ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995

Seria: INFORMATYKA z. 29 Nr kol. 1306

Małgorzata BACH
Katarzyna HARĘŻLAK

ANALIZA EFEKTYWNOŚCI WYKORZYSTANIA REKURENCJI ORAZ PROCEDUR JĘZYKA C W SYSTEMIE PCN

Streszczenie. W artykule porównano efektywność wykorzystania iteracji i rekurencji w systemie programowania współbieżnego PCN. Zbadano wpływ stosowanych struktur danych na szybkość realizowanych zadań. Zaprezentowano możliwości dołączania do aplikacji PCN fragmentów kodu sekwencyjnego napisanych w języku C. W pracy zamieszczono również wyniki przeprowadzonych eksperymentów oraz ich analizę.

THE ANALYSIS OF APPLYING RECURSION AND C PROCEDURES EFFICIENCY IN PCN SYSTEM

Summary. This paper contains a comparison of iteration and recursion efficiency in PCN (the parallel programming system). The influence of the data structures used on task performance speed was tested. A possibility for joining sequential procedures written in C with a PCN application is presented. The results of the experiments carried out and their analysis are also included in this article.

Pracę wykonano w ramach grantu KBN 3 P406 011 04

DIE ANALYSE DER AUSNUTZUNGSEFFIZIENZ DER REKURSION UND DER PROZEDUREN DER C-SPRACHE IM SYSTEM PCN

Zusammenfassung. Im Artikel wurde die Effizienz der Ausnutzung der Iteration und der Rekursion im System parallerer Programmierung PCN verglichen. Es wurde der Einfluß verwendeter Datenstrukturen auf die Geschwindigkeit realisierter Aufgaben untersucht. Man hat die Möglichkeit der Hinzufügung des sequentiellen C-Codes zu dem Code einer PCN-Applikation vorgestellt. Der Artikel enthält auch die Ergebnisse durchgeführter Experimente und ihre Analyse.

1. Wstęp

Konieczność rozwiązywania coraz bardziej skomplikowanych zadań zmusza do tworzenia narzędzi umożliwiających programowanie równoległe w systemach wieloprocesorowych lub rozpraszanie wykonywanego programu na komputery połączone siecią lokalną. W niniejszym opracowaniu skupiono się na systemie PCN (Program Composition Notation), będącym równocześnie środowiskiem i językiem programowania współbieżnego, opracownym przez naukowców z Argonne National Laboratory i California Institut of Technology. Twórcy skoncentrowali się na idei kompozycji, stwarzając w ten sposób możliwość łączenia części składowych w programy o dowolnym stopniu komplikacji, co zapewnia łatwość wielokrotnego wykorzystania kodu oraz wymusza czytelność i strukturalność. W środowisku PCN zdefiniowano trzy podstawowe operatory scalania kodu:

– operator wyboru (?),
– operator obliczeń równoległych (||),
– operator obliczeń sekwencyjnych (;).

Język PCN jest integralną częścią systemu, jego składnia jest podobna do składni języka C, ale zestaw dostępnych operatorów i typów danych znacznie skromniejszy. Użytkownik może korzystać z danych typu znakowego, całkowitego i podwójnej precyzji, może również tworzyć jednowymiarowe tablice danych wyżej wymienionych typów. Złożone struktury są reprezentowane za pomocą krotek o ogólnej postaci:

$$\{term_0, \ldots, term_k\}$$

($term_0,..., term_k$ – definicyjne struktury danych np.{1,"napis",{2}}) oraz list, czyli krotek dwuelementowych, z których pierwszy reprezentuje głowę a drugi ogon listy (np. [a1|a2]). Zestaw dostępnych operatorów jest również niewielki, możemy korzystać z dodawania, odejmowania, mnożenia, dzielenia i obliczania reszty z dzielenia (+,–,*,/,%). Zmienne w programach PCN można podzielić na:

- zmienne wielokrotne (ang. mutable variables), czyli takie, którym wartość może być nadawana wielokrotnie w trakcie działania programu (do przypisania wartości służy operator :=),
- zmienne definicyjne (ang. definitional variables), którym wartość można przypisać tylko raz; wykorzystywane są głównie do komunikacji i synchronizacji procesów (do przypisania wartości służy operator =).

