Seria: INFORMATYKA z. 28 Nr kol. 1260

Małgorzata STEINDER
Krzysztof ZIELIŃSKI

BADANIE EFEKTYWNOŚCI SYSTEMU PCN W OBLICZENIACH ROZPROSZONYCH[1]

Streszczenie. Artykuł ten ma na celu przedstawienie eksperymentów badających efektywność systemu PCN w obliczeniach rozproszonych. W szczególności przeanalizowano uzależnienie szybkości obliczeń od czynników takich jak: komunikacja między węzłami PCN-a, koszt zmiany miejsca wykonania funkcji oraz związek pomiędzy czasem przekazania do niej parametrów a ich typem i strukturą. Określono również wpływ ziarnistości zadań na możliwą do uzyskania efektywność.

RESEARCH ON EFFICIENCY OF PCN IN DISTRIBUTED COMPUTING

Summary. The aim of this article is to present experiments designed to investigate efficiency of PCN in distributed computing. In particular computing speed dependency of such factors as: communication between runtime systems, process relocation overhead and relationship between parameters passing time and their type and structure, were investigated. The impact of granularity on achievable efficiency was also estimated.

1. Wstęp

Artykuł ten jest prezentacją wyników eksperymentów mających na celu zbadanie efek-

[1]Pracę wykonano w ramach grantu KBN nr 8 S503 015 06.

tywności środowiska PCN i jego przydatności w tworzeniu równoległego i rozproszonego oprogramowania.

PCN (*Program Composition Notation*) [1, 2, 3] jest językiem programowania wysokiego poziomu łączącym w sobie cechy języków imperatywnych i deklaratywnych, jak również systemem służącym do tworzenia i wykonywania programów równoległych. Podstawową jego cechą jest możliwość łączenia części składowych programu w różne formy kompozycyjne, przy czym zdefiniowano trzy podstawowe operatory scalania fragmentów kodu umożliwiając ich sekwencyjne (operator ;), równoległe (operator ||) oraz warunkowe (operator ?) wykonanie. Użytkownikowi pozostawia się możliwość definiowania rozszerzeń do tej notacji. PCN pozwala na łączenie go z innymi językami (C i Fortran), dzięki czemu duże fragmenty sekwencyjnego kodu napisanego w tych językach mogą być wykorzystane w programie równoległym. Tworzenie aplikacji ułatwia możliwość tworzenia topologii wirtualnych i zmiany miejsca wykonania funkcji. Komunikacja i synchronizacja między procesami PCN-a odbywa się za pomocą zmiennych jednokrotnego przypisania (*definitional variables*), co koncepcyjnie odpowiada modelowi *data flow*. Możliwe jest również użycie zmiennych wielokrotnego przypisania (*mutable variables*) odpowiadających pojęciowo zmiennym w językach imperatywnych.

W dalszej części artykułu opisane są mechanizmy systemu umożliwiające rozpraszanie obliczeń oraz wpływ ich wykorzystania na efektywność przetwarzania. W sekcji drugiej przedstawiony jest model obliczeniowy PCN-a oraz narzędzia służące zrównoleglaniu zadań. Sekcja trzecia zawiera opis przeprowadzonych eksperymentów, które obejmują: określenie wpływu istnienia wielu węzłów na szybkość wykonania obliczeń, porównanie lokalnego i zdalnego czasu wykonania funkcji, pomiar czasu przeznaczanego na przełączanie wątków w przypadku procesów wykonujących się równolegle na jednym węźle. Badany też jest koszt związany z przekazywaniem różnego rodzaju parametrów do zdalnie wywoływanej funkcji. W sekcji czwartej zaprezentowane i skomentowane są wyniki obliczeń, a także dokonana jest ocena wpływu ich ziarnistości na efektywność równoległego wykonania zadań. Wynikające z badań wnioski są opisane w sekcji piątej. Artykuł kończy krótkie podsumowanie.

