# Wielkoskalowe i zautomatyzowane testowanie przypuszczenia Beal'a

*Bigscale and automatized testing of Beal's Conjecture*

**Łukasz Świerczewski**[1]

**Abstrakt :** Praca prezentuje aspekt adaptacji oraz wykorzystywania algorytmów zaprezentowanych w publikacji [1] na platformie do obliczeń rozproszonych BOINC [2]. Dodatkowo wykonano testy skalowalności przyśpieszenia oprogramowania na takich platformach jak Intel Xeon Phi 5110P [3] oraz platformie wykorzystującej Versatile SMP Foundation Advanced Platform firmy ScaleMP (rozwiązanie klasy vSMP [4][5]). Dzięki długotrwałym obliczeniom udało się znaleźć 47 rozwiązań przystających prawidłowo modulo 264. Żadne z uzyskanych rozwiązań nie jest jednak prawidłowe w przestrzeni całego zbioru liczb naturalnych, a co za tym idzie nie odnaleziono poprawnego kontrprzykładu dla przypuszczenia Beal'a.

**Słowa kluczowe**: *Przypuszczenie Beal'a, BOINC, obliczenia równoległe*

**Abstract** : This paper presents adaptation aspect and use of algorithms presented in publication [1] on distributed computing platform BOINC. What is more, there were made some test of software acceleration scalability on such platforms like Intel Xeon Phi 5110P and platform that uses Versatile SMP Foundation Advanced Platform made by ScaleMP. Thanks to long-lasting computation 47 solutions correctly congruent modulo $2^{64}$ were found. None of the solutions obtained is not correct in the space around the set of natural numbers and what's connected to that, any correct counterexample for Beal's Conjecture was not found.

**Keywords**: *Beal's conjecture, BOINC, parallel computing*

## Wprowadzenie

Przypuszczenie Beal'a jest nierozwiązanym do dnia dzisiejszego problemem matematycznym z gałęzi teorii liczb i mówi ono, że jeśli:

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

gdzie *x, y, z, m, n* oraz *r* są dodatnimi liczbami całkowitymi, *a m, n, r* > 2, to *x, y,* z mają wspólny dzielnik będący liczbą pierwszą.
Pomimo wielu usilnych prób (zarówno analitycznych jak i obliczeniowych) nie udało się tego twierdzenia obalić – znaleźć kontrprzykładu dla liczb z wartościami *x, y, z*, które są parami względnie pierwsze.
Aktualnie za rozwiązanie problemu jest przewidziana nagroda $1 000 000.

## 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 $0 < \boldsymbol{i} < 5\,000$[2] oraz $9\,999 < \boldsymbol{i} < 10\,601$, gdzie zmienna ***i*** definiuje zweryfikowany przedział w jakim mogą znajdować się wartości podstaw ***x, y*** i ***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 Państwowa Wyższa Szkoła Informatyki i Przedsiębiorczości w Łomży, Instytut Automatyki i Robotyki

2.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.
*Tab. 1. Table showing comparison of verified divisions.*

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

## 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.

### 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. 1.

Ryc. 1. Architektura klient – serwer. W centrum umieszczony serwer, do którego są podłączeni klienci. Źródło: Opracowanie własne.
*Fig. 1. Client-Server architecture. Server placed in center and clients connected to it. Source: Own data.*

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).

### Platforma BOINC

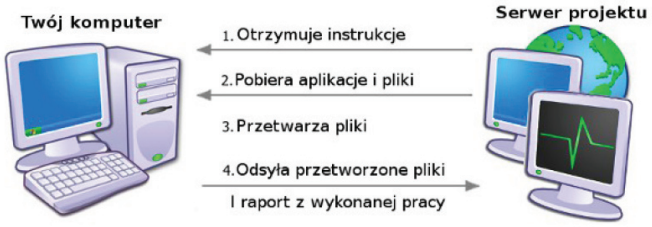
Początkowo platforma BOINC była wykorzystywana jedynie przez projekt poszukiwania cywilizacji pozaziemskich SETI@Home. Z czasem jednak zaczęło z niej korzystać wiele ośrodków naukowych na całym świecie. W Tab. 2 zaprezentowano zestawienie przykładowych projektów BOINC. Istniało oraz istnieje także wiele polskich projektów BOINC (m.in. zaprezentowany w pracy [6] [7]).