W dalszej części artykułu zaprezentowano zasady stosowania iteracji i rekurencji oraz przedstawiono możliwość dołączania do aplikacji stworzonych w PCN fragmentów kodu sekwencyjnego napisanych w języku C i wykorzystywania ich w programach równoległych. Zamieszczono również wyniki przeprowadzonych eksperymentów oraz ich analizę.

2. Iteracja i rekurencja

PCN oferuje użytkownikom tylko jedną instrukcję iteracyjną. Ogólna jej postać jest podobna do instrukcji tego samego typu w innych językach programowania, jak na przykład Pascal czy Fortran:

{op index over low .. high :: block}

gdzie *op* jest jednym z trzech operatorów:

- ? – wybór, następujące po nim bloki są implikacjami,
- ; – obliczenia sekwencyjne,
- || – obliczenia równoległe.

W wyniku działania tej instrukcji blok *block* zostanie wykonany dla każdej liczby całkowitej *index*, takiej że: *low* <=*index*<= *high*. Iteracja jest wykorzystywana głównie w przypadkach, kiedy obliczenia dotyczą zmiennych wielokrotnych.

Ze względu na fakt, iż w systemie PCN dostępna jest tylko jedna instrukcja iteracyjna, użytkownik zmuszony jest odwoływać się do rekurencji. Użyte zostało słowo "zmuszony", ponieważ konstrukcja ta nie jest powszechnie lubiana i raczej niechętnie wykorzy-

stywana przez programistów. Przełamanie tego oporu może dać w efekcie wiele korzyści, gdyż dobre zrozumienie zasad działania rekurencji może zaowocować powstaniem efektywnych programów. Do zalet rekurencji możemy także zaliczyć: czytelność i przejrzystość programów, możliwość budowania bogatszych struktur do powtórzeń.

Użycie rekurencji jest wymuszone nie tylko brakiem większej liczby różnych instrukcji iteracyjnych, lecz także proponowanymi typami danych. Mowa tu jest o zmienych definicyjnych. To one bardzo często są odpowiedzialne za konieczność wykorzystania rekurencji.

Rozważmy przykład prostej procedury wczytującej dane z pliku z wykorzystaniem funkcji **fscanf**(fp↓,format↓,args↑,status↑) o następujących parametrach formalnych:

- fp – wskaźnik pliku, z którego pobierane będą dane,
- format – format wprowadzanych danych,
- args – zmienna definicyjna przechowująca wczytywaną wartość,
- status – zmienna definicyjna, informująca o przebiegu wykonywanej operacji.

Przykład 1 zawiera fragment kodu procedury napisanej za pomocą instrukcji iteracyjnej.

Przykład 1

```
odczyt(fname,n)
FILE fname;
{
  stdio:fopen(fname,"r",fp,status),
  {; i over 0 .. n :: fscan(fp,"%d",{buf},status1)}
}
```

Próba jej uruchomienia zakończy się błędem już w drugim przejściu pętli ze względu na fakt, że zmienna *buf* wykorzystywana w funkcji **fscan** jest zmienną definicyjną, która (jak wspomniano wcześniej) może mieć przypisaną wartość tylko raz. Poniżej zamieszczono ten sam fragment procedury, w którym do odczytu danych z pliku tym razem wykorzystano rekurencję.

Przykład 2

```
odczyt(fname,a,n)
FILE fp;
int a[];
{
 stdio:fopen(fname,"r",fp,status),
 odczyt1(fp,a,0,n),
}

odczyt1(fp,a,i,n)
FILE fp;
int a[];
{
 stdio:fscanf(fp,"%d",{buf},status1),
 {?     i == n -> stdio:fclose(fp,status2),
  default -> {? data(buf) -> { a[i] := buf,
                 odczyt1(fp,a,i+1,n)}
                 default -> stdio:printf("brak danych \n",{},_)}
 }
}
```

W przytoczonym wyżej fragmencie wartości z pliku pobierane są w kolejnych rekurencyjnych wywołaniach procedury *odczyt1* do zmiennej definicyjnej *buf*, a następnie wpisywane w kolejne elementy tablicy *a*. Każde wywołanie procedury *odczyt1* generuje nowy parametr dla funkcji **fscanf**, stąd nie występuje błąd próby podwójnego przypisania wartości zmiennej definicyjnej.

