

Optymalizacja kodu

Ze wszystkich metod optymalizacji kodu programowego zwrócimy uwagę na:

- **Usunięcie (po możliwości) „skoków” danych**
- **Rozwijanie pętli**
- **Opcje kompilatora**

Usunięcie skoków danych:

```
for(i=1; i<=N; i++)
{
    for(j=1; j<=N; j++)
    {
        ij = N*(j-1)+i;
        A[ij] = .....
    }
}
```

Ten fragment kodu będzie wykonany bardzo wolno, dla tego że elementy dużej tablicy są pobierane z pamięci nie ciągle. To powoduje N^2 odczytów stron pamięci, przy czym z każdej strony będzie pobrane tylko jedno słowo (zakładamy, że $N > M$, gdzie M – ilość słów, które można umieścić w stronie pamięci głównej).

$$A = \begin{pmatrix} 1 & N+1 & 2N+1 & : & (N-1)N+1 \\ 2 & N+2 & 2N+2 & : & (N-1)N+2 \\ 3 & N+3 & 2N+3 & : & (N-1)N+3 \\ : & : & : & : & : \\ N & 2N & 3N & : & N^2 \end{pmatrix}$$

Taki fragment kodu jest zdecydowanie lepiej:

```
for(j=1, ij = 0; j<=N; j++)
{
    for(i=1; i<=N; i++, ij++)
    {
        A[ij] = .....
    }
}
```

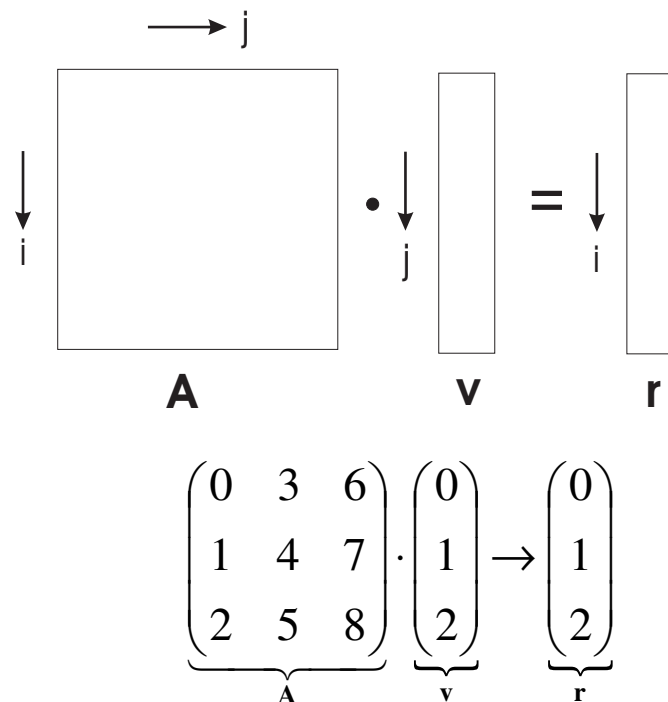
Teraz elementy tablicy A będą pobierane z pamięci ciągle, przy odczycie każdej strony będą użyte wszystkie słowa, znajdujące się w stronie, i tylko po tym będzie odczyt nowej strony. Ilość odczytów stron: $N^2/M \ll N^2$.

➤ Rozwijanie pętli – potoki procesora nie w stanie zacząć opracowywać instrukcji z następnej iteracji dotąd, dokąd nie skończą instrukcji w bieżącej iteracji – nie znają wartości indeksu po inkrementowaniu (wykład 2, Przykł. 1). Dla tego krótkie instrukcje pętli powodują zaburzenie potoków procesora. Rozwijanie pętli w sposób sztuczny przedłuża instrukcje, co nadaje możliwość pełnego wciągnięcia potoków procesora.

