*Programowanie współbieżne,*
*Maszyny wirtualne*

SYREK Michał[1]
MAŁOPOLSKI Waldemar[2]

# ANALIZA WPŁYWU ZASTOSOWANIA MASZYN WIRTUALNYCH NA WYDAJNOŚĆ OBLICZEŃ WSPÓŁBIEŻNYCH

*W artykule opisano wpływ wykorzystania maszyn wirtualnych na szybkość realizowania obliczeń współbieżnych. Testowano działający współbieżnie program do sortowania danych zapisanych w tablicy. Porównano wyniki testów obliczeń w systemie zainstalowanym natywnie i w środowisku maszyny wirtualnej.*

# IMPACT ANALYSIS OF THE VIRTUAL MACHINE APPLICATION ON CONCURRENT COMPUTING PERFORMANCE

*The article describes the impact of the use of virtual machines on speed of concurrent computation. In the experiment the concurrently running program to sort the data stored in the array was tested. The test results of calculations on the operating system running natively and in a virtual machine environment was compared.*

## 1. WSTĘP

Mechanizmy konkurencji wymuszają postęp technologiczny praktycznie w każdej gałęzi gospodarki. W zastosowaniach komercyjnych oraz naukowych szczególnie dąży się do zwiększenia szybkości obliczeniowej komputerów a tym samym do skrócenia czasu wykonywania obliczeń. Pozwala to na szybsze rozwiązywanie wielu problemów obliczeniowych w różnych dziedzinach życia społecznego, np. w takich jak: wyszukiwanie informacji, wykorzystywanie baz danych, technologie multimedialne, tworzenie animacji i grafika komputerowa, wirtualna rzeczywistość, modelowanie i optymalizacja procesów ekonomicznych i produkcyjnych, przemysł farmaceutyczny, petrochemiczny, medyczny itp. Koszty produkcji samodzielnych jednostek obliczeniowych o wysokiej wydajności są nieopłacalne. Podejście takie wydaje się niepraktyczne. W tym celu rozsądne jest wykorzystanie wypadkowej mocy obliczeniowej działających współbieżnie wielu procesorów, co można uzyskać przez odpowiednią synchronizację ich pracy. Można to osiągnąć, stosując techniki programowania współbieżnego. Z punktu widzenia automatyzacji jest to istotny problem, gdyż programy współbieżne powinny być pisane bez specjalnego wysiłku, a zadanie wykorzystania równoległości powinno być powierzone

[1]Wydział Mechaniczny, Politechnika Krakowska, Al. Jana Pawła II 37, 31-864 Kraków.
[2]Instytut Technologii Maszyn i Automatyzacji Produkcji, Politechnika Krakowska, Al. Jana Pawła II 37, 31-864 Kraków, e-mail: malopolski@m6.mech.pk.edu.pl, tel. +48 12 374 32 13

stworzonym do tego zautomatyzowanym narzędziom, które na dzień dzisiejszy nie są w pełni satysfakcjonujące.

Dążąc do prostego i jednocześnie efektywnego wykorzystania mocy obliczeniowej komputerów wprowadza się podział zadań obliczeniowych na kilka komputerów. Do testowania tego typu rozwiązań przydatne jest zastosowanie maszyn wirtualnych. Przy takim podejściu istotne jest to, jaki wpływ ma wykorzystanie maszyn wirtualnych na szybkość obliczeń współbieżnych.

## 2. PROGRAMOWANIE WSPÓŁBIEŻNE

### 2.1 Pojęcie współbieżności programów

Do niedawna większość programów była projektowana z myślą o przetwarzaniu sekwencyjnym, które odbywa się zwykle na komputerze posiadającym jeden procesor CPU, pozbawiony wielordzeniowości. Program jest dzielony na skończony ciąg serii instrukcji, które są wykonywane jedna po drugiej, przy czym wyłącznie jedna instrukcja może być wykonywana w danej chwili czasu. Serie instrukcji mogą być wykonywane na przemian z seriami instrukcji innego programu, mówi się w tym przypadku o wykonywaniu w przeplocie.

