list_t));
270     for (i=0; i<=max_base_index; ++i)
271     {
272         for (j=0; j<=max_power_index; ++j)
273         {
274             k = 0;
275             while (k<powers_counter && sorted_powers[k].z_r < powers[i][j])
276             {
277                 ++k;
278             }
279             if (k<powers_counter && sorted_powers[k].z_r == powers[i][j])
280             {
281                 temp_list = (unsigned long long int*) realloc(sorted_powers[k].z,
282                 (sorted_powers[k].count+1) * sizeof(unsigned long long int));
283
284                 if (temp_list)
285                 {
286                     sorted_powers[k].z = temp_list;
287                     sorted_powers[k].z[sorted_powers[k].count] = min_base + i;
288                 }
289                 temp_list = (unsigned long long int*) realloc(sorted_powers[k].r,
290                 (sorted_powers[k].count+1) * sizeof(unsigned long long int));
291
292                 if (temp_list)
293                 {
294                     sorted_powers[k].r = temp_list;
295                     sorted_powers[k].r[sorted_powers[k].count] = min_power + j;
296                 }
297                 ++sorted powers[k].count;
```

```
            }
            else
            {
                for (l=powers_counter; l>k; --l)
                {
                    sorted_powers[l].z_r = sorted_powers[l-1].z_r;
                    sorted_powers[l].z = sorted_powers[l-1].z;
                    sorted_powers[l].r = sorted_powers[l-1].r;
                    sorted_powers[l].count = sorted_powers[l-1].count;
                }
                sorted_powers[k].z_r = powers[i][j];
                sorted_powers[k].z = (unsigned long long int*) malloc(sizeof(unsigned long long int));
                sorted_powers[k].r = (unsigned long long int*) malloc(sizeof(unsigned long long int));
                sorted_powers[k].z[0] = min_base + i;
                sorted_powers[k].r[0] = min_power + j;
                sorted_powers[k].count = 1;

                ++powers_counter;
            }
            ++k;
        }
    }

    FILE *fp_progress;
    float progress_value;

    FILE *fp_results;

    FILE *fp_checkpoint;

    unsigned long long int checksum = 0;

    if ( (fp_checkpoint=fopen("checkpoint", "r"))==NULL )
    {
        xi = min_index;
    }
    else
    {
        fscanf(fp_checkpoint, "%lld %lld", &xi, &checksum);
        fclose(fp_checkpoint);
    }


    for (xi; xi<=max_index; ++xi)
    {
        progress_value = (float) (xi - min_index) / (float) (max_index - min_index);

        fp_progress=fopen("progress", "w");
        fprintf (fp_progress, "%f", progress_value);
        fclose(fp_progress);

        boinc_fraction_done(progress_value);

        fp_checkpoint=fopen("checkpoint", "w");
        fprintf (fp_checkpoint, "%lld %lld", xi, checksum);
        fclose(fp_checkpoint);

        boinc_checkpoint_completed();

        x = min_base + xi;
        for (yi=xi+1; yi<=max_base_index; ++yi)
        {
            y = min_base + yi;
            if (gcd(x, y) == 1ULL)
            {
                for (mi=0; mi<=max_power_index; ++mi)
                {
                    x_m = powers[xi][mi];
                    for (ni=0; ni<=max_power_index; ++ni)
                    {
                        y_n = powers[yi][ni];
                        i = binary_search(sorted_powers, 0, powers_counter-1, x_m + y_n);

```