➤ Przykład mnożenia macierzy przez wektor: $r = r + A \cdot V$, przy czym macierz jest umieszczona w pamięci jako tablica jednowymiarowa kolumna po kolumnie:

//Brak rozwijania pętli

```
memset((void *)R, 0, sizeof(double)*dim);
for(j=0; j<dim; j++)
{
    for(i=0; i<dim; i++)
    {
        R[i] += A[i*dim+j]*V[j];
    }
}
```



➤ **A teraz rozwinieśmy pętlę 6 razy:**

**//Dostęp do elementów tablicy przez
//indeksy**

```
int rest = dim%6;
for(j=0, ij=0; j<dim; j++)
{
    for(i=0; i<rest; i++, ij++)
    {
        R[i] += A[ij]*V[j];
    }

    for(i=rest; i<dim; i+=6, ij+=6)
    {
        R[i]   += A[ij] * V[j];
        R[i+1] += A[ij+1]*V[j];
        R[i+2] += A[ij+2]*V[j];
        R[i+3] += A[ij+3]*V[j];
        R[i+4] += A[ij+4]*V[j];
        R[i+5] += A[ij+5]*V[j];
    }
}
```

//Arytmetyka wskaźników

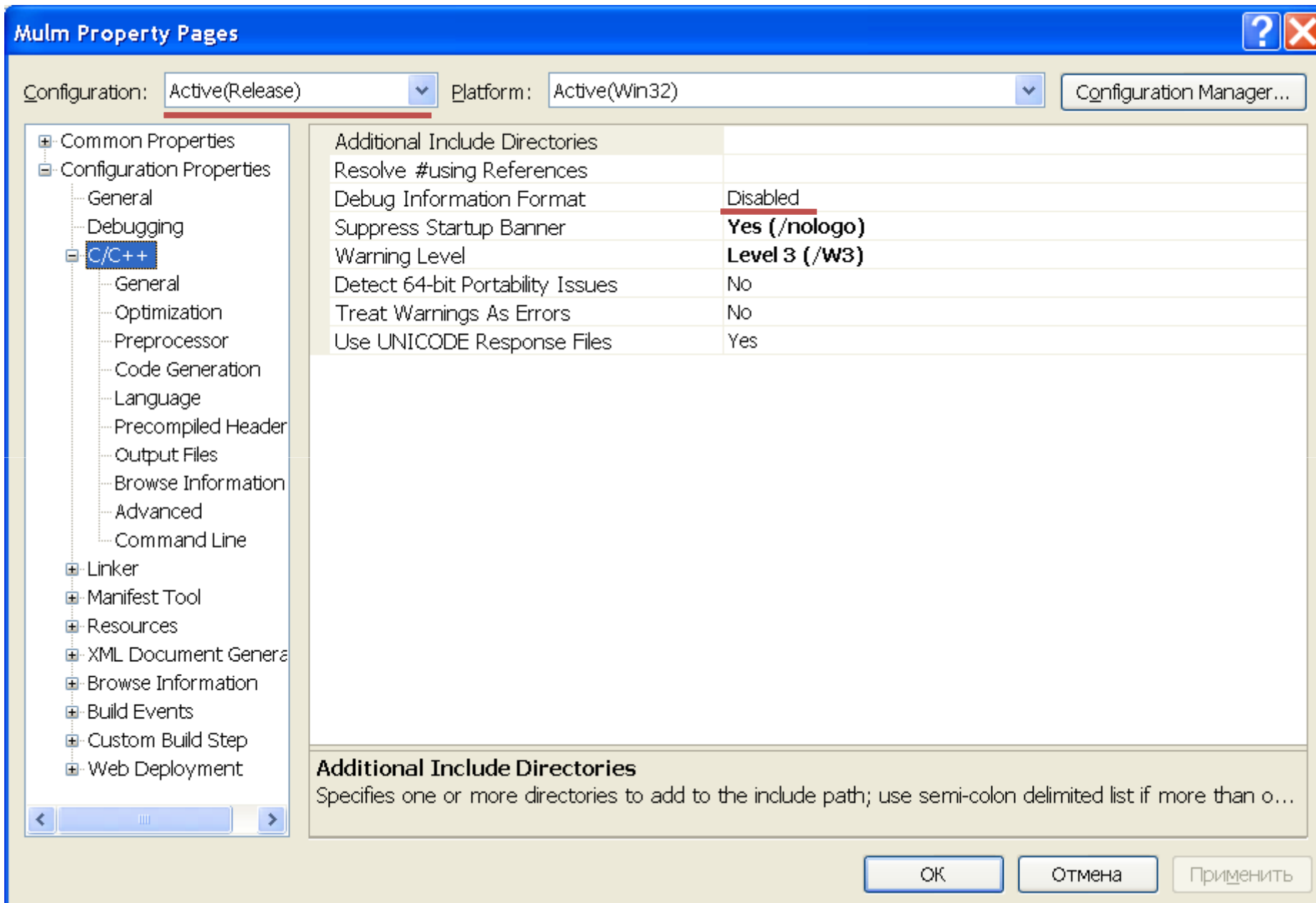
```
double *ptr_A, *ptr_R = &A[0], *ptr_V = &V[0];
int rest = dim%6;

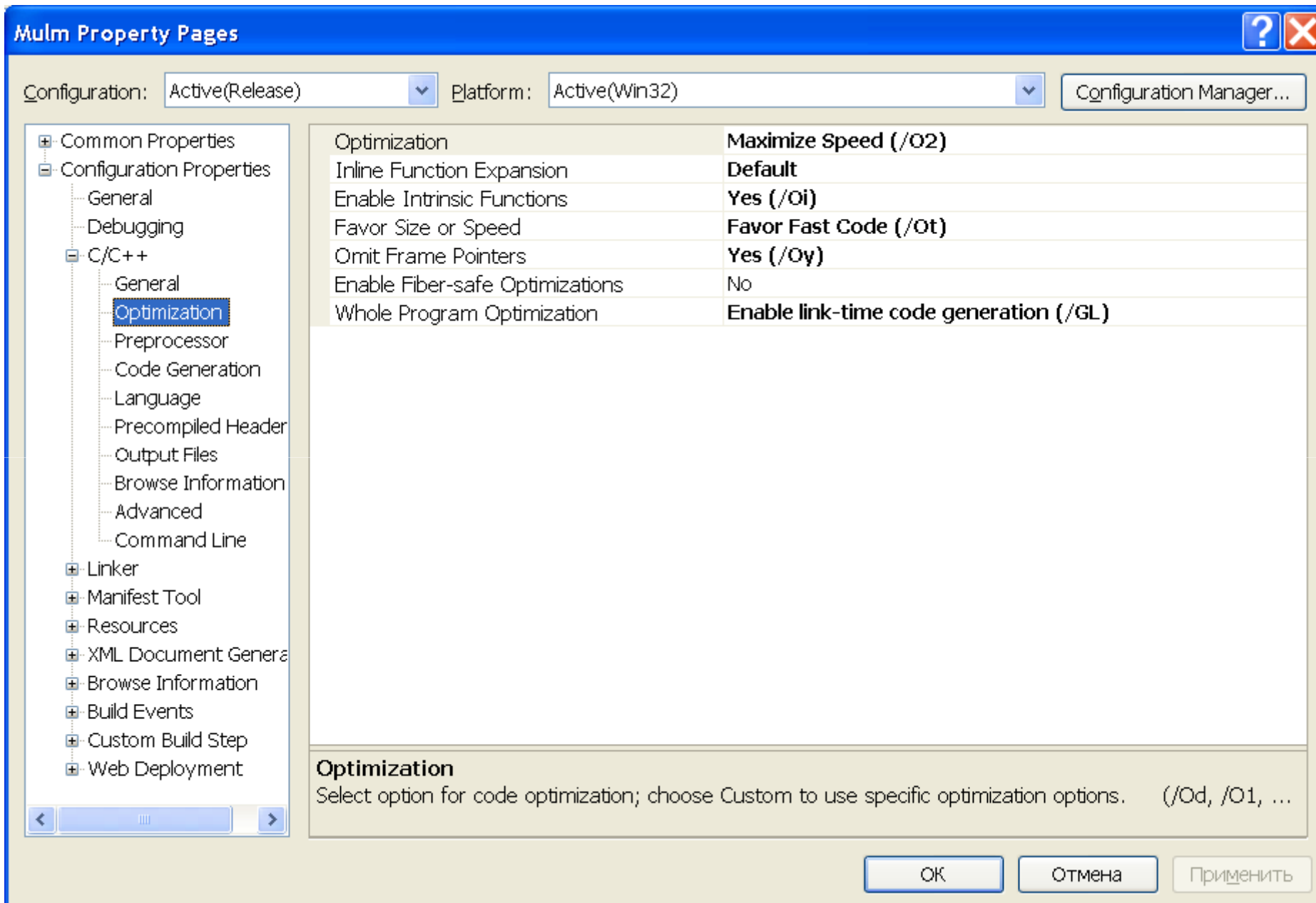
for(j=0; j<dim; j++, ptr_V++)
{
    ptr_R = &R[0];
    for(i=0; i<rest; i++, ptr_A++, ptr_R++)
    {
        *ptr_R += (*ptr_V)*(*ptr_A);
    }

    for(i=rest; i<dim; i+=6, ptr_A+=6, ptr_R+=6)
    {
        *ptr_R   += (*ptr_V)*(*ptr_A);
        *(ptr_R+1) += (*ptr_V)*(*ptr_A+1);
        *(ptr_R+2) += (*ptr_V)*(*ptr_A+2);
        *(ptr_R+3) += (*ptr_V)*(*ptr_A+3);
        *(ptr_R+4) += (*ptr_V)*(*ptr_A+4);
        *(ptr_R+5) += (*ptr_V)*(*ptr_A+5);
    }
}
```