2. Mechanizmy obliczeń rozproszonych w PCN-ie

Podstawowym elementem modelu obliczeniowego PCN-a jest maszyna abstrakcyjna sterująca wykonaniem programu. Równoległa aplikacja może być traktowana jako zbiór lekkich procesów umieszczonych we wspólnej puli, z której w niedeterministyczny sposób są one pobierane w celu redukcji. Koncepcja ta odpowiada modelowi współbieżnego programowania w logice [5].

Maszyna abstrakcyjna PCN-a działa w każdym węźle, na którym wykonywana jest aplikacja. Węzeł rozumiany jest tutaj jako odpowiedni proces Unixa stanowiący środowisko wykonania programu napisanego w PCN; wszystkie uruchomione węzły mogą znajdować się na tym samym, bądź na różnych komputerach. Ilość węzłów i ich lokacja ustalane są w chwili wywołania programu jako parametr systemu, o czym informuje opcja `-pcn`. Instrukcja postaci:

```
prog arg1 arg2 ... argn -pcn -n 4
```

powoduje stworzenie czterech węzłów PCN-a na jednym komputerze za pomocą funkcji Unixa *fork()*. Instrukcja:

```
prog arg1 arg2 ... argn -pcn -nodes host1:host2:host3
```

powoduje wykonanie komendy `rsh` uruchamiającej program `prog` również na każdym z komputerów: `host1`, `host2`, `host3`. Choć możliwe jest użycie `-nodes` i `-n` jednocześnie, to instrukcja taka będzie traktowana tak, jakby użyta była tylko opcja `-nodes`. W przypadku uruchamiania programu na więcej niż jednym komputerze stworzenie na dowolnym z nich więcej niż jednego węzła jest możliwe tylko explicité. Na przykład wywołanie:

```
prog -pcn -nodes host2:host2
```

na *host1* spowoduje stworzenie na nim jednego węzła i dwóch - na *host2*.
Rozpoczęcie wykonywania aplikacji poprzedzone jest ustaleniem połączenia "każdy z każdym" między wszystkimi równolegle działającymi węzłami PCN-a i uruchomieniem na każdym z nich maszyny abstrakcyjnej. Węzeł wywołujący pełni rolę koordynatora. Po ustaleniu połączeń koordynator daje sygnał rozpoczęcia wykonania programu. Przedtem jednak węzłom przypisane zostają unikalne identyfikatory będące kolejnymi liczbami całkowitymi. Koordynator otrzymuje numer 0. Sposób przypisania identyfikatorów do węzłów uruchomionych na pozostałych komputerach nie jest związany z kolejnością ich wystąpienia w komendzie wywołania programu. W aplikacjach użytkownik może poznać liczbę wszystkich węzłów (funkcja *nodes()*) oraz umiejscowienie procesu w sensie numeru węzła, na którym się znajduje (funkcja *location()*). Nazwy komputerów i ich odwzorowanie na identyfikatory są niewidoczne.

Standardowo procesy wykonują się w miejscu, w którym były wywołane. PCN umożliwia jednak zmianę miejsca wykonania procesu PCN-a[2] za pomocą funkcji *node()*. Wywołanie postaci: `proc(...)@node(2)` spowoduje wykonanie procedury `proc(...)` na węźle o numerze 2, jeśli taki istnieje lub 0, jeśli nie istnieje, co zostanie zasygnalizowane odpowiednim komunikatem. Parametr wywołania funkcji *node()* może być zmienną, co sprawia, że w PCN-ie określanie miejsca wykonania procedur może być dynamiczne. Pozwala to na tworzenie wygodnych dla danej aplikacji topologii wirtualnych ukrywających

[2] W dalszej części proces PCN-a będzie nazywany krótko procesem

topologię połączeń fizycznych pomiędzy komputerami i zapewnia możliwość przenoszenia programów pomiędzy systemami o różnych architekturach.

