Wavefronting

Z gniazda zawierającego k zewnętrznych w pełni przestawialnych pętli można również łatwo wygenerować $k-1$ wewnętrznych zrównoleglalnych pętli. Choć potokowanie jest rozwiązaniem preferowanym, dołączamy tę informację dla kompletności rozważań.

Dzielimy obliczanie gniazda z k w pełni przestawialnymi zewnętrznymi pętlami przy użyciu nowej zmiennej indeksowej i', zdefiniowanej jako pewna dodatnia kombinacja liniowa wszystkich k indeksów przestawialnych pętli gniazda. Przykładem takiej kombinacji może być $i' = i_1 + \ldots + i_k$.

Tworzymy najbardziej zewnętrzną pętlę sekwencyjną, iterującą się po partycjach i' w rosnącej kolejności; obliczenia zagnieżdżone w każdej partycji są uporządkowane tak samo jak poprzednio. Pierwszych $k-1$ pętli w każdej partycji na pewno można zrównoleglić. Intuicyjnie dla dwuwymiarowej przestrzeni iteracji pokazana transformacja grupuje iteracje wzdłuż przekątnych biegnących pod kątem 135° w wykonaniu najbardziej zewnętrznej pętli. Ta strategia gwarantuje, że wewnątrz iteracji pętli zewnętrznej nie występują zależności danych.

Tworzenie bloków

W pełni przestawialne gniazdo pętli o głębokości k można zblokować wzdłuż k wymiarów. Zamiast przydzielać iteracje do procesorów w zależności od indeksów zewnętrznej lub wewnętrznej pętli, możemy jako poszczególne jednostki zebrać bloki iteracji. Zblokowywanie pozwala poprawić lokalność danych, a także zminimalizować nadmiarowe obciążenie potokowania.

Przypuśćmy, że mamy dwuwymiarowe, w pełni przestawialne gniazdo pętli, takie jak na rysunku 11.55(a) i chcielibyśmy podzielić obliczenia na bloki

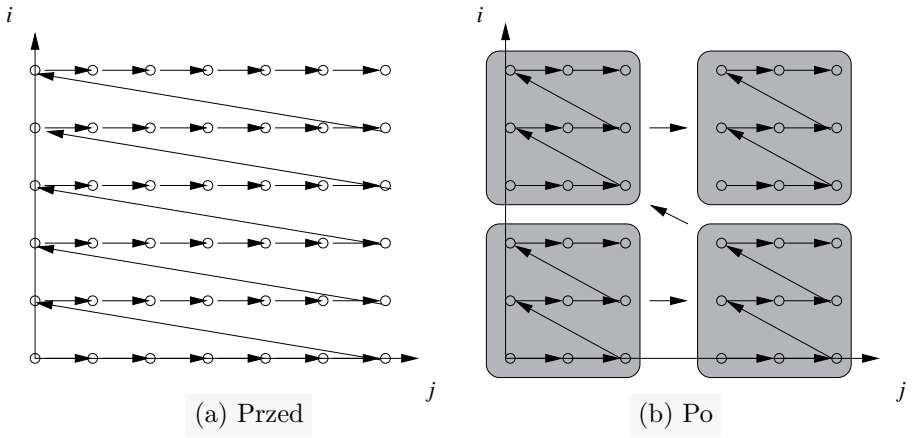
```
for (i=0; i<n; i++)
   for (j=0; j<n; j++) {
      <S>
   }
```

(a) Proste gniazdo pętli

```
for (ii = 0; ii<n; ii+=b)
   for (jj = 0; jj<n; jj+=b)
      for (i = ii; i < min(ii+b, n); i++)
         for (j = jj; j < min(jj+b, n); j++) {
            <S>
            }
```

(b) Zblokowana wersja tego gniazda

RYSUNEK 11.55: 2-wymiarowe gniazdo pętli i jego wersja blokowa



(a) Przed (b) Po

RYSUNEK 11.56: Kolejność wykonania przed zblokowaniem i po zblokowaniu gniazda pętli

o rozmiarach $b \times b$. Kolejność wykonywania zblokowanego kodu pokazana jest na rysunku 11.56, a równoważny kod na rysunku 11.55(b).

Jeśli każdy blok przydzielimy jednemu procesorowi, wówczas cały transfer danych z jednej iteracji do innej w tym samym bloku nie wymaga żadnej komunikacji między procesorami. Alternatywnie możemy zwiększyć ziarnistość potokowania, przydzielając jednemu procesorowi całą kolumnę bloków. Zauważmy, że każdy procesor synchronizuje się teraz ze swoimi poprzednikami i następnikami tylko na granicach bloków. Zatem kolejną korzyścią ze zblokowywania jest to, że programy muszą przesyłać jedynie dane występujące na granicach między sąsiadującymi blokami. Wartości wewnątrz każdego bloku są obsługiwane przez pojedynczy procesor.

Przykład 11.63: Użyjemy teraz prawdziwego numerycznego algorytmu – rozkładu Choleskiego – aby pokazać, jak algorytm 11.59 obsługuje pojedyncze gniazda pętli jedynie przy użyciu potokowania. Kod ten, pokazany na rysunku 11.57, implementuje algorytm o czasie wykonania $O(n^3)$ operujący na

```
for (i = 1; i <= N; i++) {
    for (j = 1; j <= i-1; j++) {
        for (k = 1; k <= j-1; k++)
            X[i,j] = X[i,j] - X[i,k] * X[j,k];
        X[i,j] = X[i,j] / X[j,j];
    }
    for (m = 1; m <= i-1; m++)
        X[i,i] = X[i,i] - X[i,m] * X[i,m];
    X[i,i] = sqrt(X[i,i]);
}
```

RYSUNEK 11.57: Rozkład Choleskiego