Opcje kompilatora

- **Kompilator Visual Studio 2005 C++ (Visual Studio 8.0), jak i jego poprzedniki (Visual Studio 6.0,) tworzy dwie wersje – debug i release.**
- **Wersja debug jest przeznaczona do debugowania programu, przecież kod wysokiej wydajności można osiągnąć tylko w wersji release.**





➤ Opcje /O1 (minimalizacja rozmiaru kodu), /O2 (maksymalna szybkość) .
Włączenie tej flagi powoduje ustawienie szeregu opcji. Na przykład,
włączenie /O2 powoduje ustawienie opcji: /Og /Oi /Ot /Oy /Ob2 /Gs /GF /Gy

/Og – optymalizacja globalna.

Kod do optymalizacji	Kod po optymalizacji
a = b + c; d = b + c; e = b + c;	tmp = b + c; a = d = e = tmp;

- Optymalizacja lokalna – są przeszukiwane tylko sąsiednie linie kodu, globalna – w obszarze całej funkcji.
- Kluczowe słowo register jest ignorowane. Zamiast tego kompilator umieszcza w rejestry zmienne, które mają częste powtarzania.
- Usunięcie wyrażeń, które nie zależą od indeksu iteracji, poza pętlę:

i = -100; while(i) i += x+y;	i = -100; tmp = x+y; while(i) i += tmp;
------------------------------------	--

- **Optymalizuje działanie konstruktora kopiującego i destruktora przy zwracaniu funkcją wartości.**

/Oi – funkcje wbudowane (fabs(), labs(), strlen(),...) są wywołane jako intrinsic – narzuty przy wywołaniu są usunięte.

/Ot – maksymalizację szybkości wykonania *.exe, *.dll

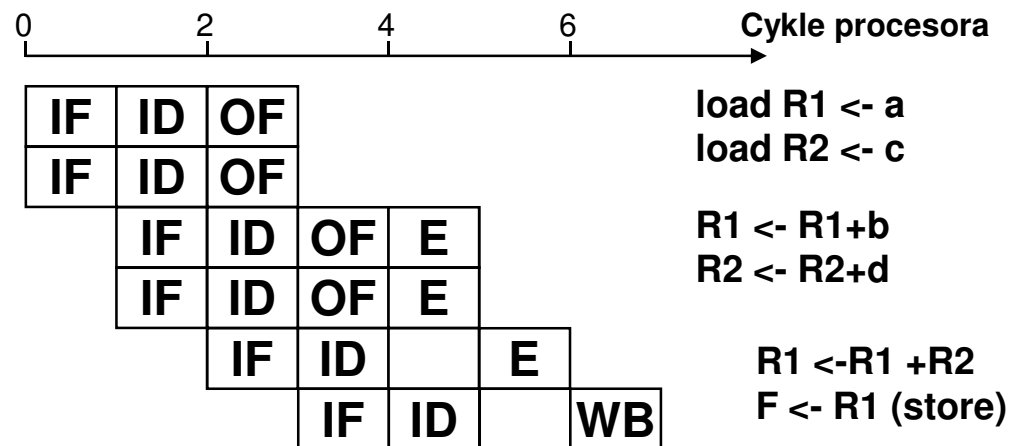
/Oy – przyspiesza działania na stacku przy wywołaniu funkcji, zwalnia jeden lub więcej rejestrów dla umieszczenia często powtarzalnych zmiennych (tylko dla x86).