Problem ten może być rozwiązany także za pomocą *strumieni*, mechanizmu pozwalającego na przechowywanie wielu danych w jednej zmiennej definicyjnej. W systemie PCN strumienie implementowane są jako struktury listowe o postaci $x = [y|z]$. Tak przedstawionej zmiennej x możemy przypisać wartość zmiennej y oraz zmiennej z. Jeśli $y = 5$ i $z = 6$, to $x = [5,6]$. Proces taki może być powtarzany dowolną ilość razy (tzn. $z = [z1|z2]$ itd). Pusta lista ([]) zamyka strumień. Dostęp do dowolnego elementu listy jest sekwencyjny i uzyskiwany za pomocą operacji *dopasowania* (operator ?=). Po wykonaniu operacji

$$x\ ?= [dana|str]$$

zmienna *dana* przyjmie wartość y (z przykładu powyżej) i powstanie nowy strumień *str*, który w podobny sposób może być dekomponowany na elementy składowe.

Poniżej zamieszczono procedurę odczytu danych z pliku z przykładu 2 z wykorzystaniem strumieni.

Przykład 3

```
odczyt(fname,a,n)
FILE fp;
{
 stdio:fopen(fname,"r",fp,status),
 odczyt1(fp,strumien,0,n),
}

odczyt1(fp,strumien,i,n)
FILE fp;
{
  stdio:fscanf(fp,"%d",{buf},status1),
  {? i == n -> { stdio:fclose(fp,status2),
                 strumien = []},
     default -> {? data(buf) -> { strumien = [buf|strumien1],
                                  odczyt1(fp,strumien1,i+1,n)},
                 default -> stdio:printf("brak danych \n",{},_)}
  }
}
```

Poważną wadą prezentowanej konstrukcji jest sekwencyjny dostęp do poszczególnych elementów danych, co w efekcie może wprowadzić opóźnienia czasowe wykonania programu. Jednak ograniczenie, jakie nakłada PCN na wykorzystanie i modyfikację zmiennych wielokrotnych przez procesy uruchomione na różnych wierzchołkach, powoduje, iż staje się ono w wielu przypadkach jedynym rozwiązaniem.

2.1. Porównanie efektywności wykorzystania iteracji i rekurencji

Analizując zagadnienie rekurencji i iteracji należy również zwrócić uwagę, iż w środowisku PCN obliczenia równoległe wykonywane są przez działający współbieżnie zbiór lekkich procesów, a nie (jak to jest np. w języku C) przez sekwencyjne wywołanie procedur operujących na stosie. Fakt ten sprawia, iż rekurencja nie tylko nie spowalnia (jak zwykło się uważać) ale wprost przyspiesza działanie programów.

W celu potwierdzenia tej tezy przeprowadzono eksperymenty, w których badano czas działania procedur zaimplementowanych iteracyjnie i rekurencyjnie. Do analizy wykorzystano problem obliczania iloczynu skalarnego wektorów, mnożenia macierzy

i wyznaczania wartości silni. W miejscu tym należy wyraźnie podkreślić, że przeprowadzone badania nie miały na celu znalezienia efektywnych metod rozwiązania postawionych zadań. Efektywne programy numerycznej algebry liniowej rozwijane są od lat i przedstawione między innymi dla przetwarzania sewkwencyjnego w [3] oraz dla obliczeń równoległych w [4]. Wszystkie eksperymenty prowadzone były na komputerze SUN SPARCstation Classic o następujących parametrach: RAM 32MB, Clock 50 MHz, OS SunOS v4.1.3.

