CD-ROM: Borland InterBase 6, KDE 2.0Beta3, Acidgimp 0.5, GNOME 1.2

9/2000

magazyn

LINUX & UNIX

Caldera przejmuje SCO!

Zestawienie najpopularniejszych routerów

Sendmail dla każdego

Język PHP

Generator skanerów Flex

Podłączanie do sieci IPv6

Borland InterBase 6

NR 9(24)/2000 WRZESIEŃ CENA 17 ZŁ ISSN 1506-1299

CD ROM

Adres kontaktowy: magazyn@tao.com.pl

Biznes 9

Nowy algorytm kompresji umożliwia rozpowszechnienie aplikacji 3D w Internecie 14

Baza danych RIPE 16
Wojciech Bojdoł

Bezpieczeństwo - Aktualności 17

Produkty

Zestawienie najpopularniejszych routerów na polskim rynku 18

Trawers - rozwiązania dla firm wielooddziałowych 23
Computer-bis

IFS - internetowy system plików 27

Nowe cechy dostępne w InterBase 6.0 28
BSC Polska

Programowanie

Generator skanerów Flex 31
Grzegorz Jacek Nalepa

Firma TAO Systems Polska, wydawca miesięcznika Linux & UNIX poszukuje osób do testowania i opisywania programów działających w systemie Linux.
Zainteresowane osoby prosimy o kontakt z Redaktorem Naczelnym
- Marcinem Koprowskim.
tel. 0502 02 98 47, e-mail: marcin.koprowski@tao.com.pl

Adres kontaktowy: magazyn@tao.com.pl

ORBIT: czyli „Kobra" w C bez cukru 34
Marek Sawerwain

High Performance Fortran 38
Piotr Koch

Technologie Sieciowe

Podłączanie do sieci IPv6 45
Wojciech Bojdoł

Język PHP 47
Wojciech Bojdoł

Technologia WAP - język WML i WMLScript 35
Wojciech Bojdoł

Linux dla każdego

Sendmali dla każdego - część II 50
Maciej Broniarz

Wdrożenia

Migracja bazy Baan do Oracle 51

Zapraszamy na naszą stronę www
www.magazyn.tao.com.pl

magazyn Linux & UNIX

High Performance Fortran

Piotr Koch

Od momentu swojego powstania, ponad trzy dekady temu, Fortran był językiem programowania wysokiego poziomu komputerów sekwencyjnych znajdującym głównie zastosowanie w dziedzinach naukowych. Z tych względów posiada obecnie znaczne wsparcie ze strony środowisk naukowych oraz ogromną ilość istniejących aplikacji. Jakkolwiek zwykły Fortran 77 oraz Fortran 90 nie wykorzystują w pełni możliwości architektur współczesnych komputerów, które coraz częściej wymagają od programów informacji, m.in. o możliwości równoległego wykonania zadań, rodzaju dostępnej równoległości (SIMD, MIMD) czy rozdzieleniu danych do pamięci poszczególnych procesorów. Aby sprostać tym wymaganiom, koalicja akademickich i przemysłowych grup utworzyła pod kierunkiem Kena Kenndy'ego High Performance Fortran Forum (HPFF), którego celem było rozwinięcie zbioru standardowych rozszerzeń dla języka Fortran, wspierających tworzenie wysoko wydajnego oprogramowania dla wielu platform. W oparciu o propozycje firm COMPASS, DEC, Cray Research oraz uniwersytetów Rice, Syracuse i Vienna, stworzono w 1992 roku standard nowego języka nazwany High Performance Fortran (HPF), który zawiera:

- konstrukcje języka Fortran 90 operujące na tablicach;
- konstrukcje do opisu pętli równoległych;
- konstrukcje do określenia złożonych zadań równoległych;
- konstrukcje do określenia efektywnego przydziału i rozmieszczenia danych.

Utworzony w wyniku prac HPFF standard języka cechuje się znaczną przenośnością między stacjami roboczymi a ogromnymi superkomputerami równoległymi. Z dostępnych informacji wynika, że High Performance Fortran jest obecnie wspierany przez platformy: Linux, IBM RS/6000, SUN/Sparc, HP 9000/700, SGI Irix, DEC Alpha oraz Windows NT.

Pomimo, że HPF wprowadza stosunkowo niewielki zbiór rozszerzeń do notacji języka Fortran 90, to mimo wszystko jest złożonym językiem. Rozszerzenia HPF cechują się znaczną liczbą różnych wariantów i współdziałają z konstrukcjami Fortranu 90 na wiele rozmaitych sposobów. Stąd dla potrzeb przedstawionej analizy zostaną rozpatrzone jedynie zasadnicze elementy równoległości języka HPF.

Podział

Podstawę HPF stanowi Fortran 90 powiększony o dyrektywy rozmieszczenia danych, wyrażenie FORALL, dyrektywę INDEPENDENT oraz nowe instrukcje operacji na tablicach. Z kolei Fortran 90 wprowadza do Fortranu 77 wskaźniki, dynamiczną alokację pamięci, definiowane przez użytkownika typy danych, operacje na tablicach i inne cechy. Instrukcje operacji na tablicach dostarczają wygodnej i zwięzłej notacji do określenia zadań równoległości danych. Pomimo tego znajdują zastosowanie jedynie w ograniczonym zbiorze takich operacji. Dla przykładu wymagają, aby operandy znajdujące się po prawej stronie wyrażenia miały ten sam kształt, co tablica po jego lewej stronie. Wprowadzane przez HPF dwie dodatkowe konstrukcje – wyrażenie FORALL i dyrektywa INDEPENDENT- umożliwiają jawne przedstawienie szerszego zakresu operacji równoległości danych.