/Ob2 – rozwinięcie inline funkcji.

.....

Superskalarność procesora – możliwość wykonywania wektoryzacji elementów obliczeń przy technologii potokowej i istnieniu rejestrów specjalnych.

➤ **Przykład: $f = a+b+c+d$;**



IF – instruction fetching

ID – decoding

OF – operand fetching

E – execution

WB – Write-back

Floating point model:

- Domyślnie (bez flagi /fp:presize) kompilator tworzy kod, który dobiera kolejność obliczeń tak, żeby przyspieszyć wykonanie kodu. Przy tym w arytmetyce zmiennoprzecynkowej może się okazać, że kolejność operacji istotnie wpływa na wynik.

- Przykład: $S = S_1 + S_2 + S_3 + S_4 + \dots$. Przy sumowaniu dwóch liczb zmiennoprzecynkowych wyrównanie jest wykonane po wartości bezwzględnej większej liczby: $1000 + 0.01 = 1.0 \times 10^3 + 0.00001 \times 10^3$. Jeśli nasz komputer nie utrzymuje wystarczający rozmiar mantysy, a różnica rzędu tych liczb jest dużą, mniejsza liczba po prostu się gubi. A teraz przedstawimy, że nasz szereg zbiega się wolno, $S_{k+1} < S_k$ ($k=1,2,\dots$), a kompilator tworzy kod:

$S = S_1;$
 $S = S + S_2;$

.....

Zaczynając od jakiegoś indeksu k suma $S \gg S_{k+1}$, i ogonek szeregu będzie pominięty.

- Teraz będziemy liczyć tak:

$S = S_N$
 $S = S + S_{N-1}$

.....

Ogonek szeregu będzie uwzględniony

Ten przykład demonstruje, że bywają zagadnienia, w których wynik istotnie zależy od kolejności operacji, więc **zmiana porządku operacji przy optymalizacji kodu może być dość niebezpieczną.**

Wyjściem z takiej sytuacji może być przedstawienie wyniku pośredniego z większą precyzją. Jeśli typ składowych – float, to wynik poprzedni może być double, jeśli double – long double, i t. p.