```
371                         checksum += abs(i);
372
373                         if (i != -1)
374                         {
375                             for (j=0; j<sorted_powers[i].count; ++j)
376                             {
377                                 z = sorted_powers[i].z[j];
378
379                                 if (z != x && z != y && gcd(z, x) == 1ULL && gcd(z, y) == 1ULL)
380                                 {
381                                     ++results_count;
382                                     temp_results = (beal_test_result_t*) realloc(*results,
383                                     results_count * sizeof(beal_test_result_t));
384
385                                     if (temp_results)
386                                     {
387                                         *results = temp_results;
388                                         (*results)[results_count-1].x = x;
389                                         (*results)[results_count-1].y = y;
390                                         (*results)[results_count-1].z = z;
391                                         (*results)[results_count-1].m = min_power+mi;
392                                         (*results)[results_count-1].n = min_power+ni;
393                                         (*results)[results_count-1].r = sorted_powers[i].r[j];
394
395                                         fp_results=fopen("result", "a");
396                                         fprintf (fp_results, "%lld^%lld+%lld^%lld=%lld^%lld\r\n",
397                                                  x, (min_power+mi), y, (min_power+ni), z, (sorted_powers[i].r[j]));
398                                         fclose(fp_results);
399                                     }
400                                 }
401                             }
402                         }
403                     }
404                 }
405             }
406         }
407     }
408
409     checksum = 1000 + checksum % 1000;
410
411     fp_results=fopen("result", "a");
412     fprintf (fp_results, "CHECKSUM=%lld", checksum);
413     fclose(fp_results);
414
415
416     for (i=0; i<max_base_index; ++i)
417     {
418         free(powers[i]);
419     }
420     free(powers);
421
422     for (i=0; i<powers_counter; ++i)
423     {
424         free(sorted_powers[i].z);
425         free(sorted_powers[i].r);
426     }
427     free(sorted_powers);
428
429     return results_count;
430 }
431
432
433
434
435 unsigned int beal_conjecture_test(unsigned long long int min_base, unsigned long long int max_base,
436     unsigned long long int min_power, unsigned long long int max_power, beal_test_result_t** results)
437 {
438     int results_count = 0;
439     *results = NULL;
440
441     beal_test_result_t* temp_results;
442     unsigned long long int* temp_list;
443
```

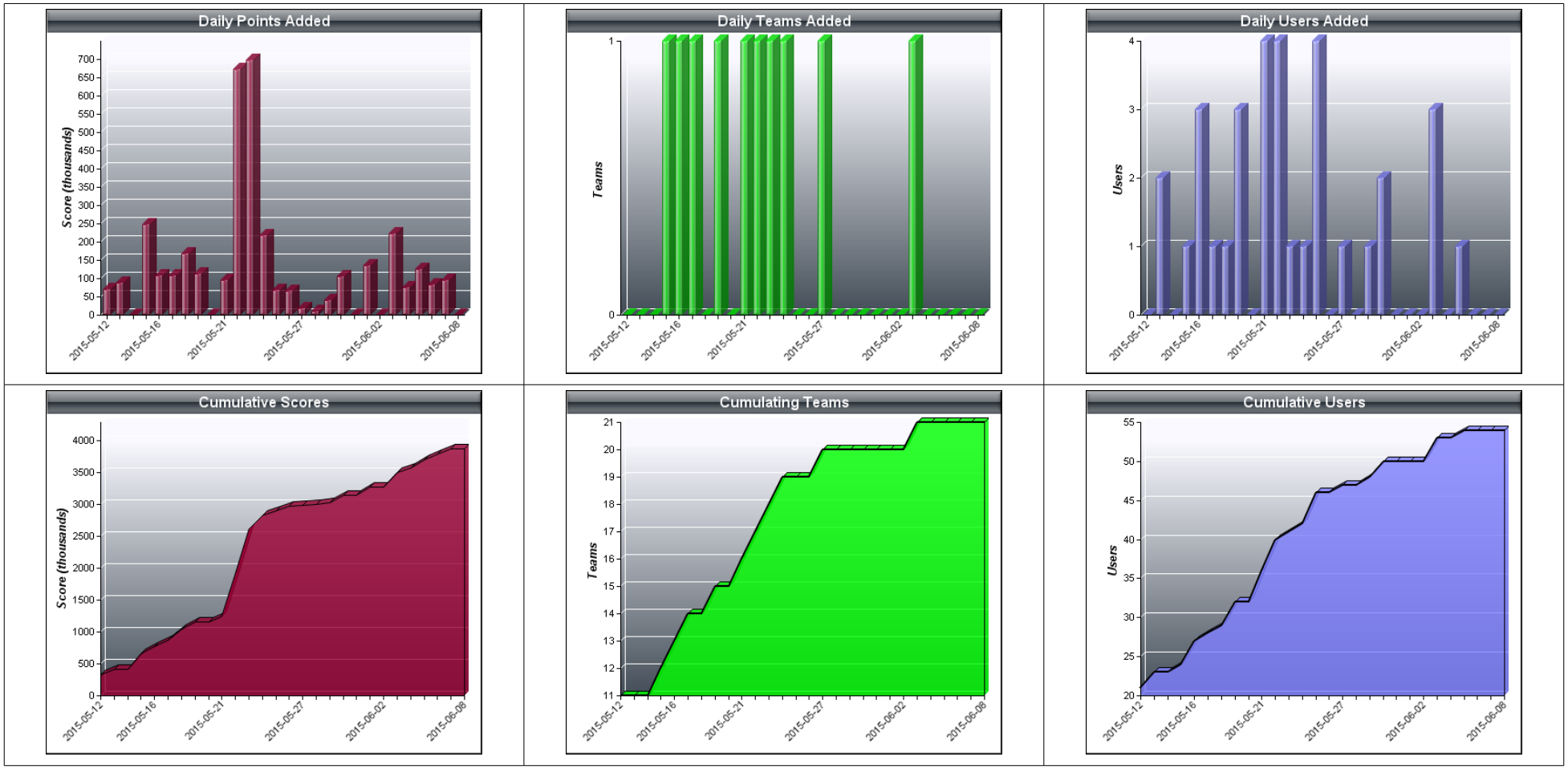
```
444     int max_base_index = (int)(max_base - min_base);