Główną przeszkodą wpływającą na efektywność równoległej realizacji programu HPF stanowią sekwencyjne wąskie gardła. Oznacza to, że jeżeli fragment kodu nie wprowadza dostatecznej równoległości lub gdy równoległość istnieje, lecz kompilator nie jest w stanie jej wykryć, wówczas taki fragment kodu zostanie wykonany sekwencyjnie. Sekwencyjne wąskie gardła nie są zbyt istotne, gdy program będzie wykonany na architekturze z niewielką liczbą procesorów, lecz oczywiście ograniczą skalowalność realizacji problemu.

Operacje na tablicach

Zależnie od użytego języka programowania, grupy danych, na których operuje program, mogą być regularne bądź nieregularne. Instrukcje języka HPF operują na regularnych strukturach danych, a konkretnie na tablicach. Wyrażenia przypisania wartości do elementów tablic są w HPF jawnymi konstrukcjami równoległymi, przy czym sposób zrównoleglenia jednoczesnych operacji na elementach tablic pozostaje wyłącznym zadaniem kompilatora, odciążając w tym programistę.

HPF umożliwia stosowanie rozmaitych operacji nie tylko w stosunku do wartości skalarnych, ale również do całych tablic danych. Ta cecha języka sprawia, że przeprowadzane operacje odnoszą się do wszystkich elementów tablic uczestniczących w operacji. Jeżeli operacja przeprowadzana jest jednocześnie na wielu korespondujących elementach każdej ze struktur, wówczas uczestniczące w operacji tablice muszą mieć taki sam kształt i rozmiar. Na przykład można zastosować operację dodawania dla uczestniczących w niej tablic A i B oraz skalaru c w następujący sposób:

```
integer A(10,10), B(10,10)
A= B + C
```

Powyższe wyrażenie przypisuje każdemu elementowi A(i, j) tablicy A sumę wartości B(i, j) + c. W zasadzie wszystkie podstawowe operacje języka HPF mogą być stosowane do tablic. Poniżej zaprezentowano kilka wybranych przykładów:

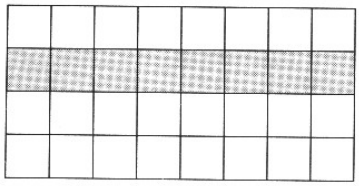
```
real  A(10,20), B(10,20)
logical     L(10,20)

A = A + 1.0     !dodaje wartość 1.0 do
                każdego elementu tablicy A
A = SQRT(A)     !oblicza kwadrat każdego
                elementu tablicy A
L = A.EQ.B      !ustawia element L(i, j)
                na .true. jeżeli A(i,
                j)=B(i, J)
                ! oraz na .false. w wypa-
                dku przeciwnym
```

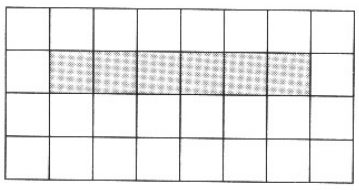
Niektóre zadania wymagają wykonania operacji jedynie dla pewnej części z całej tablicy danych. Do określenia zakresu indeksów elementów uczestniczących w operacji HPF dostarcza mechanizmu zwanego striding, to znaczy indeksowania z pewnym krokiem. Zakres indeksowanych elementów przedstawia trójskładnikowe wyrażenie o ogólnej postaci:

Start:end:stride

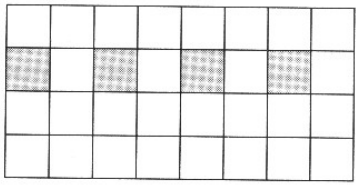
a) A (2, :)



A (2, 2:7)



A (2:1, 1:8:2)



Rysunek 1. Przykłady indeksowania z krokiem.

Pominięcie składnika stride powoduje jego domyślne ustawienie na wartość 1, natomiast pominięcie granic zakresu indeksowania powoduje domyślne wykonanie operacji w całym zakresie indeksów dla każdej z tablic.