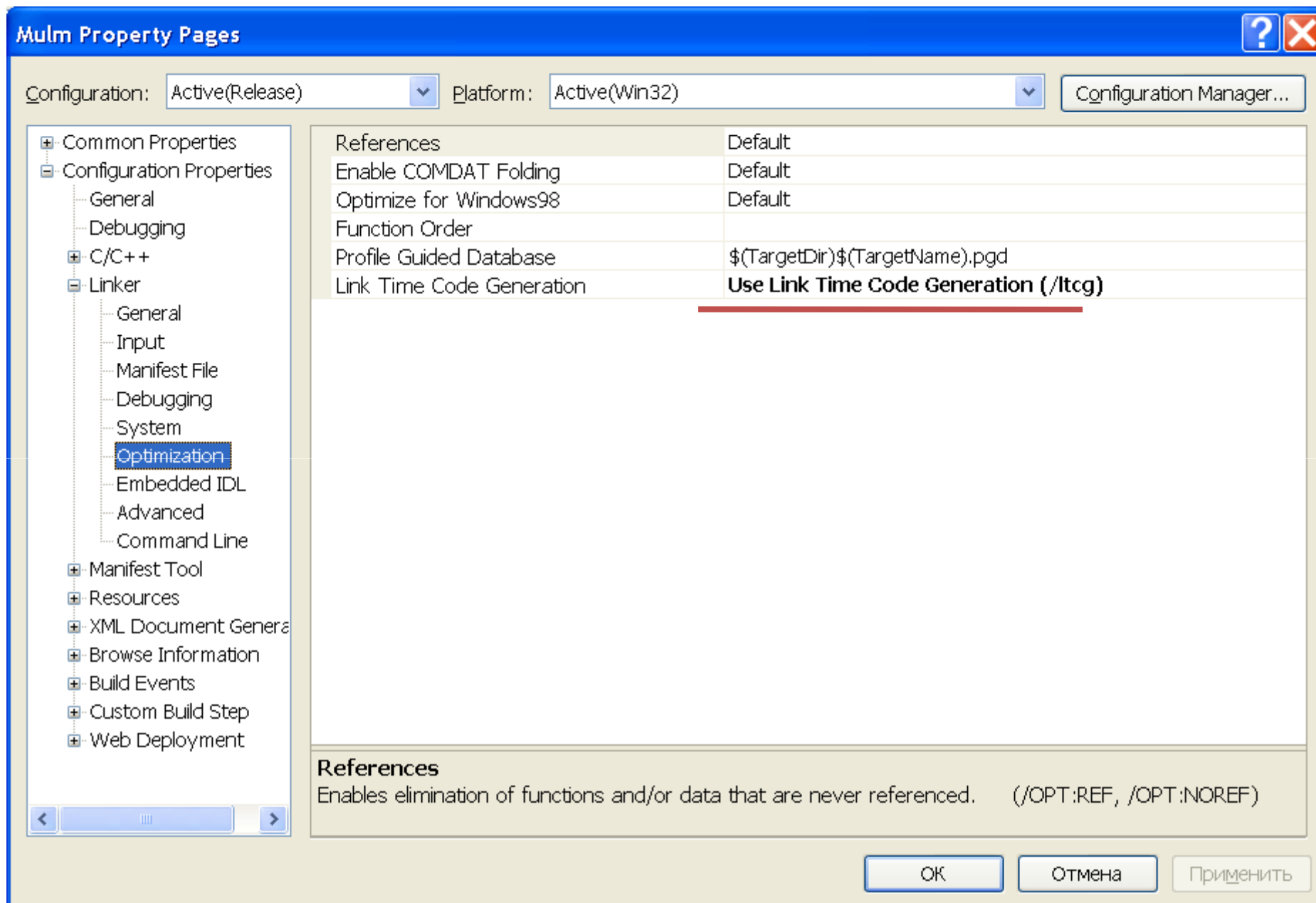
- /fp:precise na procesorach x86 oznacza, że kompilator będzie wykonywał tylko bezpieczną optymalizację na kodzie floating-point, obliczenia pośrednie (intermediate) będą wykonane z największą praktyczną dokładnością. Program, skompilowany z opcją /fp:precise, będzie wolniej i większy od programu, skompilowanego bez tej opcji. Będzie pominięta opcja /Oi (intrinsic functions), zamiast tego standardowa biblioteka runtime routines będzie użyta.
- /fp:fast – tworzy najszybszy kod w większości wypadków. Przecież może to powodować nieoczekiwane wyniki.
- /fp:strict – najbardziej ścisły model operacji zmiennie przyczynkowych. Uwzględnia wszystkie korekcje /fp:precise plus do tego poprzedza nieprawidłowe przetwarzania.