445     int max_power_index = (int)(max_power - min_power);
446
447     int i, j, k, l, xi, yi, mi, ni;
448     unsigned long long int p, q, x, y, z;
449     unsigned long long int x_m, y_n;
450
451     unsigned long long int** powers =
452     (unsigned long long int**)malloc((max_base_index+1) * sizeof(unsigned long long int*));
453
454     for (i=0; i<=max_base_index; ++i)
455     {
456         powers[i] = (unsigned long long int*)malloc((max_power_index+1) * sizeof(unsigned long long int));
457         p = min_base+i;
458         q = min_power;
459         powers[i][0] = fast_pow(p, q);
460         for (j=1; j<=max_power_index; ++j)
461         {
462             powers[i][j] = powers[i][j-1] * p;
463         }
464     }
465
466     int max_powers_count = (max_base_index+1) * (max_power_index+1);
467     int powers_counter = 0;
468     powers_list_t* sorted_powers = (powers_list_t*) malloc(max_powers_count * sizeof(powers_list_t));
469     for (i=0; i<=max_base_index; ++i)
470     {
471         for (j=0; j<=max_power_index; ++j)
472         {
473             k = 0;
474             while (k<powers_counter && sorted_powers[k].z_r < powers[i][j])
475             {
476                 ++k;
477             }
478             if (k<powers_counter && sorted_powers[k].z_r == powers[i][j])
479             {
480                 temp_list = (unsigned long long int*) realloc(sorted_powers[k].z,
481                 (sorted_powers[k].count+1) * sizeof(unsigned long long int));
482
483                 if (temp_list)
484                 {
485                     sorted_powers[k].z = temp_list;
486                     sorted_powers[k].z[sorted_powers[k].count] = min_base + i;
487                 }
488                 temp_list = (unsigned long long int*) realloc(sorted_powers[k].r,
489                 (sorted_powers[k].count+1) * sizeof(unsigned long long int));
490
491                 if (temp_list)
492                 {
493                     sorted_powers[k].r = temp_list;
494                     sorted_powers[k].r[sorted_powers[k].count] = min_power + j;
495                 }
496                 ++sorted_powers[k].count;
497             }
498             else
499             {
500                 for (l=powers_counter; l>k; --l)
501                 {
502                     sorted_powers[l].z_r = sorted_powers[l-1].z_r;
503                     sorted_powers[l].z = sorted_powers[l-1].z;
504                     sorted_powers[l].r = sorted_powers[l-1].r;
505                     sorted_powers[l].count = sorted_powers[l-1].count;
506                 }
507                 sorted_powers[k].z_r = powers[i][j];
508                 sorted_powers[k].z = (unsigned long long int*) malloc(sizeof(unsigned long long int));
509                 sorted_powers[k].r = (unsigned long long int*) malloc(sizeof(unsigned long long int));
510                 sorted_powers[k].z[0] = min_base + i;
511                 sorted_powers[k].r[0] = min_power + j;