Dodatkowe użycie konstrukcji where jako maski operacji przypisania umożliwia ograniczenie liczby elementów tablicy, dla których przypisanie jest wykonywane. Dla przykładu, poniższe wyrażenie zastępuje każdy niezerowy element tablicy X odwrotnością jego wartości:

```
where(X /= 0) X = 1.0/X
```

Jednocześnie HPF dostarcza szeregu funkcji znajdujących zastosowanie tak w przypadku skalarów, jak i realizujących działania z osobna na każdym elemencie tablicy. Dla przykładu, wynikiem działania funkcji ABS(A) jest tablica składająca się z absolutnych wartości elementów tablicy A. Dodatkowo HPF dostarcza znaczną liczbę funkcji dokonujących określonych transformacji elementów tablicy, których wynikiem działania jest przekształcona tablica o wartościach zależnych od wielu elementów transformowanej tablicy. Poniższa tabela przedstawia kilka wybranych przykładów takich funkcji:

Funkcja	Wartość zwracana
MAXVAL(A)	Maksymalna wartość A
MINVAL(A)	Minimalna wartość A
SUM(A)	Suma elementów A
PRODUCT(A)	Produkt elementów A
MAXLOC(ARRAY)	Indeksy maksymalnych wartości w A
MINLOC(ARRAY)	Indeksy minimalnych wartości w A
MATMUL(A, B)	Mnożenie macierzy A*B
TRANSPOSE(A)	Transponowanie A
CSHIFT(A, SHIFT, DIM)	Przesuwanie elementów A

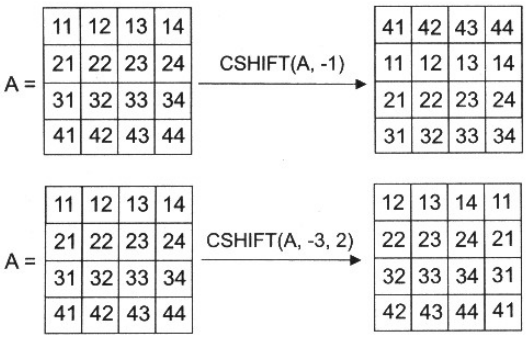
Funkcje MAXVAL, MINVAL, SUM i PRODUCT wykonują operacje redukcji na tablicy, zwracając wartość skalarną, reprezentującą odpowiednio maksimum, minimum, sumę i produkt elementów tablicy.

Istotną funkcją HPF, przekształcającą tablicę, jest CSHIFT. CSHIFT dokonuje przesunięcia typu circular wszystkich elementów. Wynikiem działania CSHIFT jest nowa tablica o tym samym rozmiarze i kształcie, lecz o innej konfiguracji elementów. Przedstawione poniżej przykłady działania funkcji CSHIFT(A, s, d) dokonują przesunięcia elementów tablicy A, przy czym tablica bądź skalar s określa rozmiar i kierunek przesunięcia, a opcjonalny argument d wskazuje wymiar, w którym przesunięcie ma być wykonane (patrz Rysunek 2).

Pomimo swoich ogromnych możliwości, operacje na tablicach języka mogą zostać użyte do implementacji jedynie ograniczonej klasy algorytmów równoległości danych. Konsekwentnie, programy HPF posiadają kod, który pomimo swojej potencjalnej równoległości jest wykonywany sekwencyjnie, ponieważ kompilator nie jest w stanie tej równoległości wykryć. Dla przykładu, poniższy fragment kodu zerujący diagonalną tablicy X:

```
do i = 1, 100
  X(i, i) = 0.0
enddo
```

pomimo, że jest oczywistą operacją równoległą nie



Rysunek 2: Działanie funkcji CSHIFT.

może zostać opisany przy użyciu wyrażeń dostarczanych przez język HPF do operacji na tablicach.

Wyrażenie FORALL

Wyrażenie FORALL umożliwia stosowanie bardziej ogólnych przypisań wartości do elementów tablicy. FORALL przyjmuje ogólną postać:

```
FORALL(start:end:stride, ...., start:end:stri-
de, mask) assignment
```

gdzie `assignment` jest arytmetyczną bądź wskaźnikową operacją przypisania.

Wyrażenie przypisania jest wykonywane dla tych wartości indeksów, które określa `start:end:stride` i jednocześnie nie odrzuca opcjonalna maska `mask`. Dla przykładu, poniższe wyrażenia odpowiednio a) przypisują każdemu elementowi tablicy X wartość będącą sumą jego indeksów, b) zerują wartości elementów tablicy Y powyżej jej diagonalnej oraz c) zerują diagonalną tablicy Z.

```
a)  FORALL(i = 1:m, j = 1:n)     X(i, j) = i + j
b)  FORALL(i = 1:n, j = 1:n, i<j)  Y(i, j) = 0.0
c)  FORALL(i = 1:n)     Z(i, i) = 0.0
```

Wyrażenie FORALL dokonuje przypisania wartości do elementów tablicy w dowolnym porządku. Aby zatem mieć pewność, że wyrażenie zostanie wykonane w sposób deterministyczny, programista powinien zadbać, aby FORALL nie dokonywało operacji przypisania wartości do tego samego elementu tablicy więcej niż jeden raz. Wprawdzie kompilator może podjąć próbę wykrycia naruszenia tej zasady, lecz nie zawsze jest w stanie tego dokonać. Dlatego poniższe wyrażenie jest poprawne jedynie wówczas, gdy tablica indeksów `Index` nie zawiera identycznych wartości.

```
FORALL(i = 1:n)   A(Index(i)) = B(i)
```

Dyrektywa INDEPENDENT

Program HPF może wskazywać na dodatkową możliwość równoległej realizacji problemu przez użycie dyrektywy INDEPENDENT do określenia, że iteracje pętli `do-enddo` powinny zostać wykonane niezależnie (tzn. współbieżnie i w dowolnej kolejności) bez wpływu na poprawność wyniku obliczeń. W efekcie dyrektywa przekształca pętlę `do-enddo` w jawną konstrukcję równoległą.

