Оптимизация программного обеспечения ЦСП TMS320C67x

Программное обеспечение ЦСП должно удовлетворять двум типовым требованиям: скорость обработки должна быть не меньше требуемой; объем кода программы должен быть не больше допустимого.

Если созданное ПО не удовлетворяет одному из этих требований, то необходима его доработка. Такая доработка называется оптимизацией.

Оптимизация по скорости, как правило, наиболее важна.

Вся программа разбивается на ряд отдельных фрагментов, и измеряется время выполнения каждого из них.

Выделяются один или несколько фрагментов кода, в основном определяющих общие временные затраты.

Выделенные фрагменты кода анализируются более детально и ищутся пути к их более эффективной реализации.

Обычно в основе таких фрагментов лежат циклы.

На процессорах TMS320C67x для оптимизации циклов используются: программная конвейеризация (Software Pipelining) и разворачивание циклов (Loop Unrolling), а также буфер SPLOOP

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

Рассмотрим пример.

Необходимо разработать программу, вычисляющую:

$$y_i = c \times x_i$$

y_i и x_i – массивы размерности N, c – константа.

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

Текст программы:

```
        .global     _c_int00

        .set    N       100
        .set    C       0x40000000

_c_int00:

        MVK     .S2     N,B0
        MVKL    .S2     C,B1
        MVKH    .S2     C,B1
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

```
_LOOP:
;-------------------------------
        LDW             .D1T1  *A7++[1],A1
||      SUB             .S2     B0,1,B0

        NOP             3

        NOP
|[B0]B                  .S1     _LOOP

        MPYSP   .M2X            A1,B1,B2
        NOP             3

        STW     .D2T2   B2,*B7++[1]
;-------------------------------

        NOP
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

Можно считать, что цикл представляет собой последовательность этапов обработки, через которые друг за другом проходят все элементы входного массива данных.

В нашем случае такими этапами являются: выборка из памяти (LDW), умножение (MPYSP) и запись (STW).

Команды LDW, MPYSP и STW образуют путь данных через цикл. Этот путь определяет время выполнения цикла.

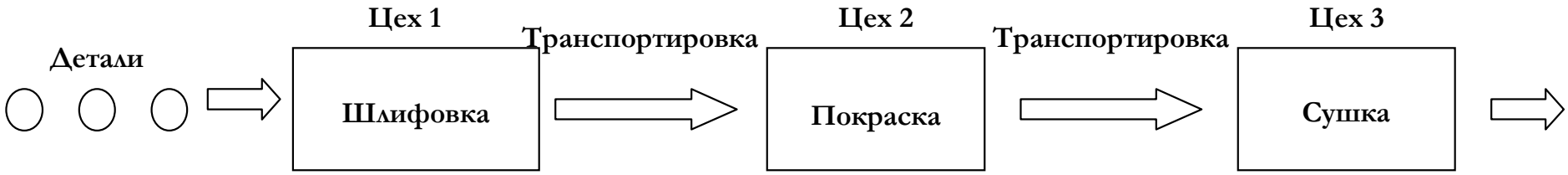
Команды B и SUB являются вспомогательными. Они могут быть выполнены параллельно с основными командами без увеличения длины пути.

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

LDW	**NOP**	**NOP**	**NOP**	**NOP**						**LDW**	**NOP**	**NOP**	**NOP**	**NOP**					
					MPY	**NOP**	**NOP**	**NOP**							**MPY**	**NOP**	**NOP**	**NOP**	
									STW										**STW**

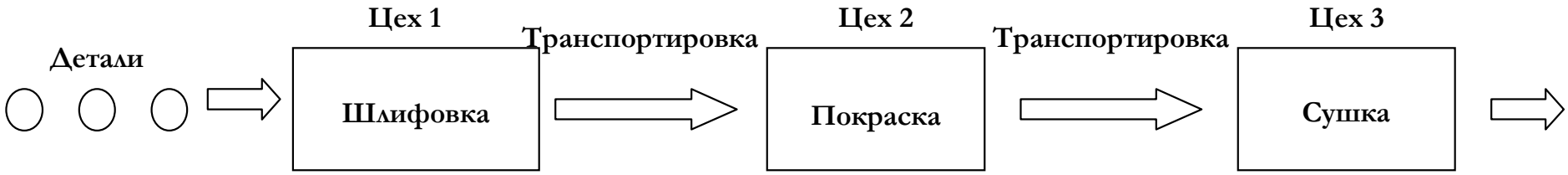
ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

LDW	NOP	NOP	NOP	NOP						LDW	NOP	NOP	NOP	NOP					
					MPY	NOP	NOP	NOP							MPY	NOP	NOP	NOP	
									STW										STW



ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

LDW	**NOP**	**NOP**	**NOP**	**NOP**						**LDW**	**NOP**	**NOP**	**NOP**	**NOP**					
					MPY	**NOP**	**NOP**	**NOP**							**MPY**	**NOP**	**NOP**	**NOP**	
									STW										**STW**



LDW	LDW	LDW	LDW	LDW	LDW	LDW	LDW	LDW	**LDW**	…									
					MPY	MPY	MPY	MPY	**MPY**	…	MPY	MPY	MPY	MPY	MPY				
									STW	…	STW	STW	STW	STW	STW	STW	STW	STW	STW

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

Перечисленные этапы образуют конвейерную обработку последовательности входных данных (программный конвейер).