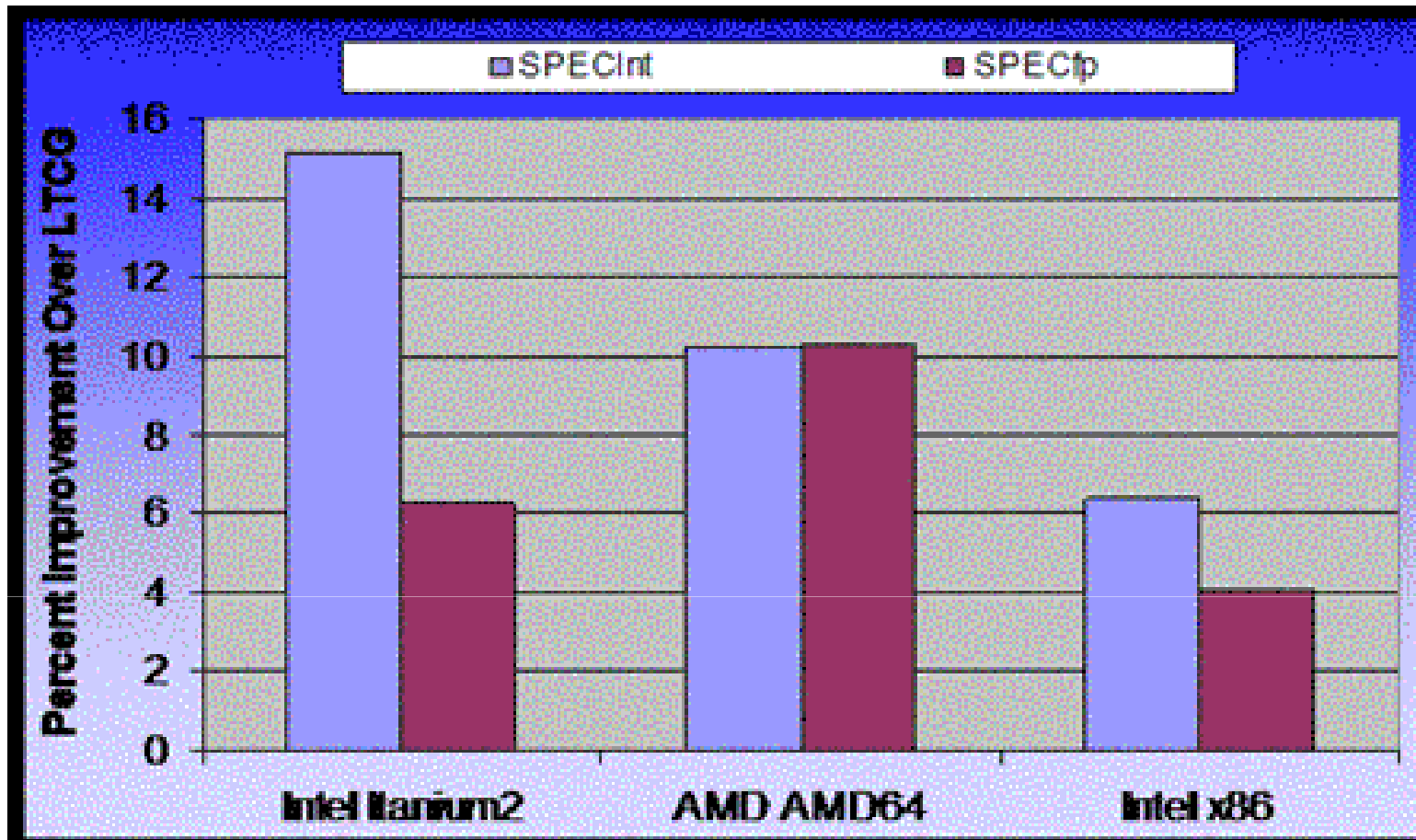
/GL – nadaje możliwość link-time code generation. Jeśli w opcjach linkera jest ustawione (a są ustawienia domyślne dla wersji release), linker i kompilator wspólnie optymalizują kod.

To jest tak zwana technika profile-guided optimization (PGO). Przy zwykłej kompilacji kompilator kompiluje jednostki kompilacji rozdzielnie, i nic nie wie o związku pomiędzy tymi jednostkami. W technice PGO na podstawie wspólnej analizy (kompilator-linker) struktury programu (związku pomiędzy funkcjami i innymi jednostkami kompilacji) jest tworzony kod z optymalizowanym wywołaniem funkcji, a do tego jeszcze wielu rzeczy.