512                 sorted_powers[k].count = 1;
513
514                 ++powers_counter;
515             }
516             ++k;
517         }
518     }
```

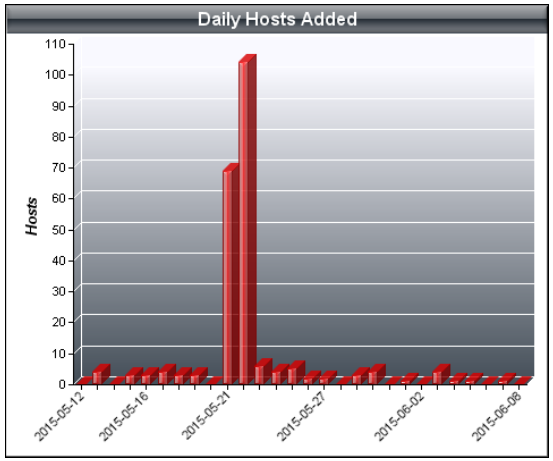
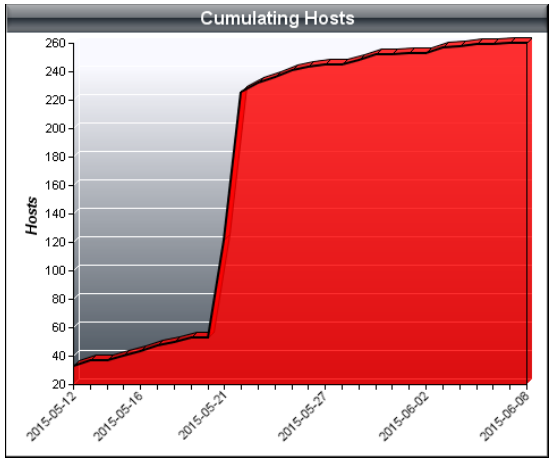
```
519
520     for (xi=0; xi<=max_base_index; ++xi)
521     {
522         x = min_base + xi;
523         for (yi=xi+1; yi<=max_base_index; ++yi)
524         {
525             y = min_base + yi;
526             if (gcd(x, y) == 1ULL)
527             {
528                 for (mi=0; mi<=max_power_index; ++mi)
529                 {
530                     x_m = powers[xi][mi];
531                     for (ni=0; ni<=max_power_index; ++ni)
532                     {
533                         y_n = powers[yi][ni];
534                         i = binary_search(sorted_powers, 0, powers_counter-1, x_m + y_n);
535
536                         if (i != -1)
537                         {
538                             for (j=0; j<sorted_powers[i].count; ++j)
539                             {
540                                 z = sorted_powers[i].z[j];
541
542                                 if (z != x && z != y && gcd(z, x) == 1ULL && gcd(z, y) == 1ULL)
543                                 {
544                                     ++results_count;
545                                     temp_results = (beal_test_result_t*) realloc(*results,
546                                     results_count * sizeof(beal_test_result_t));
547
548                                     if (temp_results)
549                                     {
550                                         *results = temp_results;
551                                         (*results)[results_count-1].x = x;
552                                         (*results)[results_count-1].y = y;
553                                         (*results)[results_count-1].z = z;
554                                         (*results)[results_count-1].m = min_power+mi;
555                                         (*results)[results_count-1].n = min_power+ni;
556                                         (*results)[results_count-1].r = sorted_powers[i].r[j];
557
558                                     }
559                                 }
560                             }
561                         }
562                     }
563                 }
564             }
565         }
566     }
567
568     for (i=0; i<max_base_index; ++i)
569     {
570         free(powers[i]);
571     }
572     free(powers);
573
574     for (i=0; i<powers_counter; ++i)
575     {
576         free(sorted_powers[i].z);
577         free(sorted_powers[i].r);
578     }
579     free(sorted_powers);
580
581     return results_count;
582 }
583
584 unsigned long long int fast_pow(unsigned long long int p, unsigned long long int q)
585 {
586     unsigned long long int r = 1ULL;
587
588     while (q != 0)
589     {
590         if (q%2)
591         {
592             r *= p;
```