3. Scenariusze eksperymentów

Badanie efektywności systemu PCN ma na celu określenie zarówno korzyści płynących z wykorzystania równoległych jego aspektów jak i kosztów, które wynikają z konieczności komunikacji między węzłami oraz z potrzeby wykonywania dodatkowych operacji; w przypadku zmiany miejsca wykonania procesu jest to przesłanie zadania i jego parametrów, w blokach równoległych – przełączanie między procesami. W punkcie tym podano programy, które użyto w eksperymentach zmierzających do oszacowania tych kosztów. Ich budowa i sposób uruchomienia zostały tak dobrane, aby można było określić wpływ poszczególnych czynników z osobna.

3.1. Wpływ istnienia wielu węzłów

Gdy aplikacja uruchomiona jest na kilku węzłach tworzące je procesy muszą się ze sobą komunikować, nawet jeśli program nie zawiera wywołań postaci `fun(...)@node(...)`. Może to spowodować spowolnienie jego wykonania w stosunku do sytuacji, gdy uruchamiany był na jednym węźle. By określić rozmiar tego spowolnienia zaproponowano następujący scenariusz eksperymentu:
Program `rs_check` mierzący czas lokalnego wykonania funkcji pustej `x()` i funkcji `r()`:

```
x() { ; }
```

```
r(res, n)
int j;
{ ;  j := 0,
     i over 0 .. n - 1 :: { ; j := j + i },
     res = j   }
```

sumującej n kolejnych liczb naturalnych dla różnych wartości n, ma być wykonywany za pomocą następujących wywołań:

```
rs_check
rs_check -pcn -nodes crocus
rs_check -pcn -nodes crocus:pansy
rs_check -pcn -nodes crocus:pansy:daisy
```

oraz

```
rs_check
rs_check -pcn -n 2
rs_check -pcn -n 3
rs_check -pcn -n 4
```

Powoduje to uruchomienie programu na odpowiednio jednym, dwóch, trzech i czterech węzłach. W pierwszym wypadku każdy z węzłów działa na innym komputerze, w drugim wszystkie emulowane są na jednym (*tulip*).

3.2. Straty wynikające ze zmiany miejsca wykonania funkcji

W celu określenia kosztu związanego z przenoszeniem procedur na węzły zaproponowano pomiar czasu wykonania tej samej funkcji `r()` wywołanej w różny sposób. Dokonujący obliczeń program działający na jednym tylko węźle (*tulip*) mierzy czas wykonania instrukcji `r(_, n)` oraz `r(_, n)@node(0)`. Program uruchomiony na dwóch węzłach (*tulip*, *crocus*) bada czas wykonania `r(_, n)@node(0)` oraz `r(_, n)@node(1)`. W tym wypadku jednak, zmierzony czas będzie obejmował narzut związany z istnieniem drugiego węzła, co należy uwzględnić w trakcie analizy. Celowe jest także dokonanie pomiaru w sytuacji, gdy oba węzły emulowane są na jednym komputerze (*tulip*).

3.3. Koszt związany z przekazywaniem parametrów

Konieczność przekazania parametru - zwłaszcza dużego do funkcji łączy się z dodatkowym narzutem czasowym. Aby określić jego rozmiar zaproponowano pomiar czasu wykonania funkcji przenoszonych na węzeł 0 (oznacza to, że w rzeczywistości funkcja wykonywana jest lokalnie) oraz czasu wykonania funkcji przenoszonych na węzeł 1 w aplikacji równoległej (*tulip*, *crocus*) i porównanie go z czasem lokalnego ich wykonania (*tulip*). Odpowiednie do tego celu wydają się następujące proste funkcje:

- funkcja pusta `x()`,
- funkcja, której parametrem jest tablica zawierająca nie mniej niż 100 elementów, wykorzystująca każde z jej pierwszych stu pól:
  ```
  a(X)
  int X[];
  { ;  i over 0 .. 99 :: { ; X[i] := 100 } }
  ```
  Eksperyment polega na zbadaniu czasu wykonania wunkcji dla tablic o różnej długości `X[]` (np. 100, 500, 1000, 1500),
- funkcja, której parametrem jest krotka reprezentująca wyważone drzewo binarne, przeglądająca tylko jej korzeń i pierwsze pokolenie potomków:
  ```
  z(X)
  { ?  X ?= { A, B, C} -> { || z1(B, _), z1(C, _) },
       default -> { ; }  }
  z1(X, Y)
  { ?  X ?= { A, B, C } -> { ; Y = 1 },
       default -> Y = 1  }
  ```

Tak jak poprzednio konieczne jest wykonanie funkcji dla drzew o różnych rozmiarach (przez rozmiar drzewa rozumiemy jego głębokość) np. : 2, 5, 7, 10.

3.4. Przyspieszenie i efektywność wynikające z wykorzystania wielu komputerów

Przyspieszenie i efektywność uzyskiwane w wyniku zastosowania wielu komputerów można określić mierząc czas równoległego wykonania takich samych funkcji, przy czym każda z nich wywołana jest na innym węźle. Celowe jest odniesienie uzyskanych wyników do czasu, jaki zajęłoby wykonanie tej samej ilości funkcji lokalnie w bloku sekwencyjnym.

3.5. Koszt związany z przełączaniem procesów

W blokach równoległych może być jednocześnie wiele aktywnych procesów i muszą być one wykonywane sprawiedliwie, co zmusza maszynę abstrakcyjną do przełączania aktywności pomiędzy nimi. Wiąże sie z dodatkowym narzutem czasowym, który można zbadać porównując czas wykonania bloków: sekwencyjnego i równoległego zawierających tę samą liczbę identycznych funkcji. Przykładowe bloki mają następującą postać:

```
{ ||  r(A1, 1000),                              { ;  r(A1, 1000),
      r(A2, 1000),                                   r(A2, 1000),
      ...                                            ...
      r(An, 1000) }                                  r(An, 1000) }
```

Podobny eksperyment zaproponowano w celu określenia tej samej zależności wykorzystując jednak komunikujące się ze sobą procesy np.

```
{ ||  r1(1, A1),                           { ; r1(1, A1),
      r1(A1, A2),                              r1(A1, A2),
      ...                                      ...
      r1(An, _) }                              r1(An, _) }
```

Warunkiem rozpoczęcia wykonania procesu `r1(A,B)` jest zdefiniowanie (przez inny proces) zmiennej `A`.Tuż przed zakończeniem wykonania zmiennej `B` przypisywana jest wartość. Oba programy uruchomione być powinny na jednym węźle.

4. Warunki realizacji i wyniki obliczeń

Przedstawione w poprzednim punkcie scenariusze eksperymentów zostały wykorzystane w badaniach przeprowadzonych na następujących komputerach:

Nazwa komputera	Typ
tulip	SPARCstation 2
crocus	SPARCstation 2
daisy	SUN LX
pansy	SPARCserver 10

Parametry wykonywanych funkcji (procesów PCN-a) dobrano w taki sposób, by ich czasy wykonania obejmowały możliwie szeroki przedział (od 0.3ms do 70s). Do pomiaru czasu wykorzystano standardową funkcję systemu Unix *gettimeofday()* wywoływaną pośrednio (poprzez funkcję *get_time()* dołączoną do programu w PCN-ie jako funkcja innego języka). Czas wykonania funkcji *get_time()* wynosi ok. 0.7ms. By uniknąć wpływu niedokładności pomiaru czasu i czynników losowych każdy eksperyment powtarzany był wielokrotnie: od kilkunastu razy - w przypadku procesów wykonujących się dłużej niż 30s do kilku tysięcy razy - w przypadku, gdy wykonanie procesu zajmowało mniej niż 1ms. Za wynik eksperymentu uznano średnią arytmetyczną uzyskanych czasów.