3.Do weryfikacji wykorzystano superkomputer HP BladeSystem/Actina – Hydra.
4. 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).
5. W tym etapie obliczeń wzięło udział maksymalnie jedynie 9 węzłów z procesorami Intel Xeon X5660.

| Nazwa | Kategoria | Dziedzina | Właściciel |
|---|---|---|---|
| Asteroids@Home [8] [9] | Nauki fizyczne | Astrofizyka | Charles University in Prague |
| ATLAS@Home [10] | Nauki fizyczne | Fizyka | CERN (European Organization for Nuclear Research) |
| Climateprediction.net [11] | Nauki o Ziemi | Klimatologia | Oxford University |
| Cosmology@Home | Nauki fizyczne | Astronomia | University of Illinois at Urbana-Champaign |
| Einstein@Home [12] [13] | 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 [14] | Biologia i medycyna | Epidemiologia | The Swiss Tropical Institute |
| MindModeling@Home [15] | 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.
*Tab. 2. Example projects working on BOINC platform. Source: Own elaboration based on UC Berkeley.*

Jak zaprezentowano 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. 2.




Ryc. 2. Przepływ danych/instrukcji pomiędzy serwerem BOINC, a klientem. Źródło: Strona zespołu BOINC@Poland.
*Fig. 2. Flow of data/instructions between BOINC server and client. Source: BOINC@Poland website.*

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.

## 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. 3.

| 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. 3. Tabela przedstawiająca specyfikację techniczną wybranych akceleratorów z rodziny Intel Xeon Phi. Źródło: *Opracowanie własne na podstawie danych technicznych firmy Intel*.
*Tab. 3. Table showing technical specifications of selected accelerators from Intel Xeon Phi family. Source: Own elaboration based on Intel website.*

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 [16] przedstawiono w Tab. 4. Analogiczną tabelą dla trybu offload [16] (środowisko OpenMP) jest Tab. 5.

| 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. 4. 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.

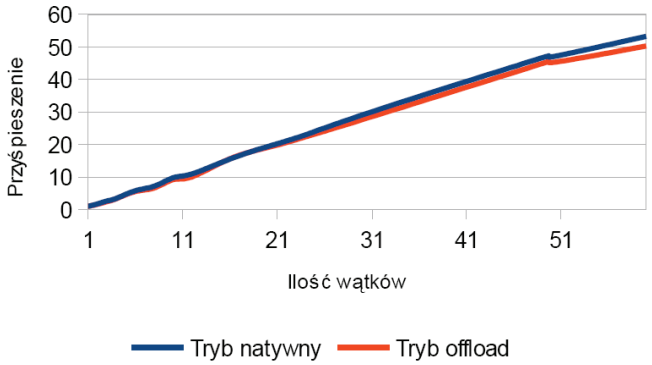
*Tab. 4. Time of realization and obtained accelerations of paralleled solution in OpenMP environment (on Intel Xeon Phi 5110P accelerator in native mode) performed with parameters max_base = 500, max_power = 500. Times measured in seconds. Source: Own data.*

| 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. 5. 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.

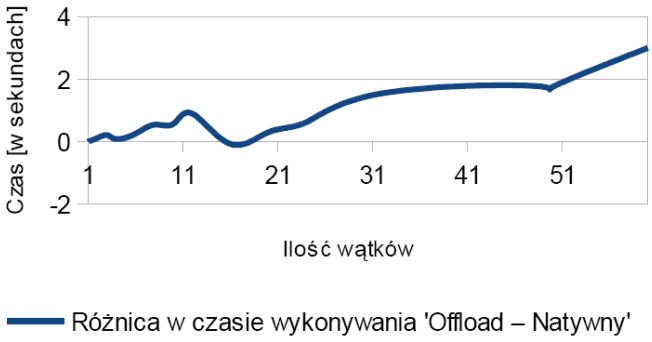
*Tab. 5. Time of realization and obtained accelerations of paralleled solution in OpenMP environment (on Intel Xeon Phi 5110P accelerator in offload mode) performed with parameters max_base = 500, max_power = 500. Times measured in seconds. Source: Own data.*

Jak widać w powyższych tabelach minimalnie lepsze rezultaty (tj. wyższe przyśpieszenia) uzyskano zgodnie z oczekiwaniami w trybie natywnym. Na Ryc. 3 oraz Ryc. 4 można zobaczyć graficzne zobrazowanie na wykresie tych wyników.




Ryc. 3. Porównanie przyśpieszeń uzyskanych na Intel Xeon Phi w trybie offload oraz w trybie natywnym. Źródło: Opracowanie własne.