Dyrektywa INDEPENDENT musi bezpośrednio poprzedzać pętlę `do-enddo`, do której się stosuje. W swojej najprostszej formie nie zawiera dodatkowych argumentów i wymaga, aby żadna operacja pętli `do-enddo` nie wpływała na pozostałe iteracje. Dlatego w poniższym fragmencie kodu tablica `Index` nie powinna zawierać identycznych wartości i jednocześnie tablice A oraz B nie powinny współdzielić jednego obszaru pamięci:

```
!HPF$ INDEPENDENT
do i=1,n
A(Index(i)) = B(i)
enddo
```

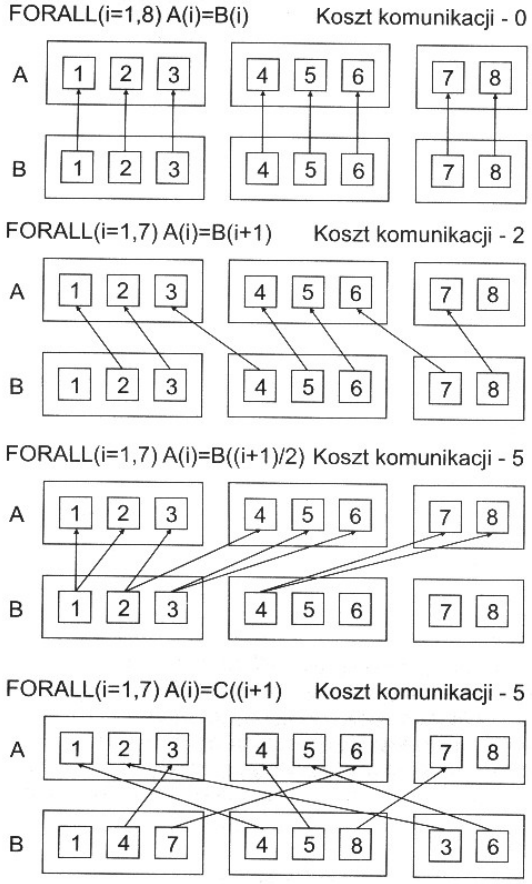
Dyrektywa INDEPENDENT wymaga użycia odrębnych przestrzeni adresowych do określenia zbioru zmiennych dla każdej iteracji zagnieżdżonej pętli `do-enddo`. W poniższym fragmencie kodu jednoczesne wykonanie różnych iteracji pętli mogłoby prowadzić do powstania błędnych wyników, gdyby wartości zmiennych `tmp1` i `tmp2` obliczane przez jedną z iteracji były używane w innej iteracji. Argument NEW dyrektywy INDEPENDENT zapewnia, że taka sytuacja nie powstanie.

```
!HPF$ INDEPENDENT
do i=1,n1
!HPF$       INDEPENDENT, NEW(tmp1,tmp2)
do j=1,n2
tmp1 = B(i, j) + C(i, j)
tmp1 = B(i, j) - C(i, j)
A(i, j) = tmp1 * tmp2
enddo
enddo
```

Komunikacja

Najbardziej atrakcyjną cechą języka HPF jest to, że kompilator przejmuje na siebie zadanie wygenerowania kodu komunikacji w programie. W przybliżeniu kompilację programu HPF można opisać następująco: wyrażenia dokonujące dekompozycji struktur danych są poddawane wstępnej analizie w celu określenia dekompozycji każdej tablicy w programie. Następnie podziałowi między dostępne procesory ulegają obliczenia, które niekiedy odnoszą się do nielokalnych (przydzielonych innym procesorom) struktur danych i stąd wymagają nawiązania komunikacji. Operacje komunikacji nie są jawnie określane przez programistę, lecz wnioskowane z programu przez kompilator, który również jest w stanie samodzielnie dokonać optymalizacji, w szczególności usiłując przenieść komunikację poza wszelkie pętle tak, aby zredukować jej koszty.

Określenie kosztu komunikacji programów HPF wymaga od programisty zwrócenia uwagi na następujące kwestie:



Rysunek 3: Koszty komunikacji dla różnych wyrażeń FORALL.

- wiele podstawowych instrukcji wykonujących operacje na tablicach łączy dane z całych tablic i przez to wymaga wprowadzenia komunikacji, jeżeli tablice, do których się stosują, zostały rozmieszczone do procesorów. Dla przykładu, operacje redukcji takie jak MAXVAL i SUM mogą zostać wykonane w liczbie logP kroków na P procesorach z całkowitym kosztem komunikacji $\log P(t_s + t_w)$. Ten koszt jest niezależny od rozmiaru tablic, na których wykonywana jest redukcja. Dla odmiany, koszt operacji TRANSPOSE lub MATMUL zależy zarówno od rozmiaru jak i rozmieszczenia argumentów operacji.
- wyrażenie FORALL może prowadzić do powstania komunikacji, gdy obliczenia pewnych elementów tablicy A(i) wymagają uzyskania wartości danych (np.: B(j)) z innego procesora. Poniższy przykład ilustruje powstanie kosztów komunikacji przy różnych wariantach wyrażenia FORALL dla trzech tablic A, B i C rozmieszczonych następująco:

```
!HPF$ PROCESSORS pr(3)
integer A(8), B(8), C(8)
!HPF$ ALIGN B(:) WITH A(:)
!HPF$ DISTRIBUTE A(BLOCK) ONTO pr
!HPF$ DISTRIBUTE C(CYCLIC) ONTO pr
```

Innym potencjalnym źródłem komunikacji jest operacja CSHIFT. Cykliczne rozmieszczenie danych zazwyczaj częściej prowadzi do powstania komunikacji niż rozmieszczenie blokowe, jednakże z drugiej strony rozmieszczenie cykliczne skutkuje lepszym wyrównywaniem obciążenia w niektórych aplikacjach.

