1. Pętle pojedyncze

Chociaż pętle zagnieżdżone zdają się dawać dużo większe możliwości równoleglizacji kolejnych ich iteracji oraz optymalizacji poprzez zmianę kolejności ich wykonywania, w przypadku pętli pojedynczych również możliwe są spore optymalizacje. Co więcej, większość pętli w programach to właśnie pętle pojedyncze. Dlatego należy również zwrócić na nie uwagę. Optymalizacja pętli pojedynczych zdaje się być dużo prostsza niż optymalizacja pętli zagnieżdżonych, dlatego to właśnie zagadnienie winno być rozpatrzone jako pierwsze.

1.1 Przypadki proste

Jako przypadki proste rozumiane są tu pętle, których żaden element (z wyjątkiem inicjalizacji, gdyż ta wykonywana jest przed rozpoczęciem pętli, a co za tym idzie nie wprowadza żadnych zależności pomiędzy jej iteracjami) nie używa instrukcji warunkowych oraz wiadoma jest (nawet jeśli dopiero w momencie wykonywania programu) ilość iteracji. Ilość iteracji pętli nie musi być znana w momencie inicjalizowania pętli. Istotne jest by przed wykonaniem równoległym kolejnych iteracji (jeśli zrównoleglenie w danym przypadku jest możliwe) wiadomym było czy i ile iteracji powinno być wykonane.

1.1.1 Brak zależności

Wszystkie iteracje pętli mogą być od siebie niezależne. Przykładem takiej pętli może być mnożenie wektora przez skalar lub dodawanie dwóch wektorów.

```
for(i = 0;i < M;++i)
{
 X[i] *= C;
}
```

W powyższym przykładzie wszystkie iteracje są od siebie całkowicie niezależne, co znaczy, że mogą być wykonywane równolegle oraz w dowolnej kolejności. Mogą także być grupowane w bloki po kilka w zależności od dostępnych zasobów. Oczywiście operator może być różny od mnożenia. Element C również nie musi być stały.

```
for(i = 0;i < M;++i)
{
 X[i] += Y[f(i)];
}
```

W powyższym przypadku nie wystąpią żadne zależności niezależnie od indeksowania Y gdyż jest on wyłącznie odczytywane i żadna z iteracji go nie modyfikuje. Potencjalne zależności mogłyby być stworzone przez funkcję indeksującą f jeśli modyfikowałaby ona indeks 'i' lub była zależna od innych zmiennych globalnych czy statycznych, których zależności uniemożliwiałyby zrównoleglenie lub byłyby trudne do przewidzenia. Przypadki takie należy jednak rozpatrzeć oddzielnie. Jeśli założyć stałą ilość iteracji w rozwinięciu pętli, można przedstawić ją w następujący sposób:

```
for(i = 0;i < (M - N - 1);i += N)
{
 X[i] += Y[f(i)];
```

```
 X[i + 1] += Y[f(i + 1)];
 ...
 X[i + N - 1] += Y[f(i + N - 1)];
}
instrukcja_końcowa(X,Y,i,M,N);
```

Powyższe rozwinięcie pokazuje wykonanie pętli blokami po N iteracji wykonywanych równolegle. Znajdująca się po pętli instrukcja końcowa służy rozwiązaniu problemów tworzonych przez warunki brzegowe omówionych w podpunkcie 1.1.4.

1.1.2 Zależności pomiędzy następującymi iteracjami

W wielu pętlach mogą występować zależności pomiędzy kolejnymi ich iteracjami. Przypadki takie uniemożliwiają jakiekolwiek zrównoleglenie kolejnych iteracji.

```
for(i = 0;i < M;++i)
{
 X[i + 1] = X[i];
}
```

W powyższym przykładzie kierunek zależności nie ma znaczenia. Nieistotnym jest czy następna iteracja jest zależna od poprzedniej czy poprzednia od następnej. W żadnym przypadku zrównoleglenie pętli nie jest możliwe i iteracje muszą być wykonane kolejno. Zależność taka nie musi być spowodowana indeksowaniem.

```
for(i = 0;i < M;++i)
{
 X += C;
}
```