W przypadku przetwarzania współbieżnego program współbieżny jest zbiorem programów sekwencyjnych, które można wykonać równolegle. Rozróżnia się współbieżność od równoległości, gdzie współbieżność jest abstrakcją równoległości zawierającą w sobie wykonanie w przeplocie i wykonanie równoległe. Wówczas wykonania programów współbieżnych mogą, lecz nie muszą nakładać się na siebie w czasie, podczas gdy wykonanie równoległe musi spełniać to założenie. Jest to możliwe do uzyskania, gdy jest dostępnych więcej zasobów obliczeniowych niż jeden procesor. Zasoby obliczeniowe mogą zawierać pojedynczy komputer z wieloma procesorami (ewentualnie procesor wielordzeniowy), określoną liczbę komputerów połączonych siecią, lub kombinację obu. Zadanie obliczeniowe wykazuje zwykle cechy takie, jak możliwość jego podziału na części, które mogą być rozwiązywane równolegle, możliwość wykonania wielu instrukcji programu w każdej chwili czasu. Czas potrzebny na wykonanie takiego programu z wieloma zasobami obliczeniowymi jest krótszy niż w przypadku pojedynczego zasobu.

Bezpośrednią korzyścią z wykorzystania programowania współbieżnego jest oszczędność czasu i pieniędzy – każdy dodatkowy zasób skraca czas wykonania zadania, potencjalnie redukując koszty. Wykorzystanie zasobów rozproszonych – zasoby rozległej sieci lub Internetu mogą być użyte do rozwiązania skomplikowanych zadań. Przykładem może być projekt Folding@home 0 badający proces zwijania białek, gdzie na moc obliczeniową przewyższającą pięć największych superkomputerów świata liczonych łącznie składa się setki tysięcy konsol Sony PlayStation 3 oraz komputerów osobistych wolontariuszy, rozmieszczonych na całym świecie, wymieniających dane za pośrednictwem Internetu. Przyspieszenie wynikające z użycia wielu procesorów w przypadku przetwarzania równoległego wg prawa Amdahla jest ograniczone częścią programu, której nie da się zrównoleglić (musi być wykonywana sekwencyjnie) i wyraża się następująco:

$$Przyspieszenie(N) = \frac{1}{1 - P \ + \frac{P}{N}} \tag{1}$$

gdzie: P - część programu, którą można wykonywać równolegle
(1-P) - część, której się nie da zrównoleglić
N - liczba procesorów

W przypadku, gdy N dąży do nieskończoności, przyspieszenie dąży do 1/(1-P). Przykładowo dla P wynoszącej 80% maksymalny wzrost szybkości będzie pięciokrotny, gdy liczba procesorów N dąży do nieskończoności.

### 2.2 Modele funkcjonalne współbieżności

W modelu scentralizowanym wykonujące się współbieżnie programy mają dostęp i wykorzystują wspólną przestrzeń adresową, co oznacza możliwość korzystania z tych samych zmiennych. Mogą być one odczytywane i zapisywane przez wiele procesów. Związany jest z tym problem przy jednoczesnej próbie modyfikacji przez nie tych samych danych, który mógłby skutkować zapisem przypadkowej wartości. Na poziomie sprzętowym konflikt rozwiązywany jest przez tzw. arbiter pamięci, szeregujący te żądania. W przypadku programów współbieżnych nie jest to wystarczające.

W modelu rozproszonym procesy nie korzystają z pamięci współdzielonej, lecz realizują komunikację opartą o wymianę komunikatów. Wyróżnia się dwa typy komunikacji: synchroniczna i asynchroniczna. Komunikacja synchroniczna ma miejsce, gdy zarówno odbiorca jak i nadawca komunikatu jest gotowy na jego otrzymanie. W przypadku komunikacji asynchronicznej nie jest wymagana gotowość odbiorcy na odebranie komunikatu, gdyż do tymczasowego składowania komunikatów wykorzystuje się bufor.