```
593         }
594         p *= p;
595         q /= 2;
596     }
597
598     return r;
599 }
600
601 unsigned long long int gcd(unsigned long long int a, unsigned long long int b)
602 {
603     unsigned long long int c;
604     while (a != 0)
605     {
606         c = a;
607         a = b%a;
608         b = c;
609     }
610
611     return b;
612 }
```

g) Statystyki projektu BOINC BeaIF@Home

Na podstawie statystyk Free-DC.



	 Daily Hosts Added Hosts 110 100 90 80 70 60 50 40 30 20 10 0 2015-05-12 2015-05-16 2015-05-21 2015-05-27 2015-06-02 2015-06-08	
	 Cumulating Hosts Hosts 260 240 220 200 180 160 140 120 100 80 60 40 20 2015-05-12 2015-05-16 2015-05-21 2015-05-27 2015-06-02 2015-06-08	

h) Tablica z odnalezionymi rozwiązaniami z wykorzystaniem jedynie zasobów superkomputerowych ICM UW prawidłowymi dla operacji modulo 2^{64}

Równania w postaci: $x^m + y^n = z^r$ *(W kolejności odnalezienia)*

Lp.	x	y	z	n	m	r	Wartość modulo
1	676444	677849	676337	20	818	857	17784773319750998129
2	604942	605391	605965	35	246	456	2288372447511289633
3	1220330	1221513	1220387	38	908	600	4948160361256468193
4	1694050	1695039	1694513	28	108	432	12609605036839803457
5	2372032	2372959	2372289	7	688	552	680105010251968001
6	2172698	2173887	2172769	37	532	760	9706423526405764353
7	2128339	2128438	2129297	560	30	492	13199036057921569473
8	3957038	3957605	3956577	25	880	38	11882078048601687105
9	3859227	3859885	3859618	35	557	31	15501064842072031232
10	4018178	4018833	4019705	41	111	890	5477952416353461617
11	4708052	4708227	4708753	17	732	75	6724712973859386417
12	5240198	5240399	5240533	3	334	306	3866028622879127929
13	5304794	5305287	5304823	34	853	555	8893799672103127271
14	5501351	5501900	5501959	228	24	628	16487540361621293025
15	5574977	5575922	5574213	620	22	576	17465805828848740097
16	5607226	5607697	5607227	42	785	860	12384975864894544401
17	6030307	6031672	6030159	752	9	868	263824262007728833
18	6708793	6709351	6709566	567	203	49	15942179730938134528

Lp.	x	y	z	n	m	r	Wartość modulo
19	6708818	6709351	6708793	50	58	162	569265639489243057
20	6709351	6709678	6708793	58	50	162	569265639489243057
21	6980534	6981047	6980681	30	292	36	14623819350573189025
22	7170340	7170971	7171143	27	612	906	113473291799894129
23	7170838	7170971	7171143	55	408	604	16563665043124141217
24	7170874	7170971	7171143	54	612	906	113473291799894129
25	7170971	7171143	7171454	306	453	53	13249590103723999232
26	7170971	7171270	7171143	612	54	906	113473291799894129
27	7170971	7171624	7171143	612	18	906	113473291799894129
28	7170971	7171862	7171143	408	55	604	16563665043124141217
29	7170971	7171898	7171143	612	54	906	113473291799894129
30	7171143	7171232	7170971	604	11	408	7232206615212473505
31	7171143	7171306	7170971	604	55	408	7232206615212473505
32	7504489	7504556	7505381	402	10	92	7432643896266708881
33	7718927	7719707	7719778	821	596	9	4756025264355303936
34	8234936	8235097	8234449	5	514	11	10930583924314857201
35	8524223	8524634	8524993	256	30	768	10111295053388726273
36	8401517	8401847	8400550	302	611	18	13584895879439187968
37	8738482	8738539	8738035	12	218	78	4609790393328797913
38	8846743	8847064	8846613	882	15	364	12111255234754506385
39	8962177	8963428	8963573	377	20	352	16483435336951299201

i) Tablica z odnalezionymi rozwiązaniami z wykorzystaniem zarówno zasobów superkomputerowych ICM UW, jak i platformy do obliczeń rozproszonych BOINC prawidłowymi dla operacji modulo 2^{64}

Równania w postaci: $x^m + y^n = z^r$ *(W kolejności odnalezienia)*

Lp.	x	n	y	m	z	r	Wartość modulo	Odkrywca	Data odkrycia
1	20070431	770	20070463	113	20071398	25	14155650366401675264	G55-11*	Fri, 08 May 2015 23:19:00 GMT
2	20200287	784	20201362	27	20200409	192	12393561479426804225	G55-11*	Mon, 18 May 2015 04:27:31 GMT
3	20394767	544	20394958	50	20394071	448	7148665709090860545	Zombie67 [MM]	Thu, 21 May 2015 01:25:10 GMT
4	20376694	33	20377193	762	20376099	964	13348672607142987985	Zombie67 [MM]	Thu, 21 May 2015 20:34:04 GMT
5	20432342	18	20433535	530	20433547	576	3277296661692875521	Zombie67 [MM]	Fri, 22 May 2015 01:48:17 GMT
6	20805104	7	20805295	936	20804583	752	4744380995960797313	Zombie67 [MM]	Sat, 23 May 2015 05:19:51 GMT
7	20817022	33	20817127	16	20816479	44	15952473183518859137	Zombie67 [MM]	Sat, 23 May 2015 07:40:39 GMT
8	21028767	608	21029887	815	21028166	32	8895905042805030912	G55-11*	Thu, 04 Jun 2015 22:02:35 GMT