*Fig. 3. Comparison of accelerations obtained on Intel Xeon Phi in offload mode and in native mode. Source: Own data.*




Ryc. 4. Przedstawienie graficzne różnicy w czasie wykonywania pomiędzy trybem offload, a natywnym. Źródło: Opracowanie własne.

*Fig. 4. Graphical presentation of difference in performance time between offload mode and native mode. Source: Own data.*

Na Ryc. 4 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.

### 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:

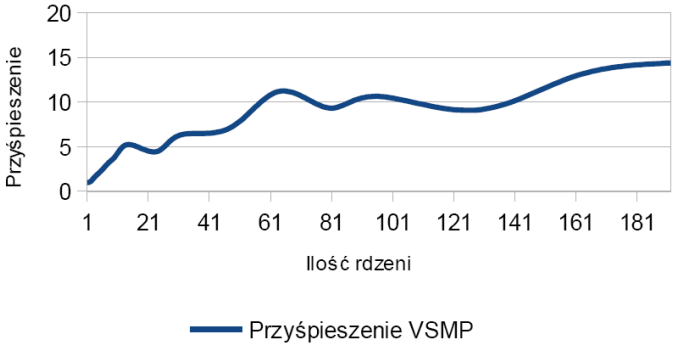
Obudowy typu blade HP C7000 10U, serwery: typu blade HP ProLiant BL490 G6 (16 płyt głównych na obudowę blade), procesory Intel Xeon X5670 @ 2.93GHz, 12MB Cache, architektura EM64T, liczba procesorów: 32, Liczba rdzeni obliczeniowych: 192, Moc obliczeniowa: 2,25 TFlops (całość), całkowita pamięć operacyjna: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

Wyniki analizy wydajnościowej przedstawiono w Tab. 6 oraz Ryc. 5.

| 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. 6. 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.

*Tab. 6. Time results recived during Beal's conjecture analysis of the parallel system based on OpenMP environment performer with parameters max_base = 500, max_power = 500. Times measured in seconds. Source: Own data.*




Ryc. 5. Przyśpieszenia uzyskane na platformie wykorzystującej Versatile SMP Foundation Advanced Platform firmy ScaleMP. Źródło: Opracowanie własne.

*Fig. 5. Accelerations received with use of Versatile SMP Foundation Advanced Platform, by ScaleMP. Source: Own data.*

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.