### 2.3 Poprawność programów współbieżnych

Programy współbieżne mają tą właściwość, że wykonywane sekwencyjnie partie instrukcji jednego procesu mogą się przeplatać z wykonywanymi partiami instrukcji drugiego procesu. Ponieważ przykładowo oba procesy mogą wykorzystywać ten sam obszar pamięci, należy zadbać, by nie doszło do niepożądanych sytuacji. Dla wykazania niepoprawności działania programu wystarczy wskazać jeden przeplot powodujący błędną sytuację. Analogicznie program współbieżny jest poprawny, jeżeli każdy przeplot zapewnia poprawne wykonanie. Analiza poprawności programu współbieżnego nie może się odbyć w sposób podobny do analizy programu sekwencyjnego polegającej na „zdiagnozowaniu" problemu, poprawieniu kodu źródłowego, przekompilowaniu programu oraz uruchomieniu go celem zweryfikowania poprawności. Należy pamiętać, że ponownie uruchomiony program może mieć w wyniku przeplotu inny scenariusz wykonania od tego, w którym pojawił się błąd. Z poprawnością programów współbieżnych związane są własności bezpieczeństwa i żywotności.

Program współbieżny posiada własność bezpieczeństwa, jeżeli nigdy nie dochodzi do niepożądanej sytuacji. Przykładem może być awaryjne przerwanie wykonywania programu, które powinno być zawsze możliwe do zrealizowania.

Własność żywotności mówi, że dla programu współbieżnego każda pożądana sytuacja w końcu zajdzie. Przykładowo klient sieciowy zgłaszający chęć pobrania od serwera pewien zasób, w końcu powinien go otrzymać. Jeśli nigdy by go nie otrzymał, mówi się, że został zagłodzony. Jest to przykład lokalnego braku żywotności. Z globalnym brakiem żywotności - zakleszczeniem - ma się np. do czynienia w sytuacji, w której dwa procesy posiadające pewny zasób utrzymują go w wyłącznej dyspozycji, czekając na zwolnienie innego zasobu zajętego przez drugi proces. Program poprawny ma zapewnione własności żywotności i bezpieczeństwa. Nie jest to proste zadanie do wykonania, zwłaszcza w złożonych programach współbieżnych. W systemach szczególnego ryzyka stosuje się metody formalne weryfikacji ich poprawności, takie jak logiki temporalne 0. Aby program był poprawny, nie może też dojść do zakleszczenia. Nie powinno dojść również do zagłodzenia. Problemem jest odpowiednia synchronizacja zapewniająca spełnienie podanych wyżej warunków.

### 2.4 Mechanizmy synchronizacji

Synchronizacja procesów dla modelu scentralizowanego może być realizowana w oparciu o algorytm Petersona. Jest to jedna z możliwości zsynchronizowania, aczkolwiek prymitywna i mało wydajna dla procesów wykonywanych współbieżnie w przypadku braku wsparcia sprzętowego i programistycznego. Wadą algorytmu jest przede wszystkim określona z góry liczba procesów oraz aktywne oczekiwanie. Praktycznie nie jest stosowany.

Test&Set jest instrukcją specjalną umożliwiającą niepodzielny odczyt i zapis wartości pewnej komórki pamięci. Do wad należą: możliwość zagłodzenia procesu oraz aktywne oczekiwanie.

Semafory są mechanizmem opisanym początkowo przez Edsgera Dijkstrę umożliwiającym kontrolę dostępu do wspólnego zasobu. Semafor to językowa konstrukcja wysokiego poziomu, jest abstrakcyjnym typem danych. Można wykonywać na nim dwie operacje: P (lub wait) oraz V (signal). Są one niepodzielne i wykluczają się wzajemnie. Semafory bywają zaimplementowane m.in. w obszarze jądra systemu operacyjnego. Istnieje również semafor binarny, który jest odmianą semafora ogólnego.

Muteksy (ang. mutex – mutual exclusion) są podobne do semafora binarnego i umożliwiają wyłączny dostęp do danego zasobu. Zwykle jednak wiąże się z nimi koncepcja „właściciela", co oznacza, że wyłącznie może zdjąć blokadę ten, który ją założył.