Poniżej przedstawiono fragmenty programów mnożących macierze:

– z wykorzystaniem iteracji

```
mno(i,rozmmac,A,B,C)
int A[], B[], C[], AB, pom1, pom2,pom3;
{ i over 0..rozmmac-1 ::
    { j over 0..rozmmac-1 ::
        { AB:=0,
        { k over 0..rozmmac-1 ::
            { pom1 := i*rozmmac+k,
              pom2 := k*rozmmac+j,
              AB := AB+A[pom1]*B[pom2]
            }
        },
          pom3 := i*rozmmac+j,
          C[pom3] := AB
        }
    }
}
```

– z wykorzystaniem rekurencji

```
mnozenie_k(k,A,B,x,rozmmac,i,j)
int A[], B[];
{? k < rozmmac -> { mnozenie_k(k+1,A,B,x1,rozmmac,i,j),
            x = x1+(A[i*rozmmac+k]*B[k*rozmmac+j])},
  default -> x = 0
}

mnozenie_j(j,A,B,y,rozmmac,i)
int A[], B[];
{? j < rozmmac -> { mnozenie_k(0,A,B,x,rozmmac,i,j),
            y = [x|y1],
            mnozenie_j(j+1,A,B,y1,rozmmac,i)
          },
```

```
  default -> y = []
}
mnozenie_i(i,A,B,z,rozmmac)
int A[], B[];
{? i < rozmmac -> { mnozenie_j(0,A,B,y,rozmmac,i),
                    z = [y|z1],
                    mnozenie_i(i+1,A,B,z1,rozmmac)
                  },
  default -> z = []
}
```

W przykładzie pierwszym do zapamiętania wartości macierzy wynikowej wykorzystano tablicę. Takie rozwiązanie może być zastosowane wówczas, gdy program będzie uruchamiany tylko na jednym komputerze. W drugim przypadku posłużono się strumieniem, stwarzając w ten sposób możliwość równoległego wykonywania na kilku wierzchołkach.

Tabela 1

Porównanie czasów mnożenia macierzy z wykorzystaniem iteracji i rekurencji

Rozmiar macierzy	Czas mnożenia macierzy (iteracyjnie) [s]	Czas mnożenia macierzy (rekurencyjnie) [s]
10*10	0.7998	0.2741
20*20	5.6431	1.6675
30*30	19.1132	5.5859
40*40	45.7583	12.8253
50*50	89.8721	25.0485
100*100	731.3854	196.0969

Tabela 1 przedstawia wyniki będące średnią arytmetyczną czasów uzyskanych w wielu próbach. Kolumna druga i trzecia zawierają odpowiednio czasy mnożenia macierzy danej wielkości przez procedury zapisane iteracyjnie i rekurencyjnie. Na ich podstawie można określić efektywność obu metod. Analiza wyników pozwala stwierdzić, iż stosowanie iteracji wydłuża ponad 3-krotnie czas wykonania programu. Podobne rezultaty, świadczące na korzyść rekurencji, otrzymano testując pozostałe wymienione wcześniej algorytmy. Oczywiście nie każde zagadnienie da się przedstawić w sposób rekurencyjny, jeśli jednak taka możliwość istnieje, warto z niej skorzystać.

3. Możliwości systemu PCN w zakresie wykorzystania procedur języka C

Jak wspomniano we wstępie, w systemie PCN istnieje możliwość dołączania procedur napisanych w innych językach, np. C lub Fortran. Istnieją jednak pewne różnice między wykonaniem procedury PCN i procedur "obcych", wywoływanych z PCN. Pierwsze z nich mogą być wykonywane nawet wówczas, gdy nie wszystkie zmienne definicyjne zostały zainicjowane (np. procedura może obliczać wartość argumentu definicyjnego), natomiast wywołanie procedury "obcej" jest wstrzymywane do czasu ich określenia. Konsekwencją tego jest fakt, że wszystkie parametry wyjściowe w procedurach "obcych" muszą być zmiennymi wielokrotnymi. Drugą ważną różnicą jest to, że ich parametry aktualne mogą być jedynie zmiennymi prostymi typu: znakowego, całkowitego, podwójnej precyzji lub tablicami tych typów, nie mogą natomiast być krotkami. Ponieważ argumentami procedur są wskaźniki do zmiennych, w języku C powinny one być zadeklarowane jako typy wskaźnikowe, tzn. typom char, int, double systemu PCN powinny odpowiadać typy char *, int *, double * języka C.