Nawet proste operacje wykonywane na rozmieszczonych inaczej (nie wyrównanych) tablicach mogą prowadzić do powstania komunikacji. Dla przykładu, przypisanie A=B może wymagać znacznej komunikacji, jeżeli tablice A i B zostały inaczej rozmieszczone. Programista powinien wówczas rozważyć możliwość restrukturyzacji rozmieszczenia obu tablic do jednakowej postaci oraz ocenić powstały przy tym koszt w stosunku do pierwotnego kosztu komunikacji.

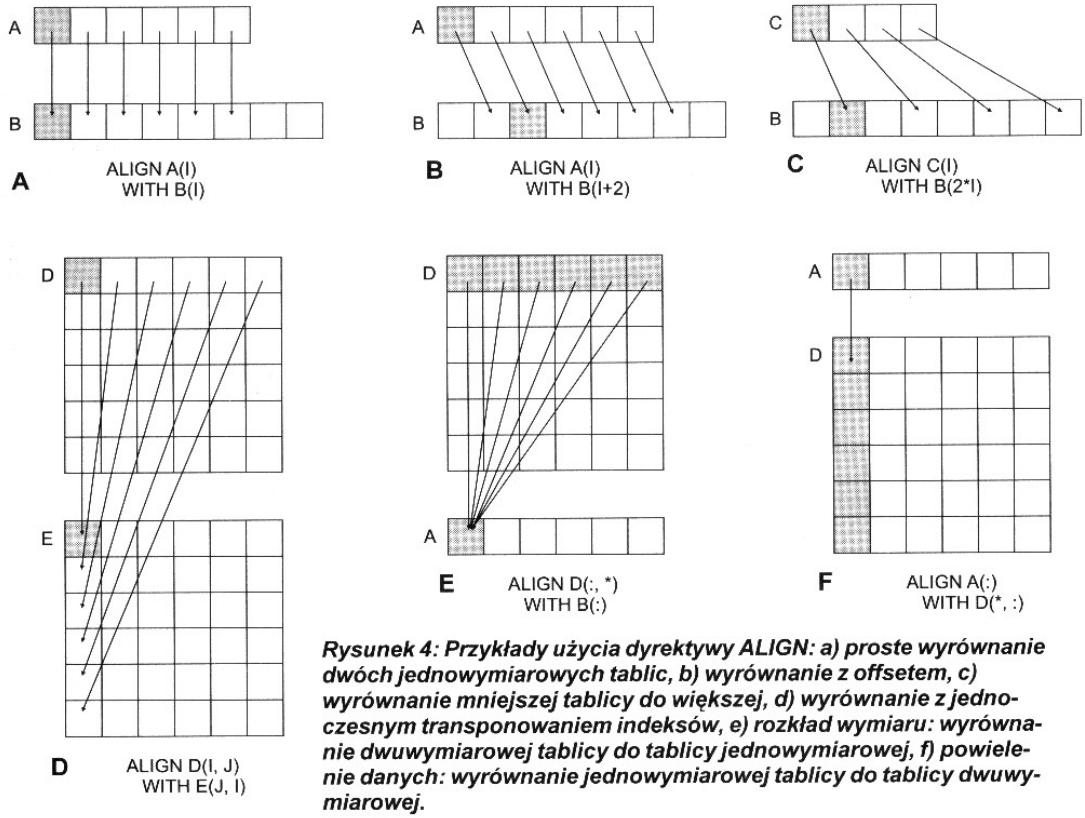
Zmiana dekompozycji tablic na granicach procedur może skutkować powstaniem znacznej komunikacji. Chociaż dokładne oszacowanie komunikacji zależy od sposobu dekompozycji, to często całkowity koszt komunikacji dla wszystkich P procesorów przy zmianie dekompozycji tablicy MxN będzie w przybliżeniu równy $t_sP^2+t_wSMN$, gdzie S jest rozmiarem pojedynczego elementu tablicy wyrażonym w czterobajtowym słowie. Koszt ten powstaje, ponieważ każdy z P procesorów musi komunikować się ze wszystkimi pozostałymi procesorami.

Dobry kompilator HPF nie kompiluje programu wyrażenie po wyrażeniu, natomiast redukuje koszty komunikacji przez łączenie operacji komunikacji czy inną reorganizację wyrażeń programu. W dodatku kompilator może wybrać odmienny sposób rozmieszczenia danych niż zalecany przez programistę.