4.1. Wpływ istnienia wielu węzłów na czas obliczeń

Komunikacja między węzłami PCN-a, gdy istnieje ich więcej niż 1, powoduje spowolnienie wykonania obliczeń. Eksperyment mierzący wartość tego spowolnienia wykonywano na następujących komputerach: *tulip*; *tulip*, *crocus*; *tulip*, *crocus*, *pansy*; *tulip*, *crocus*, *pansy*, *daisy* odpowiednio dla jednego, dwóch, trzech i czterech aktywnych węzłów.

Wyniki obliczeń przedstawia tabela 1.

Tabela 1: Zależność czasu wykonania procesu od ilości węzłów PCN-a

Czas wykonania [s]				
Funkcja	Liczba węzłów			
	1	2	3	4
`x()`	0.000308	0.000639	0.000667	0.000686
`r(_, 1)`	0.000720	0.001290	0.001342s	0.001480
`r(_, 100)`	0.008815	0.010615	0.010941	0.010696
`r(_, 1000)`	0.095543	0.112255	0.110987	0.112617
`r(_, 5000)`	0.890010	0.976766	0.982615	0.986737
`r(_, 10000)`	2.787579	2.948348	2.967994	2.971299
`r(_, 20000)`	8.576939	8.894937	8.932295	8.951969
`r(_, 50000)`	31.056919	31.847638	31.914848	31.944532

Widoczny w niej przyrost czasu wykonania jest bardzo znaczący dla procesów wykonujących się poniżej 0.1ms i zaniedbywalny, gdy czas wynosi kilka sekund. Duże różnice występują zwłaszcza między programami wykonywanymi na jednym i na dwóch węzłach. Nie zaobserwowano istotnych, dodatkowych przyrostów w przypadku istnienia większej ilości węzłów.

Podobny eksperyment wykonano uruchamiając węzły na jednym komputerze (*tulip*) i otrzymano bardzo zbliżone wyniki. Potwierdza to hipotezę, że przyczyną spowolnienia jest komunikacja między węzłami, a nie inne podejmowane przez nie działania.

4.2. Czas związany z przenoszeniem zadań

Eksperyment wykonywano na następujących komputerach: *tulip*, gdy uruchamiano 1 węzeł, oraz *tulip* i *crocus*, gdy uruchamiano 2 węzły.

Tabela 2: Czas lokalnego i zdalnego wykonania funkcji w sytuacji, gdy węzły znajdują się na różnych komputerach

Czas wykonania funkcji [s]						
Funkcja	Liczba węzłów					
	1		2 na dwóch komputerzch		2 na jednym kompuerze	
	`r()`	`@node(0)`	`@node(0)`	`node(1)`	`node(0)`	`node(1)`
`r(_, 1)`	0.000344	0.001597	0.003825	0.008018	0.004263	0.009144
`r(_, 100)`	0.009381	0.010762	0.014431	0.017858	0.013457	0.023666
`r(_, 1000)`	0.107105	0.104487	0.125927	0.133331	0.126207	0.229300
`r(_, 5000)`	0.943750	0.944723	1.024105	1.098539	1.025484	2.035835
`r(_, 10000)`	2.917255	2.881364	3.064413	3.266331	3.065736	6.242523
`r(_, 20000)`	8.750918	8.845165	9.302739	10.042646	9.318708	20.234818
`r(_, 50000)`	31.705428	31.211962	32.119722	34.784506	32.178991	69.001668
`r(_, 100000)`	69.511059	69.103701	69.947480	74.889831	70.112375	163.323819

Tabela 2 przedstawia otrzymane wyniki. W przypadku istnienia jednego węzła czas wykonania funkcji przenoszonej na węzeł 0 w stosunku do czasu jej zwykłego wykonania jest powiększony o czas przeznaczony na wykonanie funkcji *node(0)* i zmianę miejsca wykonania funkcji `r()` (choć do zmiany tej w rzeczywistości nie dochodzi). Można przyjąć, że narzut ten jest stały - nieco ponad 1ms. Gdy węzłów jest więcej, dużą rolę odgrywa już opisana komunikacja między nimi. Rzeczywiste przeniesienie zadania (`r()@node(1)`) jest znacznie bardziej kosztowne i nawet dla długo wykonujących się obliczeń łączy się z zauważalnymi stratami. Rezultaty otrzymane w wyniku wykonania tego samego eksperymentu na jednym komputerze (*tulip*) z emulacją węzłów przedstawiają ostatnie dwie kolumny. Wyniki te sugerują, że pomimo wykonywania procesu na węźle 1, sama aktywność węzła 0 stanowi duże obciążenie dla procesora.