Monitor jest bardziej zaawansowanym mechanizmem synchronizacji, który eliminuje niektóre wady podanych wyżej metod, głównie chroniąc w pewnym stopniu programistów przed popełnianiem błędów. Można go określić modułem programistycznym obejmującym deklaracje zarówno zmiennych jak i stałych, deklaracje procedur i funkcji. Spełnia zadanie sekcji krytycznej, gdyż nie więcej niż jeden proces może się w nim znajdować. Monitor umożliwia synchronizację oczekiwania na określony zasób lub na zakończenie jakiegoś warunku. Umożliwia to za pomocą specjalnego typu danych, zwanego zmiennymi warunkowymi, które mogą znajdować się jedynie we wnętrzu monitora.

Mechanizmy synchronizacji dla modelu rozproszonego to m.in. potoki (ang. pipes).. Potok można porównać do bufora służącego do wymiany komunikatów. Potok nazwany zaimplementowany jest jako plik, który może być nadpisywany i czytany przez wiele procesów. Potok nienazwany łączy wyjście jednego procesu z wejściem drugiego procesu.

Inną metodą jest zdalne wywoływanie procedur (ang. RPC – Remote Procedure Call). Stosuje się je w przypadku potrzeby wykonania danej usługi przez inny proces. Polega to na wywołaniu żądania z odpowiednimi parametrami i oczekiwaniu na wynik. Wymienia się również takie mechanizmy jak: spotkania (CSP) będące specjalną notacją komunikacji, przestrzenie krotek, oraz różne algorytmy uzgadniania.

### 2.5 Sprzętowa obsługa równoległości

Na podstawie poziomu sprzętowego obsługującego (umożliwiającego) równoległość można dokonać klasyfikacji komputerów równoległych.

Komputery wielordzeniowe posiadają procesor zawierający kilka rdzeni (jednostek arytmetyczno-logicznych) usytuowanych na jednym układzie scalonym. Procesor wielordzeniowy wykonuje wiele instrukcji, w ciągu jego cyklu pracy, z wielu wątków (ciągów instrukcji).

Komputery SMP (Symmetric Multiprocessor) posiadają wiele identycznych procesorów dzielących ze sobą pamięć i połączonych magistralą. Ze względu na ograniczenia przepustowości magistrali komputery symetryczne nie mogą posiadać zbyt wielu procesorów, zazwyczaj nie więcej niż 32.

Komputery rozproszone wymieniają dane poprzez sieć (są to komputery z pamięcią rozproszoną). Połączone w sieć jednostki komunikują się za pomocą wymiany komunikatów. Wydajność obliczeniowa może być bardzo wysoka, co zależy od liczby połączonych komputerów.

Komputery klastrowe są złożone z wielu połączonych ze sobą, blisko usytuowanych komputerów. Z tego względu można rozpatrywać je jako jeden komputer. Połączone są siecią. Duża część najszybszych komputerów na świecie jest tego właśnie typu.

Komputery masowo równoległe (MPP - Massively Parallel Processor) składają się z wielu procesorów połączonych siecią, podobnie jak komputery klastrowe. Jednak w przypadku komputerów masowo równoległych łączące je sieci są wyspecjalizowane, w przeciwieństwie do wolniejszych, standardowych sieci klastrów. Zazwyczaj MPP są większe niż klastry. Każdy procesor posiada własną pamięć i zawiera kopię systemu operacyjnego oraz aplikacji.

W przypadku komputerów Grid do komunikacji między komputerami wykorzystuje się sieć Internet. Ze względu na nie wielką przepustowość i opóźnienia w wymianie danych przetwarzanie sieciowe znajduje zastosowanie prawie wyłącznie w problemach, w których obliczenia można niezależnie podzielić między wiele zasobów obliczeniowych (ang. embarassingly parallel).

Wyspecjalizowane komputery równoległe GPGPU (ang. general-purpose computing on graphics processing units) – wykorzystanie procesorów kart graficznych do ogólnych obliczeń. Obecnie moc obliczeniowa procesorów GPU większości kart graficznych dorównuje mocy obliczeniowej procesorom CPU domowych komputerów. Ze względu na budowę kart graficznych przystosowanych do optymalizacji obliczeń związanych z grafiką (ta w znacznym stopniu opiera się na równoległości danych, obliczeniach macierzowych) znajdują one zastosowanie w rozwiązywaniu ogólnych problemów numerycznych. Przykładem może być architektura CUDA firmy Nvidia, dla której stworzone środowisko programistyczne oparte na języku programowania C. Istnieją również biblioteki dla innych

języków programowania oraz próby stworzenia ustandaryzowanego programowania jak np. framework OpenCL.