### 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.

### Dostosowanie do potrzeb silnika analizującego przypuszczenie Beal'a

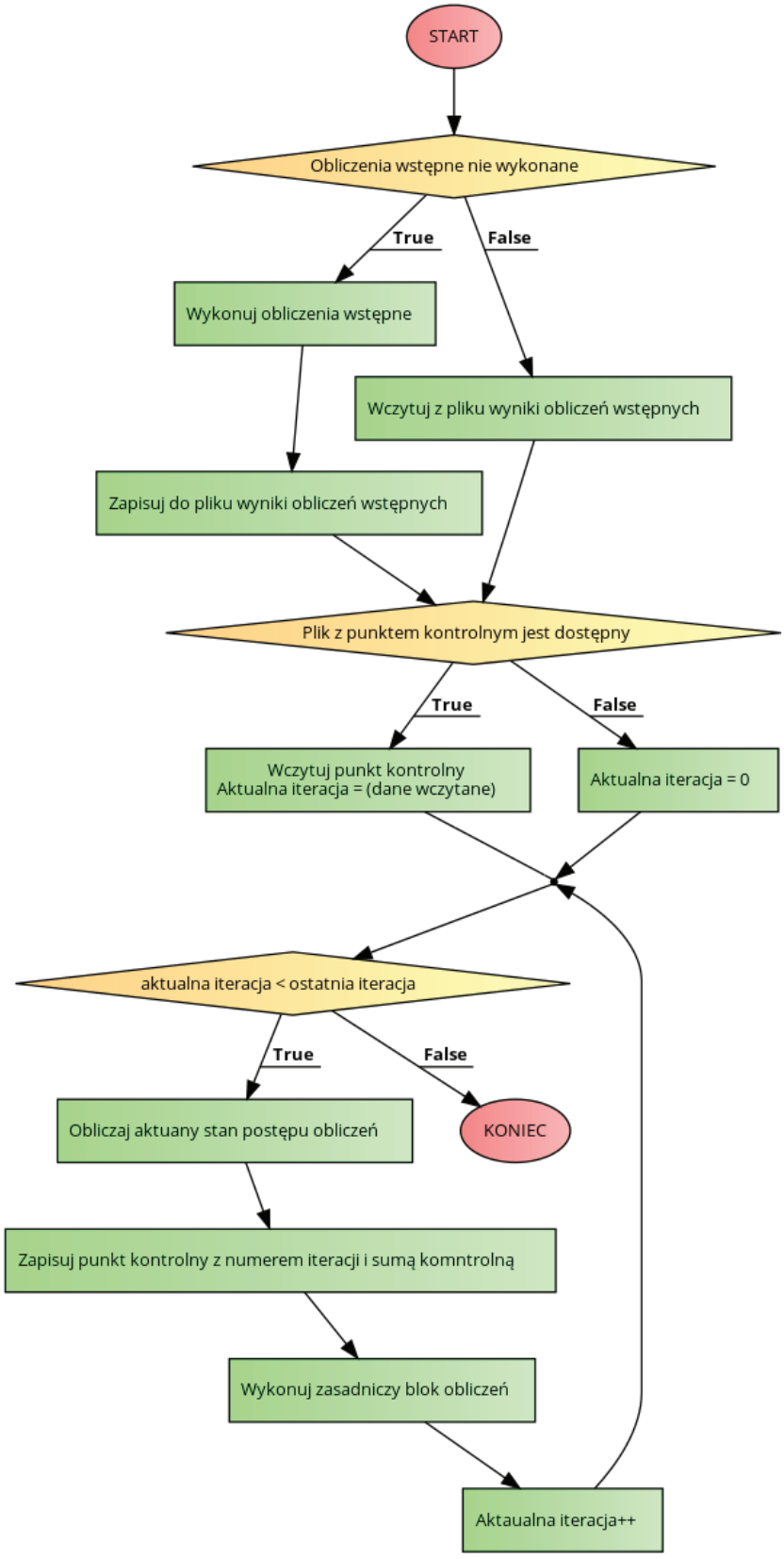
Praktycznie każdą uruchamianą niezależnie przez klienta BOINC aplikację należy dostosować do jego wymogów. Wyjątkiem tutaj mogą być programy obliczeniowe uruchamiane za pomocą tak zwanego *wrappera* (pośrednika). W projekcie BealF@Home zastosowano jednak bezpośrednie wykorzystanie BOINC API z pominięciem funkcjonalności jakie daje wrapper, gdyż jest on całkowicie zbędny[6]. Na Listingu 1 przedstawiono uproszczony schemat zastosowanego rozwiązania. Na Ryc. 6. zaprezentowano natomiast analogiczny schemat blokowy.

6.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.

```
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 1. Uproszczony pseudokod prezentujący możliwość adaptacji programu na potrzeby projektu BOINC. Źródło: Opracowanie własne.

*Listing 1. Simplified pseudocode presenting programs adaptation capabilities for BOINC project. Source: Own data.*




Ryc. 6. Uproszczony schemat blokowy prezentujący możliwość adaptacji programu na potrzeby projektu BOINC. Źródło: Opracowanie własne.
*Fig. 6. Simplified block diagram presenting programs adaptation capabilities for BOINC project. Source: Own data.*

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 2.

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

boinc_fraction_done(0.0);
```

Listing 2. Obsługa funkcji boinc_init() oraz ustawienie postępu obliczeń na 0% za pomocą funkcji boinc_fraction_done(). Źródło: Opracowanie własne.

*Listing 2. Handling of boinc_init() function and setting of computing progres to 0% with use of boinc_fraction_done(). Source: Own data.*

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 3.

```
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();
```

Listing 3. Kod realizujący obliczanie aktualnego stanu postępu obliczeń, aktualizowanie paska postępu oraz zapis do pliku punktu kontrolnego i sumy kontrolnej. Źródło: Opracowanie własne.

*Listing 3. Code realizing computation of actual computing state, actualization of progress bar and record to control point file and control sum. Source: Own data.*

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 4.

```
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);
}
```

Listing 4. 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.

*Listing 4. Code of function searching for counterexample for Beal's conjecture responsible for resuming computation from stored control point (control sum is also loaded). Source: Own data.*

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 5.

```
checksum = 1000 + checksum % 1000;

fp_results=fopen("result", "a");
fprintf (fp_results, "CHECKSUM=%ld", checksum);
fclose(fp_results);
```

Listing 5. Kod prezentujący obliczenie końcowe sumy kontrolnej oraz operację jej zapisu do pliku „result". Źródło: Opracowanie własne.