4.3. Koszt przekazania parametru do zdalnie wykonywanej funkcji

Na czas związany z przekazaniam parametru po stronie nadawcy składają się czas przygotowania danych do transmisji oraz czas transmisji.

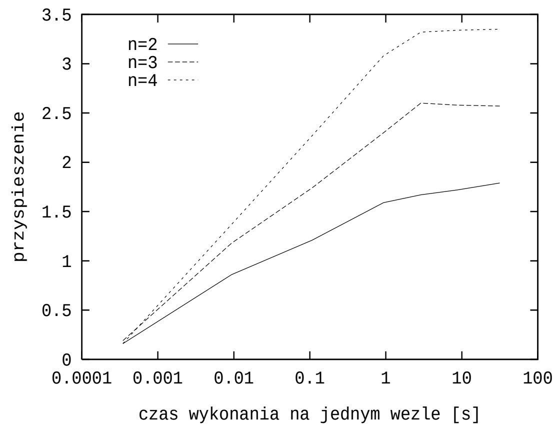
Tabela 3: Czasy wykonania funkcji z parametrami

Funkcja	Rozmiar parametru	Czas wykonania [s]		
		Wywołanie lokalne	`@node(0)`	`@node(1)`
`x()`		0.000308	0.004766	0.011668
`z()`	2	0.000583	0.005318	0.009032
	5	0.000538	0.005912	0.012256
	7	0.000543	0.005590	0.014241
	10	0.000575	0.005518	0.023203
`a()`	100	0.009165	0.013931	0.019298
	500	0.009212	0.013414	0.018247
	1000	0.008765	0.013515	0.019007
	1500	0.009369	0.013798	0.019860

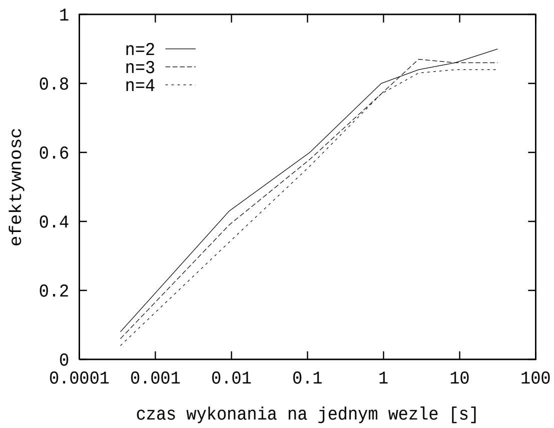
Eksperyment mierzący wartość tych parametrów wykonano tworząc na początku jeden węzeł na *tulip*ie, a następnie dwa: na *tulip*ie i *crocus*ie. Alokacja procesu do węzła 0 nie pociąga za sobą konieczności przekazania parametrów, gdyż nie zmienia się miejsce wykonania zadania. Spowolnienie widoczne w tabeli 3 spowodowane jest istnieniem komunikacji między węzłami i operacjami związanymi z wykonaniem funkcji `@node()` (por. tabela 2). Czas związany z przesłaniem parametru do procesu działającego na innym węźle zależy od jego typu. Gdy jest to liczba lub tablica, to parametr właściwie nie wymaga przygotowania (polega ono na kopiowaniu do bufora), stąd też czas trwania tej operacji, jak też jego zależność od rozmiaru parametru są niewidoczne. Inaczej jest w przypadku, gdy parametrem tym jest krotka. Czas jej przygotowania do transmisji jest duży i wzrasta ze wzrostem złożoności jej budowy. Innymi słowy, koszt związany z przesłaniem parametru jest znaczący tylko wtedy, gdy parametr ten jest krotką lub listą. Zależy on od rozmiaru danej i wynika z konieczności jej przekształcenia do odpowiedniej postaci, a nie z czasu transmisji.