Powyższy przykład różni się od pierwszego przypadku opisanego w podpunkcie 1.1.1 wyłącznie brakiem indeksowania (zmiana samego operatora nie wpływa na zależności). Modyfikacja ta sprawia, że iteracja zależy od wszystkich poprzednich. Analogiczna zależność wystąpi jeśli usuniemy indeksowanie elementu X w drugim przykładzie podpunktu 1.1.1.

```
for(i = 0;i < M;++i)
{
 X += Y[f(i)];
}
```

Analogicznie jak w poprzednim przypadku zależność jest wprowadzona przez element X podczas gdy element Y nie generuje żadnych zależności gdyż jest wyłącznie odczytywany i żadna z iteracji pętli go nie modyfikuje. Istotnym jest również zwrócenie uwagi na fakt, iż nawet jeśli w poprzednim przykładzie uda się rozwinąć pętle i zastąpić kolejne jej iteracje mnożeniem wynik nie dla każdego typy danych będzie identyczny.

```
for(i = 0;i < M;++i)
{
 X += C;
}
```

oraz:

```
X += (M * C);
```

mogą różnic się w przypadku użycia liczb zmiennoprzecinkowych z powodu błędów zaokrąglenia. W przypadku liczb całkowitych problemem taki nie występuje. Jednak w przypadku gdy kolejność wykonywania operacji nie ma znaczenia można wcześniejszy przykład:

```
for(i = 0;i < M;++i)
{
 X += Y[f(i)];
}
```

rozwinąć do postaci:

```
i = 0;
if (M >= N)
{
 X1 = Y[f(0)];          /*początek bloku instrukcji wykonywanych równolegle*/
 X2 = Y[f(1)];
 …
 XN = Y[f(N – 1)];
 i = N;                 /*koniec bloku instrukcji wykonywanych równolegle*/
}
for(;i < (M – N – 1);i += N)
{
 X1 += Y[f(i)];         /*początek bloku instrukcji wykonywanych równolegle*/
 X2 += Y[f(i + 1)];
 …
 XN += Y[f(i + N - 1)];/*koniec bloku instrukcji wykonywanych równolegle*/
}
instrukcja_końcowa(X,Y,i,M,N);
X += (X1 + X2 + … + XN);
```

W powyższym rozwinięciu nadal iteracje są wzajemnie od siebie zależne, jednak instrukcje w obrębie pojedynczej iteracji już nie, a co za tym idzie mogą być wykonane równolegle. Znajdująca się przed rozpoczęciem pętli instrukcja warunkowa ma na celu zainicjalizowanie zmiennych pomocniczych i zastąpić tym samym pierwszą z iteracji nowej pętli. Znajdująca się poza pętlą instrukcja końcowa ma na celu rozwiązanie problemów związanych z warunkami brzegowymi (patrz podpunkt 1.1.4). Ponieważ w szczególnym przypadku przy odgórnym założeniu stałego rozmiaru bloku instrukcji wykonywanych równolegle (N) możliwa jest sytuacja, w której należy wykonać mniej niż jeden blok instrukcji równoległych, koniecznym jest dodanie instrukcji warunkowej mającej na celu zapewnić poprawne działanie programu nawet w takim przypadku.

1.1.3 Zależności pomiędzy iteracjami nienastępującymi po sobie

Możliwy jest przypadek by nawet w pętlach prostych występowały zależności pomiędzy iteracjami nienastępującymi bezpośrednio po sobie. W takich przypadkach iteracje niezależne od siebie mogą być wykonane blokami równolegle.

```
for(i = 0;i < M;++i)
{
 X[i + 2] = X[i];
}
```

W powyższym przykładzie iteracje parzyste są zależne wyłącznie od parzystych, a nieparzyste od nieparzystych. Inne zależności pomiędzy iteracjami nie zachodzą. Dzięki temu iteracje pierwsza z drugą mogą być wykonane równolegle, gdyż nie ma żadnych zależności pomiędzy nimi. Dalej możliwe jest równoległe wykonanie iteracji trzeciej z czwartą, piątej z szósta itd. Powyższa pętlę można rozwinąć do postaci:

```
for(i = 0;(M != 0) && (i < (M - 1));i += 2)
{
 X[i + 2] = X[i];
 X[i + 3] = X[i + 1];
}
if (i != M)
{
 X[M + 1] = X[M - 1];
}
```