Aglomeracja

Operacje na tablicach języka HPF umożliwiają równoległą realizację problemu, lecz nie wskazują w jaki sposób należy minimalizować koszty komunikacji podczas wykonania programu na konkretnej platformie równoległej.

Dyrektywy rozmieszczenia danych są zasadniczą częścią algorytmu HFP, ponieważ planowanie danych do procesorów określa lokalność obliczeń a przez to w znacznym stopniu wydajność programu równoległego. Dla przykładu, prosta operacja przypisania A = B*C może zostać wykonana bez jakiejkolwiek komunikacji, bądź wymaga nawiązania komunikacji dla każdej operacji przypisania zależnie od tego, czy korespondujące elementy tablic A, B i C są rozmieszczone na tych samych, czy na różnych procesorach. Określenie najlepszego sposobu rozmieszczenia rozmaitych struktur danych, na których operuje program HPF jest ogólnym i niełatwym problemem optymalizacji. Z tych względów język HPF umożliwia programiście określenie w jaki sposób struktury danych powinny zostać rozmieszczone na dostępnych procesorach. Co ważne, dyrektywy rozmieszczenia danych mają wpływ wyłącznie na wydajność programu, a nie na wynik obliczeń. Warto również zaznaczyć, że dyrektywy są jedynie zaleceniami dla kompilatora HPF a nie jego instrukcjami, przez co kompilator nie musi się im podporządkowywać jeżeli stwierdzi, że wydajność programu może zostać poprawiona przez



Rysunek 4: Przykłady użycia dyrektywy ALIGN: a) proste wyrównanie dwóch jednowymiarowych tablic, b) wyrównanie z offsetem, c) wyrównanie mniejszej tablicy do większej, d) wyrównanie z jednoczesnym transponowaniem indeksów, e) rozkład wymiaru: wyrównanie dwuwymiarowej tablicy do tablicy jednowymiarowej, f) powielenie danych: wyrównanie jednowymiarowej tablicy do tablicy dwuwymiarowej.

ich zignorowanie. Dyrektywy rozmieszczenia danych funkcjonują w następujący sposób:

- dyrektywa ALIGN porządkuje elementy każdej z tablic (inaczej wyrównuje) wskazując, że powinny zostać rozmieszczone na procesorach w ten sam sposób;
- dyrektywa PROCESSORS określa kształt i rozmiar logicznej tablicy procesorów;
- dyrektywa DISTRIBUTE przydziela elementy tablicy (i wszystkich wyrównanych z nią tablic danych) do logicznej tablicy procesorów.

Największe znaczenie dla zagadnienia aglomeracji ma dyrektywa ALIGN, ponieważ określa, które elementy tablicy powinny zostać (o ile to możliwe) przydzielone do tego samego procesorach, a przez to wpływa ostatecznie na lokalność wykonywanych obliczeń warunkując realizację komunikacji.

Operacje między wyrównanymi strukturami danych są prawdopodobnie znacznie bardziej efektywne niż operacje między strukturami, które nie zostały wyrównane. Ogólna postać dyrektywy ALIGN jest następująca:

```
!HPF$ ALIGN array WITH target
```

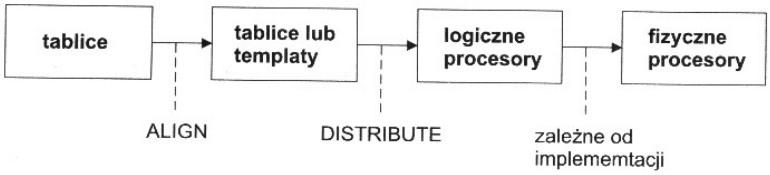
co wskazuje, że tablica `array` powinna zostać wyrównana z tablicą `target`. Użyteczne w wielu przypadkach może okazać się zastosowanie dyrektywy TEMPLATE, która pozwala na zadeklarowanie abstrakcyjnej przestrzeni indeksowej, a następnie użycie jej jako wzorca dla wyrównania elementów pozostałych tablic. Dla przykładu, poniższy fragment kodu określa proste wyrównanie tablic B i C, w którym każdy element B(i) jest wyrównany z korespondującym elementem C(i):

```
real B(50), C(50)
!HPF$ ALIGN C(:) WITH B(:)
```

W dyrektywie ALIGN mogą zostać użyte pseudo argumenty do określenia wymiarów tablic, wyrażenia całkowitoliczbowe do określenia offsetu oraz symbol "*" do określenia rozkładu wymiarów tablic. Na Rysunku 4 przedstawiono kilka przykładów użycia tych mechanizmów.

ALIGN może również zostać użyty do określenia, które z elementów tablicy powinny zostać powielone na różnych procesorach. Umożliwia to przeważnie poprawę efektywności programu w sytuacji, gdy tablica jest częściej odczytywana niż zapisywana. Programista musi jednak zwrócić szczególną uwagę, aby nie powielać tablic często modyfikowanych, ponieważ doprowadzi to do znacznego wzrostu komunikacji i/lub nadmiarowych obliczeń.