3.1. Opis przeprowadzonych eksperymentów i uzyskane wyniki

Analiza możliwości i efektywności wykorzystania procedur języka C w systemie PCN oparta została na trzech problemach. Pierwszy z nich dotyczył wspomnianego w punkcie 2 odczytu danych z pliku. Przygotowane zostały trzy programy: jeden – zawierający procedurę napisaną za pomocą rekurencji, w efekcie działania której generowany jest strumień wczytanych wartości (przykład 3), drugi – analogiczną wypełniającą tablicę (przykład 2) i trzeci – wywołujący procedurę napisaną w języku C.

Eksperymenty prowadzone były dla tablic i strumieni o rozmiarach 100, 500, 1000, 5000, 10000, 25000 elementów i wielokrotnie powtarzane dla danej wielkości. Wartości zamieszczone w tabeli 2 są średnią arytmetyczną uzyskiwanych rezultatów. Pierwsza kolumna wyników zawiera czasy realizacji zadania odczytu danych z pliku przez procedurę zapisaną w języku PCN, gdy miejscem ich przeznaczenia była tablica. W drugiej znajdują się czasy odczytu danych do strumienia. Wreszcie w trzeciej kolumnie umieszczono czasy wykonania tego zadania przez procedurę zaimplementowaną w języku C.

Do dalszych badań problem ten został rozszerzony o posortowanie wypełnionych wcześniej struktur: strumienia i tablicy. Zostały one podzielone w tym celu na 5 części, które porządkowane były we wzajemnie równoległych procesach. Tak posortowane ciągi podlegały łączeniu, aż do uporządkowania wszystkich danych. Ponieważ w badaniach

Tabela 2

Porównanie czasów odczytu danych z pliku realizowanych za pomocą procedur zapisanych w języku PCN oraz C

Rozmiar tablicy i strumienia	Odczyt [s]		
	PCN		C
	tablice	strumienie	
100	0.173131	0.176513	0.008212
500	0.845233	0.889212	0.022113
1000	1.793167	1.867223	0.038521
5000	8.155513	15.309833	0.175512
10000	15.790331	27.344352	0.344433
25000	39.204813	68.954166	0.859534

skupiono się na operacji sortowania, część dotycząca scalania nie została uwzględniona w testach. Wyniki zawarte w tabeli 3 są uśrednionymi czasami realizacji omawianego zadania. Podobnie jak w przypadku tabeli 2, pierwsze dwie kolumny zawierają czasy sortowania jednego podciągu, będącego odpowiednio częścią tablicy i strumienia (implementacja w języku PCN), w trzeciej natomiast znajdują się wyniki realizacji tego zadania przez procedurę języka C. Pomiary czasowe wykonywane były za pomocą

Tabela 3

Porównanie czasów sortowania realizowanych za pomocą procedur zapisanych w języku PCN oraz C

Rozmiar tablicy i strumienia	Sortowanie [s]		
	PCN		C
	tablice	strumienie	
100	0.062166	0.033911	0.000101
500	0.441334	0.296301	0.001130
1000	1.180667	0.841902	0.002267
5000	7.454167	7.424261	0.014830
10000	18.148156	22.275801	0.034451
25000	66.108101	171.056766	0.117230

funkcji **ftime** (dostępnej w języku C). Natomiast wszystkie wykorzystywane procedury wywoływane były z programu głównego zaimplementowanego w języku PCN. Należą do nich procedury sortujące dołączone do dokumentacji systemu PCN oraz procedura **qsort** dostępna w języku C. Treści ich zamieszczone zostały w Dodatku.

Ostatni z analizowanych problemów dotyczył omawianego w punkcie 2.1 mnożenia macierzy. Aby nie powtarzać wyników uzyskiwanych przy porównaniu efektywności działania rekurencji i iteracji, poniżej przedstawione zostaną średnie arytmetyczne czasów wykonania programu uzyskiwanych przy wykorzystaniu procedury języka C dla analogicznych rozmiarów macierzy:

– 10*10 – 0.05648 s,
– 20*20 – 0.12056 s,
– 30*30 – 0.23351 s,
– 40*40 – 0.43893 s,
– 50*50 – 0.70692 s,
– 100*100 – 5.42734 s.