W powyższym rozwinięciu należy zwrócić uwagę na fakt, iż M może być liczba nieparzystą, a co się z tym wiąże należy uwzględnić warunki brzegowe. Problem warunków brzegowych został opisany w podpunkcie 1.1.4.

```
for(i = 0;i < M;++i)
{
 X[i + 3] = X[i];
}
```

W powyższym przykładzie iteracje mogą być wykonywane równolegle trójkami, gdyż analogicznie jak w poprzednim przypadku nie występują pomiędzy nimi żadne zależności.

```
for(i = 0;(M > 1) && (i < (M - 2));i += 3)
{
 X[i + 3] = X[i];
 X[i + 4] = X[i + 1];
 X[i + 5] = X[i + 2];
}
if (i != M)
{
 if (i != (M - 1))
 {
  X[M + 1] = X[M - 2];
 }
 X[M + 2] = X[M - 1];
}
```

W tym jak i w poprzednim rozwinięciu instrukcje znajdujące się w ciele pętli są od siebie wzajemnie niezależne, co oznacza, ze mogą być wykonywane równolegle. Sama ilość iteracji

również uległa zmniejszeniu. Niestety jednak warunki brzegowe mogą być skomplikowane, co sprawia, że optymalizacja ta może się nie sprawdzić dla małych pętli wykonywanych raptem kilka razy. Oba powyższe przypadki można uogólnić do postaci:

```
for(i = N;i < M;++i)
{
 X[i ±C] = f(X[i ±D]);
}
```

W postaci ogólnej iteracje pomiędzy ±C, a ±D mogą być wykonane równolegle pod warunkiem, że funkcja f nie wprowadza dodatkowych zależności, zezwalają na to zasoby oraz nie koliduje to z warunkami brzegowymi pętli.

1.1.4 Warunki brzegowe

Jeśli iteracje pętli są wykonywane równolegle blokami po N iteracji to jeśli do końca pętli pozostało jeszcze M iteracji może okazać się, że M jest mniejsze od N. W takim przypadku część iteracji z bloku nie może zostać wykonana, gdyż mogłoby to spowodować niewłaściwe działanie programu. W przypadku prostych pętli pojedynczych sytuacja taka nastąpi za każdym razem kiedy łączna ilość iteracji nie będzie podzielna przez ilość iteracji wykonywanych równolegle w bloku. W takim wypadku konieczne jest użycie bloku wykonującego mniejszą ilość iteracji lub niezapisywanie wartości rejestrów przechowujących wyniki nadmiarowo wykonanych iteracji jeśli samo wykonanie operacji nie ma wpływu na program a jedynie jej wartość. W przypadkach krytycznych koniecznym może okazać się sekwencyjne wykonanie ostatnich iteracji. Jedynie pętle nieskończone nie wymagają rozpatrywania warunków brzegowych, gdyż takowe nie nastąpią. Jednakże pętla nieskończona bez możliwości przerwania jej wykonania zdaje się być niepoprawna. Więcej na temat warunkowego przerywania pętli w podpunkcie 1.2.3.

1.1.5 Pętle z zależnościami wielokrotnymi

Możliwym jest by pętla była zależna od więcej niż jednej iteracji. W przypadku takim o możliwości wykonania równoległego decyduje gorsza z zależności. Dla przykładu:

```
for(i = 0;i < M;++i)
{
  X[i] = X[i + 2] + X[i + 3];
}
```

W powyższym przykładzie równolegle mogą być wykonywane iteracje: pierwsza z drugą, trzecia z czwartą itd. Choć każda z powiązana jest z dwiema po niej następującymi tak naprawdę przypadek ten z racji gorszej zależności da się uprościć do przykładu opisanego jako pierwszy w podpunkcie 1.1.3. i jako taki należy go rozpatrywać.