4.4. Ocena wpływu ziarnistości obliczeń na przyśpieszenie i efektywność wynikające z zastosowania kilku komputerów

Do eksperymentu wykorzystano wymienione wcześniej komputery, uruchamiając program kolejno na komputerach: *tulip*; *tulip*, *crocus*; *tulip*, *crocus*, *daisy*; *tulip*, *crocus*, *daisy*, *pansy*. Wyniki obliczeń przedstawione są na wykresach 1 i 2. Na wykresie przyśpieszenia widoczne są trzy krzywe obrazujące stosunek czasu wykonania kilku funkcji w bloku sekwencyjnym na jednym węźle do czasu wykonania tej samej ilości funkcji równolegle na różnych komputerach. Jak można zauważyć przyspieszenie to jest mniejsze od 1 (a więc występują straty) dla małej ziarnistości zadań. Najlepsze rezultaty otrzymuje się rozpraszając zadania liczące się na jednym węźle dłużej niż 1s.



Rysunek 1: Wpływ ziarnistości na przyspieszenie

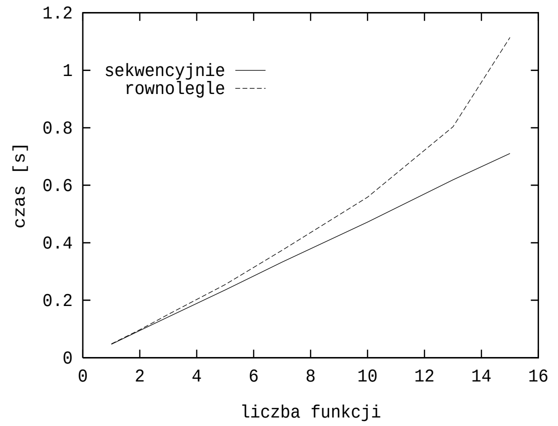


Rysunek 2: Wpływ ziarnistości na efektywność

Na wykresie 2 przedstawione są krzywe obrazujące wpływ ziarnistości obliczeń na efektywność dla różnej liczby komputerów. Wynika z nich, ze maksymalna możliwa do uzyskania jej wartość kształtuje się na poziomie 0.85 dla dużych ziaren. Wśród przyczyn dużych strat pojawiających się w przypadku rozpraszania zadań o małej złożoności należy wymienić bardzo wysokie obciążenie węzłów ich wzajemną komunikacją i znaczący czas przemieszczenia zadania, które to czynniki były już wcześniej analizowane.

4.5. Czas równoległego i sekwencyjnego wykonania pewnej liczby funkcji na jednym węźle

Eksperyment mający na celu określenie kosztów związanych z przełączaniem procesów przez maszynę abstrakcyjną wykonano na *pansy*. W trakcie działania istniał jeden węzeł PCN-a. Różnicę czasu między wykonaniem tej samej ilości funkcji w bloku równoległym i sekwencyjnym przedstawia wykres 3.



Rysunek 3: Narzut związany z przełączaniem procesów

Tabela 4: Porównanie czasów wykonania tej samej ilości funkcji `r1()` w bloku sekwencyjnym i w bloku równoległym

Liczba funkcji	Czas wykonania [s] w bloku sekwencyjnym	równoległym
1	0.025713	0.027163
2	0.054196	0.054294
3	0.081256	0.081663
5	0.133791	0.133327
7	0.189069	0.187473
10	0.267056	0.267481
13	0.347531	0.348938
15	0.403789	0.406757