3.2. Analiza wyników

Wartości zamieszczone w tabeli 2 i 3 już na pierwszy rzut oka wykazują przewagę w szybkości działania procedur zapisanych w języku C nad procedurami stworzonymi w PCN. To, co w języku C wykonywane jest w przeciągu niecałej sekundy, PCN realizuje w ciągu kilkunastu, a nawet kilkudziesięciu sekund.

Przypatrzmy się wynikom zawartym w tabeli 2 (odczyt danych z pliku). Jeśli zbadamy stosunek czasu wykonania tego zadania przez procedurę napisaną w C do czasu jego realizacji w języku PCN, okaże się, że ten ostatni do pobrania tej samej liczby danych z pliku potrzebuje ok 4500% czasu więcej (a nawet ponad 8000% w przypadku strumieni 25 000–elementowych) niż procedura "obca". Najmniejszą różnicę, wynoszącą około 2000%, można zauważyć dla 100-elementowych struktur danych.

Na bazie otrzymanych wyników przeprowadzono także analizę wpływu zwiększania liczby danych na czas wykonania badanych procedur. Porównując przyrost czasu w stosunku do przyrostu ilości elementów można powiedzieć, że we wszystkich przypadkach wydłużenie działania programu jest wprost proporcjonalne do zmian w ilości wprowadzanych elementów.

Trochę inaczej wygląda sytuacja w przypadku sortowania. Tutaj różnice czasowe sięgają ok 30 000% dla strumieni o małej liczbie elementów (100, 500, 1000), 50 000% dla większości tablic oraz strumieni składających się z 10 000 danych i aż 145 000% dla strumieni 25 000–elementowych. Ciekawym zjawiskiem jest fakt, że dla mniejszej liczby

elementów sortowanie strumieni przebiega szybciej niż porządkowanie tablic (mowa o języku PCN). Czasy wyrównują się przy 5000 elementach, a dopiero później sytuacja się odwraca.

Analizując zmiany zachodzące w czasie wykonania procedur, gdy zmienia się liczba wykorzystywanych danych, stwierdzono, że w przypadku zwiększania rozmiaru sortowanych tablic w obu językach wydłużenie działania programu jest 0.5 razy większe niż zmiany w ilości danych. W przypadku strumieni stosunek czasów jest większy od odpowiadających mu przyrostów liczby elementów niecałe 2 razy.

Rezultaty uzyskiwane w przypadku mnożenia macierzy w pełni potwierdzają efektywność wykorzystania języka C w systemie PCN. Przyspieszenie działania programu (wynikające z wykorzystania procedury "obcej") wzrasta wraz z ilością przetwarzanych danych w przedziale od 1300% do 3600% w stosunku do czasu jego rekurencyjnego wykonania.

3.3. Wnioski

Podsumowując analizę otrzymanych wyników można stwierdzić, że zastosowanie języka C w środowisku PCN daje bardzo dobre rezultaty czasowe i pozwala zaoszczędzić wiele cennych sekund, o które z taką usilnością walczy się w ostatnich czasach. Przeprowadzone testy wykazały, że system PCN, dostarczający konstrukcje pozwalające na współbieżne wykonywanie programu, może być z powodzeniem łączony z językiem C. Rola wspomnianego systemu widziana jest tu w nadzorowaniu wykonania programu, rozsyłaniu jego poszczególnych części do równoległego wykonania na jednym lub kilku wierzchołkach i gromadzeniu wyników. Język C może być natomiast wykorzystywany do zaimplementowania algorytmu rozwiązania problemu przeznaczonego do realizacji przez współbieżne procesy. Dodatkową korzyścią takiego rozwiązania jest możliwość łatwego przystosowania już istniejącego oprogramowania do równoległej pracy. Wysiłek programistów może być skierowany na wybór niezależnych od siebie fragmentów zadania, które realizowane współbieżnie mogą dać w efekcie krótszy czas wykonania programu. Nie bez znaczenia jest też przyzwyczajenie i umiejętności informatyków, którzy dobrze zaznajomieni z językiem C mogą skupić się na rozwiązywaniu problemu, a nie na nauce nowego języka, jakim jest PCN.