FPGA (ang. field programmable gate array) – odnosi się do wykorzystania programowalnych układów logicznych w postaci macierzy bramek do obliczeń równoległych

ASIC (ang. application-specific integrated circuit) – wyspecjalizowane układy scalone, zaprojektowane z myślą o obliczeniach równoległych. Ze względu na wysoki koszt produkcji powstało niewiele tego typu układów.

Komputery wektorowe – wykonujące te same instrukcje na wielu różnych danych, jak sugeruje nazwa m.in. na wektorach; w obecnych czasach rzadko spotykane.

## 3. PROGRAMOWANIE WYKORZYSTUJĄCE RÓWNOLEGŁOŚĆ

Ze względów praktycznych dąży się do zautomatyzowania procesu pisania programów równoległych przez kompilator lub preprocesor Jest to tzw. zrównoleglenie pośrednie. W tym przypadku kompilator analizuje kod źródłowy pod kątem możliwości wykonywania równoległego. Zazwyczaj pętle (for, do) są najczęstszymi kandydatami na automatyczne zrównoleglenie. Analiza obejmuje również partie kodu utrudniające bądź spowalniające równoległość. Dokonuje się badania potencjalnego zysku w stosunku do programu sekwencyjnego. Pomimo ciągłych badań i projektów do tego zmierzających cel udaje się osiągać we wciąż niezadowalającym stopniu. Również programy te nie są pozbawione błędów. Powstało niewiele języków programowania z pośrednim zrównolegleniem m.in. Parallel Haskell, SISAL. Dlatego w celu zwiększenia wydajności programów równoległych wykorzystuje się zrównoleglenie bezpośrednie, za które odpowiedzialny jest programista.

W zależności od modelu pamięci można dokonać klasyfikacji istniejących środków umożliwiających programowanie równoległe. Można je podzielić na m.in. współbieżne języki programowania, interfejsy programistyczne (API) i biblioteki [4]. Jeśli chodzi o model pamięci współdzielonej, POSIX Threads (w skrócie Pthreads) 0 oraz OpenMP są najbardziej rozpowszechnionymi interfejsami programistycznymi, podczas gdy dla modelu pamięci rozproszonej jest to MPI (Message Passing Interface). Spotyka się również rozwiązania hybrydowe, w których zastosowano interfejs przekazywania komunikatów MPI wraz z bazującym na wątkach interfejsem POSIX Threads.

POSIX Threads (Pthreads) jest częścią standardu POSIX (IEEE POSIX 1003.1c) określającego wielowątkowość. Definiuje interfejs programistyczny dla języka C, służący do tworzenia i manipulowania wątkami na systemach wywodzących się z rodziny systemów Unix zgodnych z tym standardem, czyli np. GNU/Linux, FreeBSD, NetBSD, Mac OS X, Solaris. Istnieje również implementacja dla systemów Windows z nieco ograniczoną funkcjonalnością.

OpenMP to interfejs programistyczny umożliwiający programowanie współbieżne w językach C/C++ i Fortran bazując na modelu pamięci współdzielonej na platformach zarówno z rodziny systemów Unix, jak i rodziny Microsoft Windows NT. Składa się ze zbioru dyrektyw kompilatora, funkcji bibliotecznych i zmiennych środowiskowych wpływających na zachowanie uruchomionego programu.

MPI (Message Passing Interface) jest biblioteką programistyczną implementującą model przekazywania komunikatów. Nie jest oficjalnie objęta standardem, lecz można powiedzieć, że stała się nim nieoficjalnie (zwłaszcza w centrach obliczeniowych).

Przeznaczona na platformy z pamięcią rozproszoną, pamięcią współdzieloną oraz na platformy hybrydowe, będące połączeniem dwóch wcześniejszych.

W wykorzystywanym dalej Pthreads kluczowe jest pojęcie wątku. Aby zrozumieć jego znaczenie, warto najpierw przytoczyć związane z nim bliskie pojęcia programu oraz procesu.