Krytyczną kwestią, która powstaje przy rozmieszczaniu różnych struktur danych jest ich konsekwentne współdzielenie przez wszystkie komponenty programu. Na przykład, dla określonego komputera i roz-



Rysunek 5. Strategia planowania w języku HPF.

miaru problemu wywoływany w programie HPF podprogram przypuszczalnie cechuje się optymalnym rozmieszczeniem danych w tym sensie, że minimalizuje czas wykonania tego podprogramu. Jakkolwiek takie optymalne rozmieszczenie może nie odpowiadać rozmieszczeniu określonemu w programie wywołującym dla zmiennych przekazywanych jako argumenty wywołania. Stąd możliwa jest realizacja dwóch różnych strategii rozmieszczenia danych w wywoływanych programach HPF:

- zignorowanie rozmieszczenia użytego w programie wywołującym i określenie lokalnego rozmieszczenia pomimo, iż wymaga to potencjalnie kosztownej operacji ponownej organizacji danych do procesorów. W tym celu stosuje się zwykłe dyrektywy rozmieszczenia danych. Ponowne rozmieszczenie struktur danych jest niszczone po powrocie z podprogramu, przez co całkowity koszt przemieszczania danych między procesorami jest podwójny.
- przyjęcie rozmieszczenia danych użytego w programie wywołującym, nawet jeżeli nie jest ono lokalnie optymalne. W tym celu stosuje się dyrektywę INHERIT, np.:

```
  subroutine fft(n, Z)
  real Z(n,n)
  ...
!HPF$      INHERIT Z  !Z przejmuje dotychcza
                      !sowe rozmieszczenie
  ...
  end
```

- Przyjęcie w określonych okolicznościach jednej z powyższych strategii wiąże się z koniecznością podjęcia kompromisowego wyboru. Koszt ponownego rozmieszczenia danych związany ze strategią pierwszą powinien być rozważony w stosunku do spadku wydajności mogącej zajść przy strategii drugiej. Podobnie, wysiłek włożony w optymalizację podprogramu przy określaniu rozmieszczenia danych powinien zostać rozważany w stosunku do czasu wykonania podprogramu i częstotliwości jego wykorzystywania. W niektórych sytuacjach korzystne może okazać się zamieszczenie w podprogramie różnych fragmentów kodu dla różnych rozmieszczeń.

Planowanie

Rysunek 5 ilustruje dwuetapową strategię planowania realizowaną w języku HPF. Przyjęcie takiego sposobu planowania redukuje liczbę zmian niezbędnych przy przenoszeniu programu z jednej architektury na inną. Wprawdzie różne maszyny mogą wymagać odmiennych strategii podziału ze względu na liczbę dostępnych procesorów, lecz jest mniej prawdopodobne, że będą wymagać zmian przy wyrównywaniu elementów tablic uczestniczących w operacjach.

Konsekwentnie do omówionego w poprzednim rozdziale zagadnienia tworzenia związku między tablicami na etapie aglomeracji w języku HPF, etap planowania wiąże się z użyciem dyrektyw PROCESSORS oraz DISTRIBUTE, realizujących przydział elementów tablicy do logicznej tablicy procesorów. Planowanie logicznych procesorów do fizycznych procesorów jest w HPF zależne od implementacji.

Informacje o procesorach

HPF w stosunku do Fortranu 90 wprowadza dwie dodatkowe funkcje NUMBER_OF_PROCESSORS i PROCESSORS_SHAPE, umożliwiające programowi uzyskanie informacji o liczbie dostępnych fizycznych procesorów oraz topologii ich połączeń. Uzyskane w ten sposób informacje mogą zostać użyte do budowy efektywnych programów wykonywanych przez maszyny z różną liczbą procesorów i konfiguracją ich połączeń. Wywołanie funkcji NUMBER_OF_PROCESSORS przyjmuje ogólną postać:

```
NUMBER_OF_PROCESSORS(dim)
```

gdzie `dim` jest opcjonalnym argumentem. Funkcja zwraca liczbę wszystkich fizycznych procesorów, bądź, gdy opcjonalny argument jest obecny, rozmiar tablicy procesorów wzdłuż określonego nim wymiaru. Wywołanie funkcji PROCESSORS_SHAPE przyjmuje ogólną postać:

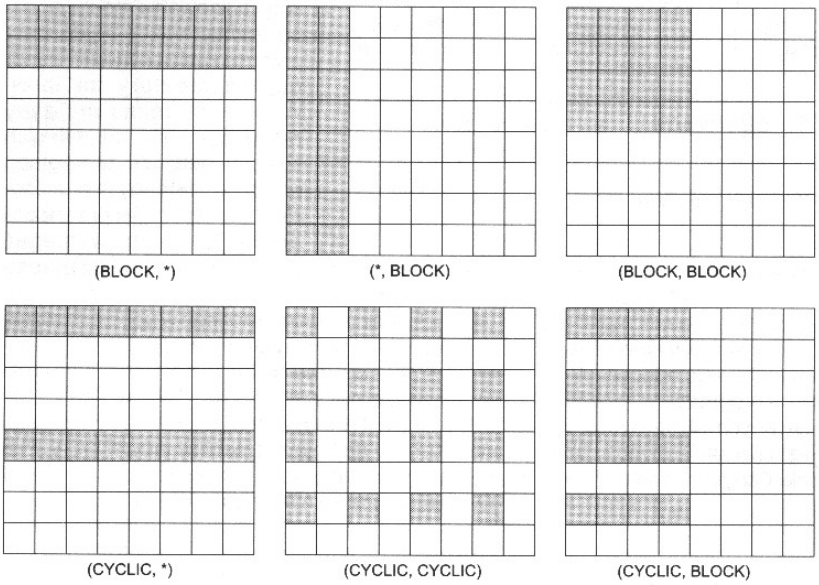
```
PROCESSORS_SHAPE()
```

i zwraca wymiar tablicy procesorów wraz z jej rozmiarami w każdym z wymiarów.