4. Dodatek

Zestaw procedur wykorzystywanych w eksperymentach dotyczących efektywności wykorzystania procedur języka C w systemie PCN.

– **qsort** operujący na strumieniach:

```
qsort(Ns,Lb,Le)
{? Ns ?= [N|Ns1] -> {|| part(N,Ns1,L,R), qsort(L,Lb,[N|Lm]), qsort(R,Lm,Le)}
  Ns ?= [] -> Lb=Le
}

part(X,Y,L,R)
{? Y ?= [N|Ys], X>N -> {|| L=[N|Ls], part(X,Ys,Ls,R)},
  Y ?= [N|Ys], X<=N -> {|| R=[N|Rs], part(X,Y,L,Rs)},
  Y ?= [] -> {|| L=[], R=[]}
}
```

– **qsort** operujący na tablicy:

```
qsort(lb,ub,a)
int a[],i,j;
{? ub>lb -> {; s=a[lb], i:=lb, j:=ub,
          partition(lb,ub,i,j,s,a),
          i:=lb, swap(i,j,a),
          {|| qsort(lb,j-1,a), qsort(j+1,ub,a)}
          }
}
partition(lb,ub,i,j,s,a)
int i,j,a[];
{? i<j -> {;{|| movej(lb,j,s,a), movei(ub,i,s,a)},
          {? i<j -> swap(i,j,a)},
          partition(lb,ub,i,j,s,a)
        }
}

swap(i,j,a)
int i,j,a[];
{; tmp=a[i], a[i]:=a[j], a[j]:=tmp}

movej(lb,j,s,a)
int j,a[];
{? j>=lb, a[j]>s ->{; j:=j-1, movej(lb,j,s,a)}}
```

```
movei(ub,i,s,a)
int i,a[];
{? i<=ub, a[i]<=s -> {; i:=i+1, movei(ub,i,s,a)}}
```

– procedura mnożąca macierze zapisana w języku C:

```
void mno(rozmmmac,A,B,C)
int A[], B[], C[];
int *rozmmac;
{int w, k, j, wskA, wskB, wskC, elewyn;
int rozm1 =*rozmmac;

for (w = 0; w < rozm1; w++)
  for (k = 0; k < rozm1; k++)
    { elewyn = 0;
      for (j = 0; j < rozm1; j++)
        { wskA = j+w*rozm1;
          wskB = k+j*wsk1;
          elewyn += A[wskA]*B[wskB];
        }
      wskC = w*rozm1+k;
      C[wskC] = elewyn;
    }
}
```

LITERATURA

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

[2] Czajkowski G., Zieliński K.: PCN system rozproszonego przetwarzania, Informatyka, nr 5, 1994.

[3] Anderson E., Bai Z., Bischof C.H., Demmel J., Dongarra J.J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Ostrouchov S., Sorensen D.C.: LAPACK Users' Guide Release 2.0, SIAM, Philadelphia, 1995.

[4] Choi J., Demmel J., Dhillon I., Dongarra J., Ostrouchov S., Petitet A., Stanley K., Walker D., Whaley R.C.: ScaLAPACK: a portable linear algebra library for distributed memory computers – design issues and performance, LAPACK Working Note 95.

Recenzent: Dr inż. Marian Bubak

Wpłynęło do Redakcji 19 kwietnia 1995 r.

Abstract

This paper contains a comparison of iteration and recursion efficiency in PCN (a parallel programming system). Chapter 1 includes a short description of PCN language. The execution times of the matrix multiplication using two methods mentioned above are presented in the table 1 in chapter 2. The best results were obtained (by PCN) using recursion.

A possibility of joining sequential procedures written in C with a PCN aplications is presented in chapter 3. Results of performed tests are shown in table 2 and 3. These experiments proved that it is better (if possible) to use C-procedures.

This paper also presents the influence of the data structures used on task performance speed.

Text of used procedures (written in C and PCN language) are included in chapter Dodatek.