1.1.6 Pętle z więcej niż jedną instrukcją w ciele

Większość pętli ma więcej niż jedną instrukcję w ciele. Sytuacje takie możemy rozdzielić na przypadki, kiedy wspomniane instrukcje są od siebie wzajemnie niezależne oraz na takie, gdzie występuje pomiędzy przynajmniej dwójką z nich zależność. Przypadki instrukcji bez wzajemnych zależności możemy ideowo traktować jak kilka pętli wzajemnie od siebie niezależnych, a co się z tym wiąże rozpatrywać je (lub wręcz wykonywać) niezależnie od siebie. Dla przykładu pętlę:

```
for(i = 0;i < M;++i)
{
 X[i + 2] = X[i];
 Y[i + 3] = Y[i];
}
```

możemy traktować jako dwie całkowicie niezależne, dające się wykonać równolegle pętle:

```
for(i1 = 0;i1 < M;++i1)
{
 X[i1 + 2] = X[i1];
}
```

oraz:

```
for(i2 = 0;i2 < M;++i2)
{
 Y[i2 + 3] = Y[i2];
}
```

obie dające się zrównoleglić w sposób opisany w podpunkcie 1.1.3. Co więcej, w takim przypadku nawet jeśli jedna lub więcej z instrukcji wymusza sekwencyjne wykonywanie kolejnych iteracji, możliwe jest równoległe wykonywanie pozostałych instrukcji oraz jednoczesne wykonywanie sekwencyjne. W przypadku jeśli występują zależności pomiędzy instrukcjami w iteracjach określenie zależności pomiędzy iteracjami zdaje się być trudniejsze. Dla przykładu w pętli:

```
for(i = 0;i < M;++i)
{
 X[i + 2] = X[i];
 X[i + 3] = X[i];
}
```

zależności pomiędzy iteracjami zdają się być identyczne jak w przykładzie opisanym w podpunkcie 1.1.5, jednak nie możemy w tym przypadku zastosować tamtego rozwiązania gdyż każde dwie kolejne iteracje mają jeden wspólny element, który modyfikują. Powyższy przykład różni się od wspomnianego przykładu opisanego w podpunkcie 1.1.5 jedynie kierunkiem zależności. Tam odczytywane były dane, które miały być dopiero zmodyfikowane przez następne iteracje. Tutaj modyfikowane są dane, odczytywane w następnych iteracjach. Choć w takiej postaci równoległe wykonanie którychkolwiek iteracji jest niemożliwe, zależność tą można usunąć poprzez zmianę samej pętli na następujący kod:

```
for(i = 0;i < M;++i)
{
 X[i + 2] = X[i];
}
X[M + 2] = X[M - 1];
```

Po tej transformacji pętla została uproszczona do przypadku opisanego w podpunkcie 1.1.3 oraz pojedynczej dodatkowej instrukcji znajdującej się poza samą pętlą. Rozwiązanie to działa pod warunkiem, że M jest różne od zera. W przeciwnym razie pętla nie zostanie wykonana, więc nie ma potrzeby dokonywania jakichkolwiek jej modyfikacji. Zaproponowane rozwiązanie opierało się na fakcie, iż wartość zapisana przez instrukcję:

```
X[i + 3] = X[i];
```

zostanie nadpisana przez wykonana przez następną iterację instrukcję:

```
X[i + 2] = X[i];
```

co nie nastąpi jedynie dla ostatniej iteracji. Z tej przyczyny właśnie koniecznym było dodanie instrukcji znajdującej się za pętlą. Jest ona jednak niestety zależna od iteracji dla której 'i' jest równe M – 3, a ta niestety jest zależna od wcześniejszych, co znaczy, że może być wykonana równolegle dopiero z dwoma ostatnimi iteracjami.