*Listing 5. Code presenting final computation of control sum and operation of it's record to the „result" file. Source: Own data.*

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.

## 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 6. Na Listingu 7 przedstawiono natomiast zawartość pliku *version.xml*.

```
<job_desc>
    <task>
        <application>beal_enigne</application>
    </task>
</job_desc>
```

Listing 6. Zawartość pliku job.xml. Źródło: Opracowanie własne.
*Listing 6. Contents of job.xml file. Source: Own data.*

```
<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 7. Zawartość pliku version.xml. Źródło: Opracowanie własne.
*Listing 7. Contents of version.xml file. Source: Own data.*

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 8, a przykładowy szablon wyjściowy na Listingu 9.

```
<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 8 Przykładowy szablon wejściowy. Źródło: Opracowanie własne.
*Listing 8. Example input template. Source: Own data.*

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 9. Przykładowy szablon wyjściowy. Źródło: Opracowanie własne.
*Listing 9. Example output template. Source: Own data.*

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. 10.

```
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
```

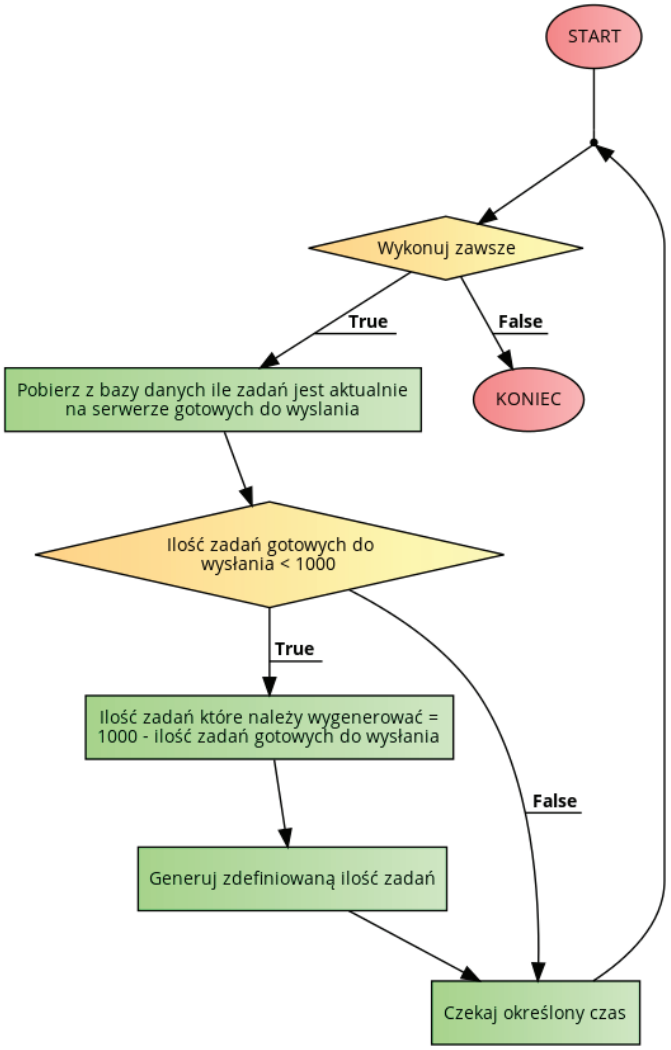
Listing. 10. Wywołanie polecenia ./bin/update_versions podczas pierwszej aktualizacji wersji aplikacji. Źródło: Opracowanie własne.
*Listing 10. Induction of ./bin/update_versions instruction during first actualization of application version. Source: Own data.*

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 schemat blokowy tego generatora zaprezentowano na Ryc. 7.




Ryc. 7. Uproszczony schemat blokowy bardziej zaawansowanego generatora zadań wykorzystanego na platformie BealF@Home. Źródło: Opracowanie własne.
*Fig. 7. Simplified block diagram of more advanced task generator used on BealF@Home platform. Source: Own data.*

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.

## Podsumowanie

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 Załączniku).