Program można zdefiniować jako sekwencję zapisanych instrukcji wykonywaną przez komputer w celu zrealizowania konkretnego zadania – skompilowany kod źródłowy.

Proces jest instancją programu wykonywaną pod kontrolą systemu operacyjnego. Ma przydzielone zasoby (m.in. pamięć, urządzenia we/wy) i „rywalizuje" o dostęp do procesora. Może składać się z wątków.

Wątek to część procesu mogąca się wykonywać współbieżnie z innymi wątkami. Wątki danego procesu mogą współdzielić zasoby, mają dostęp m.in. do tych samych obszarów pamięci, tych samych plików.

Podobnie jak proces, wątek może być odrębną jednostką wykonawczą, której przydzielony zostaje procesor, może mieć konkretny priorytet. Jego niezależność jest umożliwiona m.in. przez posiadanie własnego wskaźnika na element stosu, własnego zbioru oczekujących i zablokowanych sygnałów, rejestrów oraz właściwości harmonogramowania. Gdy proces kończy działanie, wątki do niego należące również przestają istnieć. Podobnie jak w przypadku procesu, operacje mogą być dokonywane na współdzielonym obszarze pamięci, dlatego należy zadbać o implementację synchronizacji. Właściwie można by zamiast wątków korzystać z procesów, lecz ich tworzenie wymaga nieco więcej czasu (konieczne jest przejście w tryb jądra).

## 4. OPRACOWANIE PROGRAMU TESTUJĄCEGO

### 4.1 Przedstawienie problemu

Do testowania szybkości obliczeń współbieżnych na systemie zainstalowanym natywnie i w środowisku maszyny wirtualnej wykorzystano program do sortowania przez scalanie (ang. merge sort). Algorytm ten jest zazwyczaj rekurencyjny i w takiej też postaci został zaimplementowany. Oparty jest na strategii „dziel i zwyciężaj", która polega na podziale problemu na mniejsze problemy, dopóki nie dadzą się prosto, bezpośrednio rozwiązać. Następnie wyniki rozwiązań się scala, otrzymując kompletne rozwiązanie.

Ogólnie algorytm można podzielić na kilka kroków:

1. jeżeli ciąg ma 0 lub 1 elementów, jest posortowany
2. w przeciwnym przypadku należy podzielić nieposortowany ciąg na dwie części po połowie (dla nieparzystej liczby elementów jedna część będzie dłuższa o 1)
3. wykonywać rekursywnie sortowanie przez scalanie, dopóki nie zostanie 1 element
4. scalić posortowane mniejsze ciągi w jeden

### 4.2 Program w języku C, wykorzystanie POSIX Threads

Implementacja sortowania przez scalanie z uwzględnieniem wielowątkowości została zrealizowana w języku C z wykorzystaniem interfejsu programistycznego POSIX Threads w środowisku Linux. W celu napisania programu posłużono się edytorem Geany rozprowadzanym na licencji GNU GPL. Kod został skompilowany za pomocą kompilatora gcc. Sortowaniu poddana została tablica złożona z 107 elementów typu *integer*. Wartości tablicy zostały przypisane pseudolosowo. Kompilacja programu wykorzystującego Pthreads w przypadku kompilatora gcc na GNU Linux musi odbyć się z dodatkowym parametrem –pthread. Chcąc mieć do czynienia z wątkami w programie, należy zadbać o zadeklarowanie zmiennych typu *pthread_t* przechowujących identyfikatory wątków.

## 5. PRZEPROWADZENIE BADAŃ CZASÓW URUCHOMIENIA PROGRAMU

Opracowany program został uruchomiony w systemie Linux na dystrybucji Ubuntu 10.10 2.6.35-23-generic i686. System działał za pośrednictwem maszyny wirtualnej Sun VirtualBox, która zainstalowana była na komputerze z procesorem Intel Core i3 posiadającym dwa rdzenie, na systemie Microsoft Windows 7 Home Premium 64bit.

Wspomniana maszyna wirtualna posiada funkcję konfiguracji liczby wirtualnych rdzeni CPU, przy czym wymagane jest sprzętowe wsparcie dla danego ustawienia. Dlatego, w ustawieniach wybrano dwa procesory i dla tej konfiguracji przeprowadzano testy czasu uruchomienia programu.