Uproszczenie takie nie zawsze jest jednak możliwe. Rozpatrzmy drobną modyfikację powyższego przykładu:

```
for(i = 0;i < M;++i)
{
 X[i + 2] += X[i];
 X[i + 3] += X[i];
}
```

W tym przypadku poprzednie rozwiązanie nie może zostać zastosowane gdyż nie występuje wyłącznie nadpisanie wartością, lecz do obliczenia nowej wartości niezbędna jest wartość obliczona przez instrukcję z poprzedniej iteracji. Tworzy to sytuację w której nie istnieją iteracje będące od siebie niezależne, co oznacz, ze pętla musi być wykonana sekwencyjnie bez możliwości wykonania jakichkolwiek jej iteracji równolegle. Warto zwrócić uwagę, że zależność ta została wprowadzona przez instrukcję:

```
X[i + 2] += X[i];
```

natomiast druga z instrukcji, niezależnie czy jest to:

```
X[i + 3] += X[i];
```

czy też:

```
X[i + 3] = X[i];
```

ma wpływ na wynik operacji, ale nie na możliwość optymalizacji lub brak takowej.

1.2 Pętle złożone

W sytuacjach, w których pętle zawierają w sobie instrukcje warunkowe lub ilości iteracji nie daje się przewidzieć, wyznaczenie zależności pomiędzy iteracjami, a więc również możliwości równoległego ich wykonywania, okazuje się być nietrywialne, jeśli w ogóle możliwe.

1.2.1 Pętle w których nie da się przewidzieć ilości iteracji

Klasycznym przykładem pętli w której w żadnym momencie nie da się z góry przewidzieć ile iteracji jeszcze pozostało jest pętla jaka jest używana do badania długości ciągu znaków.

```
while(*(X++));
```

Chociaż w powyższym przykładzie między kolejnymi iteracjami pętli nie występują żadne zależności (gdyż żadna ze zmiennych prócz tej, którą można uznać za licznik pętli nie jest modyfikowana przez kolejne iteracje, więc na dobrą sprawę pętla ta nie robi nic co wymagałoby wykonywania synchronicznego) nie można wykonywać ich równolegle gdyż nie można przewidzieć kiedy trzeba będzie pętle tą przerwać. Przypadek ten można uogólnić do sytuacji kiedy ilość iteracji pętli zależy od danych na których ona operuje nie zaś od licznika czyli pojedynczej wartości.

1.2.2 Pętle z instrukcjami warunkowymi w ciele

Jeżeli warunek w ciele pętli jest zależny od danych na jakich ona operuje, wtedy dokładna predykcja zależności pomiędzy kolejnymi iteracjami bez znajomości tychże danych jest niemożliwa. Co się z tym wiąże jakakolwiek próba równoległego wykonania iteracji może okazać się trudna lub wręcz niemożliwa. O ile we wszystkich poprzednich przykładach byliśmy pewni zależności o tyle teraz są one jedynie możliwe. Mimo to powinniśmy w większości sytuacji traktować je tak samo jak zależności zależności pewne.

```
for(i = 0;i < M;++i)
{
 if (X[i])
 {
  X[i + 1] = X[i];
 }
}
```

W powyższym przykładzie próba równoległego wykonania iteracji może okazać się trudna gdyż możliwa jest zależność iteracji od tej następującej bezpośrednio po niej. Instrukcja warunkowa nie musi wprowadzać wyłącznie pojedynczej zależności.

```
for(i = 0;i < M;++i)
{
 if (X[i])
 {
  X[i + 2] += X[i];
 }
 else
 {
  X[i + 3] += X[i];
 }
}
```

Powyższy przykład jest drobna modyfikacją ostatniego przykładu opisanego w podpunkcie 1.1.6. Chociaż sam układ zależności jest identyczny, należy zwrócić uwagę na fakt, iż są to jedynie zależności możliwe i za każdą iteracja ma miejsce tylko jedna z nich. Oznacza to, że dopiero w momencie wykonywania iteracji (wynika to z faktu, że instrukcja warunkowa zależy od danych na których operuje pętla, nie zaś od licznika, a dane te nie muszą być z góry znane) jesteśmy w stanie stwierdzić czy możliwe jest równoległe wykonanie iteracji po niej następującej. Jeśli nastąpi wykonanie instrukcji:

```
X[i + 2] += X[i];
```