Zależność czasu wykonania bloku sekwencyjnego od ilości znajdujących się w nim funkcji ma, jak należało się spodziewać, charakter liniowy. Czas ten jest po prostu sumą ich

czasów wykonania. Dla bloku równoległego wykres nie jest liniowy. Czas związany z przełączaniem procesów określa przestrzeń między krzywymi; jak widać wzrasta on nieliniowo wraz ze wzrostem ilości funkcji w bloku.
Warto w tym miejscu zaznaczyć, że w eksperymencie wykorzystano niezależne funkcje (nie komunikujące się ze sobą). Gdy tak nie jest, przełączanie odbywa się tylko między wątkami nie będącymi aktualnie w stanie zawieszenia. Dowodem na to jest tabela 5 przedstawiająca czas wykonania pewnej ilości funkcji `r1()`, opisanej w punkcie 3.5 umieszczonej w bloku sekwencyjnym i równoległym.

5. Wnioski

Oszacowanie takich parametrów jak efektywność i przyśpieszenie w środowisku PCN nie jest zadaniem łatwym. Model obliczeniowy i wynikający z niego sposób realizacji zadań sprawiają, że określenie czasów związanych z transmisją danych jest niewystarczające. W czasie przeprowadzonych ekspryment ów przekonano się, że pod uwagę brana być musi także komunikacja między węzłami PCN-a, a nie tylko zadaniami, które są na nich wykonywane. Zaprezentowane w poprzednim punkcie wyniki pozwalają na wyciągnięcie pewnych ogólnych wniosków dotyczących zasad zrównoleglania obliczeń za pomocą środowiska PCN. Wykonane eksperymenty wykazały, że środowisko to jest najbardziej odpowiednie do łączenia i rozpraszania dużych fragmentów kodu, również napisanych w innych językach programowania. Rozmiar podlegających rozproszeniu zadań powinien być tak dobrany, by zminimalizować wpływ takich czynników jak komunikacja między węzłami, czas zmiany miejsca wykonania funkcji, przesłanie do niej parametrów. Określenie rozmiaru zadań i miejsca ich wykonania nie jest proste. Wiadomo, że powinny to być zadania stosunkowo duże. Przy określaniu miejsca wykonania procedur powinno się brać pod uwagę także ich wzajemne powiązania i typy ich parametrów. Przekazywanie parametrów o typach właściwych dla PCN-a (krotek i list) zwłaszcza o rozbudowanej strukturze łączy się z dużym narzutem czasowym. W niektórych sytuacjach czas przygotowania takiego parametru do transmisji, może być większy niż czas wykonania funkcji, do której jest on przekazywany. Wyniki przedstawione w poprzednim punkcie prowadzą również do spostrzeżenia, że nie należy uruchamiać na jednym węźle dużej ilości niezależnych zadań, gdyż nie tylko nie wpływa to na poprawę efektywności, ale może być przyczyną dużych strat. Podobnie, komunikacja między węzłami działającymi na tym samym komputerze może prowadzić do tego, że uruchomione na nich zadania, będą się wykonywały wolniej, niż gdyby je uruchomiono tylko na jednym węźle.

Biorąc pod uwagę dużą liczbę czynników mających wpływ na efektywność rozproszonego przetwarzania w PCN-ie należy stwierdzić, że tworzenie aplikacji właściwie wykorzystujących to środowisko jest skomplikowane i wymaga niemałego doświadczenia.

LITERATURA

[1] I.Foster, S.Tuecke. *Parallel Programming with PCN*, Technical Report, Argonne National Laboratories, 1992

[2] H.C.Cunningham. *Notes on Concurrent Programming with PCN*, 1992

[3] G.Czajkowski, K.Zieliński *PCN-system rozproszonego przetwarzania*, Informatyka nr 5, 1994

[4] I.Foster, R.Olson, S.Tuecke. *Productive Parallel Programmimg:The PCN Approach*, Scienticic Programming, Vol.1, 51-66, 1992

[5] I.Foster, S. Taylor. *Strand. New Concepts in Parallel Programming*, Prentice Hall, 1990