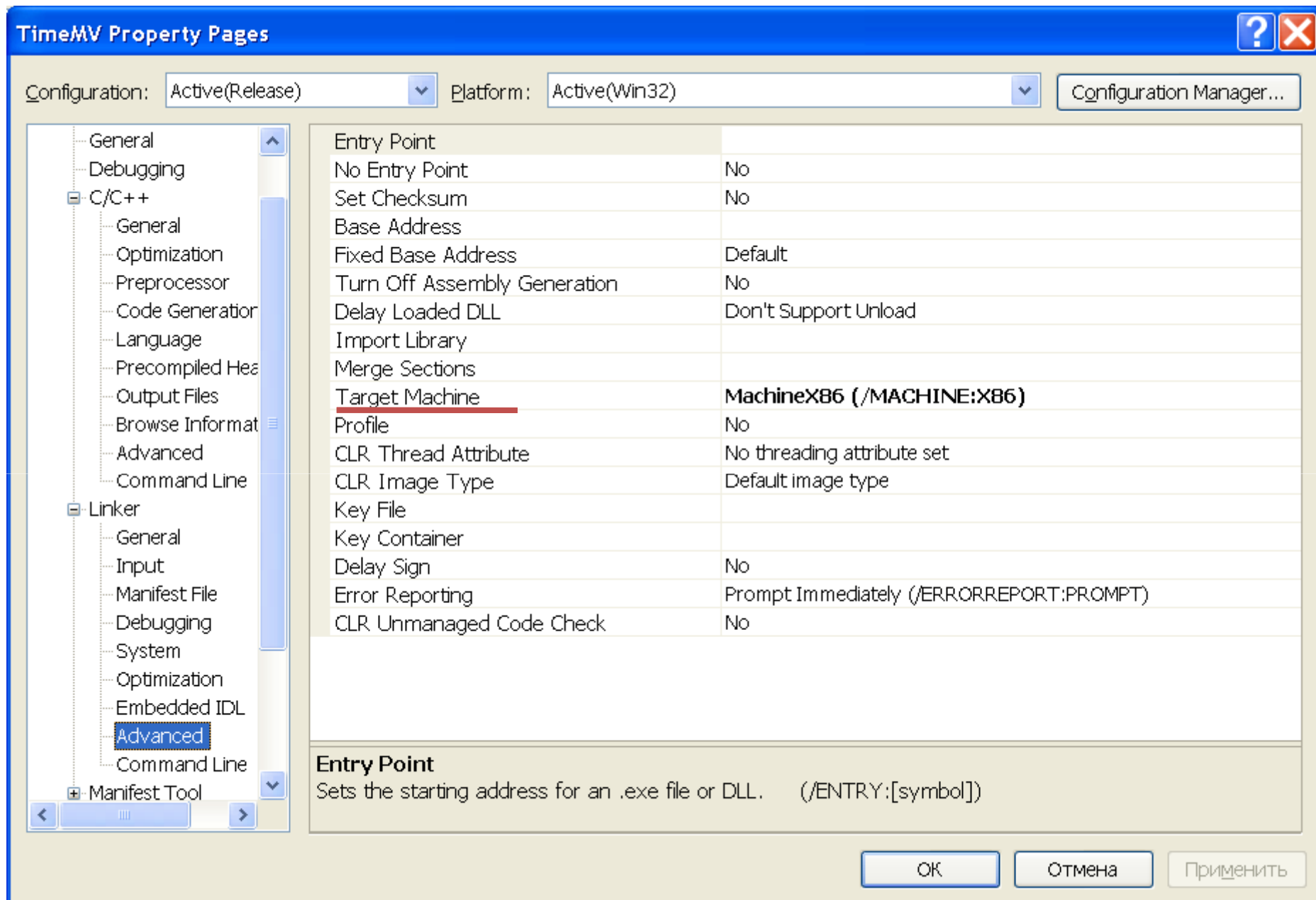
Dla użycia PGO są konieczne etapy:

- **Kompilacje w trybie instrumentation code (opcje Linker/Optimization/Link Time Code Generation/ LTCG:PGINSTRUMENT)**
- **Jednokrotne wykonanie programu ze zbiorem informacji w osobny plik**
- **Przestawienie opcji linkera /LTCG:PGOPTIMIZE i ponowny rebuild programu. Optymalizacja jest wykonana automatycznie na podstawie informacji przy testowym wykonaniu programu.**





Przykład: SPEC performance improvement with PGO over that of Link Time Code Generation, for all three platforms



Fiksuje typ procesora. Jest ustawiony automatycznie na danym komputerze. Kod będzie wygenerowany pod dany typ procesora.