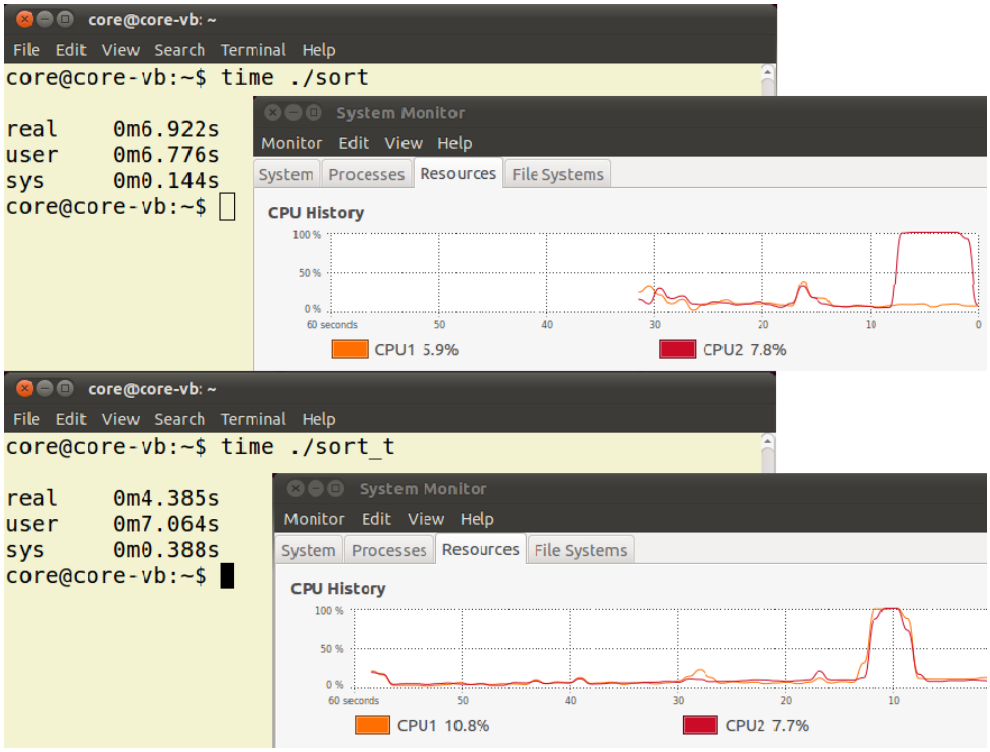
Wykonano pięć prób uruchomienia programu nie wykorzystującego współbieżności oraz pięć prób uruchomienia programu wykorzystującego współbieżność. Czasy uruchomienia mierzone były poleceniem time [opcje] program [argumenty], które przedstawia czasy wykonania programu na poziomie systemu, jądra (sys) oraz na poziomie użytkownika (user). Rzeczywisty czas wykonania programu jest sumą obydwu czasów (real) i ten jest brany pod uwagę w dalszej analizie. Na rysunku 1 widać wykresy użycia procesorów (w tym przypadku rdzeni). Fakt uruchomienia napisanego programu można zaobserwować na wykresie jako prawie stuprocentowy skok wykorzystania procesora. W przypadku programu nie wykorzystującego wielowątkowości widać, że użycie drugiego procesora się nie zmienia.

Na podstawie przeprowadzonych badań średni czas uruchomienia programu bez wykorzystania wielowątkowości na maszynie wirtualnej wynosi 6,895 sekund:

$$t_{(v)} = \frac{6{,}868 + 6{,}931 + 6{,}917 + 6{,}868 + 6{,}890}{5} = 6{,}895 \qquad (2)$$

W przypadku uruchomienia programu wykorzystującego wielowątkowość można zaobserwować skrócony czas jego wykonania oraz wykorzystanie drugiego procesora. Średni czas uruchomienia programu z wykorzystaniem wielowątkowości na maszynie wirtualnej wynosi 3,865 sekund:

$$t_{w(v)} = \frac{3{,}818 + 3{,}868 + 3{,}903 + 3{,}892 + 3{,}844}{5} = 3{,}865 \qquad (3)$$