to nie wprowadza ona żadnych zależności, które uniemożliwiałyby równoległe wykonanie następnej iteracji. Oznacza to, że niektóre iteracje da się wykonać parami równolegle, inne natomiast trzeba wykonywać sekwencyjnie. Przykład powyższy można rozwinąć w następujący sposób:

```
for(i = 0;i < (M - 1);)
{
 if (X[i])
 {                      /*początek bloku, który można wykonać równolegle*/
  X[i + 2] += X[i];
  if (X[i + 1])
  {
   X[i + 3] += X[i + 1];
  }
  else
  {
   X[i + 4] += X[i + 1];
  }                      /*koniec równolegle wykonywanych instrukcji*/
  i += 2;
 }
 else
 {
  X[i + 3] += X[i];
  ++i;
 }
}
if (i < M)
{
 if (X[M - 1])
 {
  X[M + 1] += X[M - 1];
 }
 else
 {
  X[M + 2] += X[M - 1];
 }
}
```

Powyższe rozwiniecie jest poprawne jeśli M jest różne od zera. Jednakże jeśli M jest równe zero oryginalna pętla nie zostanie wykonana, zatem żadna jej optymalizacja nie ma sensu. Dodanie instrukcji warunkowej będącej de facto ostatnią iteracją wynika z problemu warunków brzegowych. Ostatnia z iteracji nie powinna wywoływać jeszcze jednej nawet jeśli z powodu zależności teoretycznie może. Oczywiście może okazać się, że przedostatnia z iteracji wykonała ostatnia równolegle. Z tej właśnie przyczyny dodana jest jeszcze jedna instrukcja warunkowa determinująca wykonanie ostatniej z iteracji. Niestety elementu znajdującego się poza pętla nie można wykonać równolegle z żadną z iteracji pętli.

Innym przykładem pętli z wieloma zależnościami pomiędzy iteracjami może być pętla używająca instrukcji switch. Dla przykładu:

```
for(i = 0;i < M;++i)
{
 switch(f(X,i))
 {
  case A:
  instrukcja_1(X,i);
  break;
  case B:
  instrukcja_2(X,i);
  break;
  case C:
  instrukcja_3(X,i);
  break;
 }
}
```

O ile w przypadku instrukcji if … else w konkretnej iteracji naraz mogły zajść zależności tworzone tylko i wyłącznie przez jedną z instrukcji wykonywanych warunkowo (gdyż wykonanie jednej w danej iteracji wyklucza wykonanie w tej samej iteracji drugiej), to w przypadku instrukcji switch taka relacja może, ale nie musi zajść. Chociaż w powyższym przykładzie wykonanie dowolnej z instrukcji 1, 2 czy 3 wyklucza wykonanie w tej samej iteracji pozostałych, przykład ten nie jest uniwersalny. Rozważmy drobna modyfikację:

```
for(i = 0;i < M;++i)
{
 switch(f(X,i))
 {
  case A:
  instrukcja_1(X,i);
  break;
  case B:
  instrukcja_2(X,i);
  case C:
  instrukcja_3(X,i);
  break;
 }
}
```

Teraz przypadek B spowoduje wykonanie instrukcji zarówno drugiej jak i trzeciej, podczas gdy przypadek C spowoduje nadaj wykonanie wyłącznie instrukcji trzeciej. Oznacza to, ze w przypadku B trzeba brać pod uwagę gorsza z zależności tworzonych przez instrukcji 2 i 3. Powyższy przykład można przedstawić w innej postaci:

```
for(i = 0;i < M;++i)
{
 switch(f(X,i))
 {
  case A:
  instrukcja_1(X,i);
  break;
  case B:
```

```
  instrukcja_2(X,i);
  instrukcja_3(X,i);
  break;
  case C:
  instrukcja_3(X,i);
  break;
 }
}
```

W powyższym przykładzie widać, że można zastosować te same mechanizmy optymalizacji co w przypadku instrukcji if … else, należy zwrócić jedynie uwagę na fakt, iż przypadek B składa się z dwóch instrukcji.