## 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.

## Literatura

1. Monika Kwiatkowska, Łukasz Świerczewski, „Testowanie przypuszczenia Beal'a z wykorzystaniem klasycznych procesorów”, Biuletyn Naukowy Wrocławskiej Wyższej Szkoły Informatyki Stosowanej. Informatyka, 2015 5.
2. Swierczewski, Lukasz. "The Distributed Computing Model Based on The Capabilities of The Internet." *arXiv preprint arXiv:1210.1593* (2012).
3. Liu, Tianyu, X. George Xu, and Christopher D. Carothers. "Comparison of two accelerators for Monte Carlo radiation transport calculations, Nvidia Tesla M2090 GPU and Intel Xeon Phi 5110p coprocessor: A case study for X-ray CT imaging dose calculation." *Annals of Nuclear Energy 82* (2015): 230-239.
4. Schmidl, Dirk, et al. "How to scale nested openmp applications on the scalemp VSMP architecture." *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*. IEEE, 2010.
5. Berr, Nicolas, et al. "Trajectory-Search on ScaleMP's vSMP Architecture." *PARCO*. 2011.
6. Świerczewski, Łukasz. "Symulacja funkcjonalnego systemu kwantowego na równoległych komputerach klasycznych IV generacji." (2013).
7. Свєрчевський, Лукаш, and Łukasz Świerczewski. "Polish BOINC projects." *Proceedings of the third international scientific and practical conference FOSS Lviv 2013*. 2013.
8. Ďurech, J., J. Hanuš, and R. Vančo. "Asteroids@ home—A BOINC distributed computing project for asteroid

shape reconstruction." *Astronomy and Computing 13* (2015): 80-84.
9. Durech, Josef, J. Hanus, and R. Vanco. „Asteroids@ Home." *AAS/Division for Planetary Sciences Meeting Abstracts*. Vol. 44. 2012.
10. Cameron, David. ATLAS@ *Home: Harnessing Volunteer Computing for HEP*. No. ATL-SOFT-SLIDE-2015-159. ATL-COM-SOFT-2015-018, 2015.
11. Stainforth, Dave, et al. „Climateprediction. net: Design Principles for Publicresource Modeling Research." *IASTED PDCS*. 2002.
12. Abbott, B. P., et al. „Einstein@ Home search for periodic gravitational waves in early S5 LIGO data." *Physical review d* 80.4 (2009): 042003.
13. Aasi, Junaid, et al. „Einstein@ Home all-sky search for periodic gravitational waves in LIGO S5 data." *Physical Review D* 87.4 (2013): 042001.
14. Krebs, Viola. „Motivations of cybervolunteers in an applied distributed computing environment: MalariaControl. net as an example." *First Monday* 15.2 (2010).
15. Harris, Jack, et al. *Mindmodeling@ Home... and Anywhere Else You Have Idle Processors*. AIR FORCE RESEARCH LAB MESA AZ WARFIGHTER READINESS RESEARCH DIVISION, 2009.
16. Ronald W Green (Intel), Native and Offload Programming Models, September 9, 2012, (Online), https://software.intel.com/en-us/articles/native-and-offload-programming-models.

## Załączniki

### 1. 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 |
| 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 |

**2. Tablica z odnalezionymi rozwiązaniami z wykorzystaniem zarówno zasobów superkomputerowych ICM UW, jak i platformy do obliczeń rozproszonych BOINC prawidłowymi dla operacji modulo 264**

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 | 14155650366 401675264 | G55-11* | Fri, 08 May 2015 23:19:00 GMT |
| 2 | 20200287 | 784 | 20201362 | 27 | 20200409 | 192 | 12393561479 426804225 | G55-11* | Mon, 18 May 2015 04:27:31 GMT |
| 3 | 20394767 | 544 | 20394958 | 50 | 20394071 | 448 | 71486657090 90860545 | Zombie67 [MM] | Thu, 21 May 2015 01:25:10 GMT |
| 4 | 20376694 | 33 | 20377193 | 762 | 20376099 | 964 | 13348672607 142987985 | Zombie67 [MM] | Thu, 21 May 2015 20:34:04 GMT |
| 5 | 20432342 | 18 | 20433535 | 530 | 20433547 | 576 | 32772966616 92875521 | Zombie67 [MM] | Fri, 22 May 2015 01:48:17 GMT |
| 6 | 20805104 | 7 | 20805295 | 936 | 20804583 | 752 | 47443809959 60797313 | Zombie67 [MM] | Sat, 23 May 2015 05:19:51 GMT |
| 7 | 20817022 | 33 | 20817127 | 16 | 20816479 | 44 | 15952473183 518859137 | Zombie67 [MM] | Sat, 23 May 2015 07:40:39 GMT |
| 8 | 21028767 | 608 | 21029887 | 815 | 21028166 | 32 | 88959050428 05030912 | G55-11* | Thu, 04 Jun 2015 22:02:35 GMT |