Przedstawienie konkretnego komputera równoległego jako tablicy procesorów jest zależne od implementacji języka HPF. Poniżej zamieszczono przykładowe wyniki działania obu funkcji:

```
!komputer złożony z 32-procesorów połączonych
w siatkę 4x8
PROCESSORS_SHAPE()              ? /4, 8/
NUMBER_OF_PROCESSORS()  ??32
NUMBER_OF_PROCESSORS(1) ??4
NUMBER_OF_PROCESSORS(2) ??8

!jednoprocesorowa stacja robocza
PROCESSORS_SHAPE()              ??/1/
NUMBER_OF_PROCESSORS()  ??1
```



Rysunek 6: Różne sposoby rozmieszczania danych z użyciem dyrektywy DISTRIBUTE.

Użycie omawianych funkcji przy deklaracji tablic i w dyrektywach języka HPF umożliwia programiście tworzenie logicznych tablic procesorów, które odpowiadają dostępnym zasobom fizycznym konkretnej maszyny równoległej, np.:

```
!HPF$ PROCESSORS P(NUMBER_OF_PROCESSORS())
integer Q(SIZE(PROCESSORS_SHAPE()))
```

Dyrektywa PROCESSORS

Dyrektywa PROCESSORS deklaruje logiczną topologię procesorów. Dla przykładu, oba poniższe wyrażenia deklarują 32 logiczne procesory tworzące jednakże różne topologie:

```
!HPF$ PROCESSORS P(32)     !liniowe połączenie 32
                           !procesorów
!HPF$ PROCESSORS Q(4,8)    !sześcian 4x8 procesorów
```

Zazwyczaj jeden logiczny procesor jest tworzony dla każdego procesora fizycznego, chociaż niektóre implementacje języka mogą do deklaracji logicznych procesorów używać mniejszej liczby fizycznych procesorów. Planowanie logicznych procesorów do fizycznych procesorów nie jest określone w HPF i może zmieniać się zależnie od implementacji.

Dyrektywa DISTRIBUTE

Dyrektywa DISTRIBUTE wskazuje, jak dane powinny zostać podzielone między dostępne procesory. Dla każdego wymiaru tablicy zostaje określony sposób planowania elementów tablicy do logicznych procesorów w tworzonej przez nie topologii. Każdy wymiar tablicy może zostać rozmieszczony na jeden z trzech sposobów:

*: brak rozmieszczenia
BLOCK(n): rozmieszczenie blokowe (domyślnie: n = N/P) CYCLIC(n): rozmieszczenie cykliczne (domyślnie: n = 1)

gdzie N jest liczbą elementów rozmieszczanego wymiaru tablicy, a P liczbą procesorów.

Rysunek 6 ilustruje działanie dyrektywy DISTRIBUTE, rozmieszczającej dwuwymiarową tablicę 8x8 na czterech procesorach zorganizowanych w jednowymiarową tablicę (trzy przykłady z "*") oraz w tablicę 2x2 (trzy pozostałe przykłady). Dane przydzielone do jednego procesora zostały zaznaczone zacienionym polem.

Dodanie do dyrektywy DISTRIBUTE argumentu ONTO definiuje rozmieszczenie danych dla konkretnej topologii procesorów. Przy pominięciu ONTO kompilator samodzielnie dokona wyboru odpowiedniej konfiguracji, np.:

```
!HPF$ PROCESSORS p(32)
real D(1024), E(1024)
!HPF$ DISTRIBUTE D(BLOCK)
!HPF$ DISTRIBUTE E(BLOCK) ONTO p
```

Użycie dyrektywy rozmieszczenia odnosi się do wszystkich powiązanych ze sobą (wyrównanych) tablic. Dla przykładu, poniższa dyrektywa DISTRIBUTE określa planowanie dla wszystkich trzech tablic:

```
!HPF$ PROCESSORS p(20)
real A(100,100), B(100,100), C(100,100)
!HPF$ ALIGN B(:,:) WITH A(:,:)
!HPF$ ALIGN C(i,j) WITH A(j,i)
!HPF$ DISTRIBUTE A(BLOCK,*) ONTO p
```

Artykuł powstał dzięki nieocenionej pomocy Pana mgr inż. Krzysztofa Kraski pracownika Wydziału Informatyki na Politechnice Szczecińskiej. *Bardziej obszerny opis zagadnienia można znaleźć w książkach: I. Foster, Designing and Building Parallel Programs, Addison-Wesley, 1995; G. V.Wilson, Practical Parallel Programming, The MIT Press Cambridge Massachusetts, London 1995; M. Ben-Ari, Podstawy programowania współbieżnego i rozproszonego, Wydawnictwa Naukowo-Techniczne, Warszawa 1996.* ■