*Rys. 1 Wykresy użycia procesorów przez program na maszynie wirtualnej*

Dla porównania, skompilowany program uruchomiono na systemie Ubuntu zainstalowanym bezpośrednio (natywnie) na tym samym komputerze, dzięki czemu można było zaobserwować wpływ maszyny wirtualnej na czas wykonania programu. Średni czas uruchomienia programu bez wykorzystania wielowątkowości na systemie zainstalowanym natywnie na komputerze wynosi 6,504 sekund:

$$t_{(n)} = \frac{6{,}476 + 6{,}475 + 6{,}525 + 6{,}564 + 6{,}478}{5} = 6{,}504 \tag{4}$$

Średni czas uruchomienia programu z wykorzystaniem wielowątkowości na systemie zainstalowanym natywnie na komputerze wynosi 3,539 sekund:

$$t_{w(n)} = \frac{3{,}567 + 3{,}540 + 3{,}520 + 3{,}532 + 3{,}535}{5} = 3{,}539 \tag{5}$$

Różnicę procentową czasów uruchomienia programu z użyciem maszyny wirtualnej i bez jej użycia, bez wykorzystania wielowątkowości, można przedstawić następująco

$$\frac{t_{(v)} - t_{(n)}}{t_{(n)}} = \frac{6{,}895 - 6{,}504}{6{,}504} \approx 6\% \tag{6}$$

Różnicę procentową czasów uruchomienia programu z użyciem maszyny wirtualnej i bez jej użycia, z wykorzystaniem wielowątkowości, można przedstawić następująco

$$\frac{t_{w(v)} - t_{w(n)}}{t_{w(n)}} = \frac{3{,}865 - 3{,}539}{3{,}539} \approx 9\% \tag{7}$$

Na systemie zainstalowanym natywnie stosunek wzrostu prędkości czasu wykonania między programem obsługującym wielowątkowość a programem jej nie obsługującym wynosi 1,84

$$\frac{t_{(n)}}{t_{w(n)}} = \frac{6{,}504}{3{,}539} \approx 1{,}84 \tag{8}$$

## 6. WNIOSKI

Na podstawie wyników przeprowadzonych badań czasów wykonywania programu realizującego sortowanie przez scalanie widać znaczne korzyści z wykorzystania programowania współbieżnego, jeśli chodzi o szybkość obliczeniową. W analizowanym przypadku wykorzystanie drugiego rdzenia, przy pomocy POSIX Threads, zwiększyło szybkość wykonania 1,84 razy. Można w dalszej kolejności badać skuteczność innych metod oraz środków programowania współbieżnego, jak np. komunikację międzyprocesową opartą na MPI, framework OpenCL umożliwiający użycie mocy obliczeniowej GPU kart graficznych.

Wykorzystanie maszyny wirtualnej spowodowało wydłużenie czasu obliczeń tylko o 9% w stosunku do obliczeń na systemie zainstalowanym natywnie. Ta niewielka różnica sprawia, że możliwe jest wykorzystywanie maszyn wirtualnych, jako środowiska do testowania różnych rozwiązań z zakresu obliczeń współbieżnych. Szczególnie ciekawe może być testowanie rozwiązań typu Grid w zastosowaniu do modelowania i symulacji systemów. Łatwość pracy z wirtualnymi maszynami czyni je bardzo przydatnymi do stosowania w procesie dydaktycznym.

## 7. BIBLIOGRAFIA

[1] Barney B.: *Introduction to Parallel Computing, Lawrence Livermore National Laboratory*, https://computing.llnl.gov/tutorials/parallel_comp/
[2] Barney B.: *POSIX Threads Programming, Lawrence Livermore National Laboratory*, https://computing.llnl.gov/tutorials/pthreads/
[3] Ben-Ari M.: *Podstawy programowania współbieżnego i rozproszonego*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2009
[4] Weiss Z., Gruźlewski T.: *Programowanie współbieżne i rozproszone w przykładach i zadaniach*, Wydawnictwa Naukowo-Techniczne, Warszawa, 1993
[5] http://pl.wikipedia.org/wiki/Folding@home