Oddzielnie należy zwrócić uwagę na przypadki w których warunek jest zależny wyłącznie od licznika pętli, nie zaś od danych na których pętla operuje lub dane te znane są w momencie kompilacji. W takich przypadkach prostych nie ma potrzeby stosowania zasady zależności prawdopodobnych, gdyż da się jednoznacznie je określić.

Przypadki w których warunek w pętli nie zależy od elementów, które w trakcie wykonywania pętli się zmieniają na chwile obecna rozwijane są przez optymalizacje zawarte w gcc. Na przykład:

```
for(i = 0;i < M;++i)
{
 if (A)
 {
  X[i] += X[i + 1];
 }
 else
 {
  X[i] -= X[i + 1];
 }
}
```

zostanie rozwinięte do postaci:

```
if (A)
{
 for(i = 0;i < M;++i)
 {
  X[i] += X[i + 1];
 }
}
else
{
 for(i = 0;i < M;++i)
 {
  X[i] -= X[i + 1];
 }
}
```

Rozwiniecie to usuwa instrukcję warunkową z ciała pętli, dzięki czemu może ona być rozpatrywana jako dwie niezależne od siebie pętle proste, bez potrzeby rozpatrywania możliwych zależności pomiędzy iteracjami.

1.2.3 Pętle z instrukcjami przerwania oraz kontynuacji

Możliwe jest wystąpienie instrukcji break pomiędzy instrukcjami zawartymi w ciele pętli. Wystąpienie tej instrukcji nie jako element instrukcji warunkowej lecz jako niezależna instrukcja w ciele pętli zdaje się mało prawdopodobne, choć nie jest zabronione. W przypadku takim pętla wykona się raz i tylko do momentu wystąpienia instrukcji przerwania. Przypadek taki nie powinien być rozpatrywany w tym dokumencie gdyż de facto nie jest to pętla. Wystąpienie instrukcji break jako element instrukcji warunkowej prowadzi jednak do sytuacji, w której jeśli nie jesteśmy w stanie przewidzieć zachowania się samego warunku to nie niemożliwym staje się przewidzenie ilości iteracji, co sprowadza się do przypadku opisanego w podpunkcie 1.2.1. Jeśli sam fakt wykonania pozostałych instrukcji w pętli nie ma znaczenia dla samego programu, a istotny jest jedynie ich wynik, można wtedy spróbować wykonywać iteracje równolegle ignorując instrukcję przerwania, a w momencie jej wystąpienia zignorować wynik niepotrzebnie wykonanych instrukcji. Warto zwrócić uwagę na fakt, iż w przypadku jeśli instrukcja break jest poprzedzona jakimiś innymi instrukcjami wykonają się tylko razy i tylko i wyłącznie w ostatniej iteracji pętli.

Wystąpienie instrukcji continue, choć pozornie zaburza wykonywanie się kolejnych iteracji w pętli nie wprowadza większych komplikacji w mechanizmie optymalizacji. Jej wystąpienie może zadecydować jedynie o tym czy należy wykonywać dalsze instrukcje znajdujące się w obrębie obecnej iteracji, czy należy je pominąć. Należy jednak zwrócić szczególna uwagę na fakt, iż instrukcje znajdujące się po instrukcji kontynuacji nie muszą być wykonane, więc jeśli fakt samego ich wykonania może mieć wpływ na zachowanie się programu należy wstrzymać ich wykonanie do momentu zakończenia rozpatrywania czy powinny być wykonane czy też nie. Jeśli istotny dla działania programu jest wyłącznie wynik późniejszych instrukcji, mogą one być wykonane równolegle bez potrzeby wstrzymywania, a wynik ich może być opcjonalnie zignorowany.

1.2.4 Instrukcje skoku

Wykonywanie pętli może również zostać przerwane przez instrukcję skoku. Jeśli instrukcja taka prowadzi do miejsca znajdującego się poza ciałem pętli powinna być traktowana tak samo jak instrukcja przerwania wykonywania pętli, przy czym nie ma tu znaczenia czy skok został wywołany instrukcją goto, czy jest to skok daleki powodowany przez mechanizm wyjątków. Przy tym w przypadku wyjątków można założyć, że sytuacje w których dojdzie do skoku są rzadkie.