В представленном варианте программы конвейер работает нерационально: его отдельные этапы постоянно простаивают в ожидании окончания обработки на предыдущих этапах.

Конвейер должен работать более эффективно за счет независимого параллельного функционирования всех его этапов.

Для этого программу следует переписать в следующем виде.

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

```
_LOOP_PROLOG:

...

_LOOP_KERNEL:

;-------------------------------
        LDW     .D1T1  *A7++[1],A1
||      MPYSP   .M2X    A1,B1,B2
||      STW     .D2T2   B2,*B7++[1]
||[B0]SUB     .S2     B0,1,B0
||[B0]B       .S1     _LOOP_KERNEL
;-------------------------------

 _LOOP_EPILOG:

...
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

```
        .global     _c_int00

N       .set    100
C       .set    0x40000000

_c_int00:

        MVK     .S2     N,B0
        MVKL    .S2     C,B1
        MVKH    .S2     C,B1

        SUB     .S2     B0,15,B0
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

```
_LOOP_PROLOG:

        LDW             .D1T1   *A7++[1],A1
        LDW             .D1T1   *A7++[1],A1
        LDW             .D1T1   *A7++[1],A1
        LDW             .D1T1   *A7++[1],A1

        LDW             .D1T1   *A7++[1],A1
||      B               .S1      _LOOP_KERNEL

        LDW             .D1T1   *A7++[1],A1
||      MPYSP           .M2X            A1,B1,B2
||      B               .S1      _LOOP_KERNEL

        LDW             .D1T1   *A7++[1],A1
||      MPYSP           .M2X            A1,B1,B2
||      B               .S1      _LOOP_KERNEL
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

```
        LDW         .D1T1  *A7++[1],A1
||      MPYSP       .M2X         A1,B1,B2
||      B           .S1     _LOOP_KERNEL

        LDW         .D1T1  *A7++[1],A1
||      MPYSP       .M2X         A1,B1,B2
||      B           .S1     _LOOP_KERNEL
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

```
_LOOP_KERNEL:

;-------------------------------
        LDW     .D1T1  *A7++[1],A1
||      MPYSP   .M2X    A1,B1,B2
||      STW     .D2T2   B2,*B7++[1]
||[B0]SUB     .S2     B0,1,B0
||[B0]B       .S1     _LOOP_KERNEL
;-------------------------------
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

```
_LOOP_EPILOG:

        MPYSP   .M2X            A1,B1,B2
||      STW     .D2T2   B2,*B7++[1]

        MPYSP   .M2X            A1,B1,B2
||      STW     .D2T2   B2,*B7++[1]

        MPYSP   .M2X            A1,B1,B2
||      STW     .D2T2   B2,*B7++[1]

        MPYSP   .M2X            A1,B1,B2
||      STW     .D2T2   B2,*B7++[1]

        STW     .D2T2   B2,*B7++[1]
        STW     .D2T2   B2,*B7++[1]
        STW     .D2T2   B2,*B7++[1]
        STW     .D2T2   B2,*B7++[1]
        STW     .D2T2   B2,*B7++[1]
```

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

Все этапы обработки данных выполняются параллельно над разными данными. Ожидание окончания работы предыдущих команд исключается. Тело цикла «сжимается» до одной команды.

Параллельная работа всех этапов остается невозможной при запуске и при остановке конвейера. Появляются пролог и эпилог цикла.

Чем больше итераций в цикле, тем меньшее влияние на время его выполнения оказывают пролог и эпилог.

Оценим выигрыш примененного принципа конвейерной обработки.

ПРОГРАММНАЯ КОНВЕЙЕРИЗАЦИЯ (SOFTWARE PIPELINING)

Для первой реализации программы характерно N-кратное прохождение данных через все этапы обработки за 10 тактов. Поэтому время выполнения цикла обработки не может быть меньше 10*N тактов. В случае N = 100 получаем 1000 тактов.

Для второй реализации имеем: пролог цикла длится 9 тактов, ядро цикла – N-9 тактов, эпилог цикла – 9 тактов. То есть, время выполнения цикла ограничивается снизу значением 9+(N-9)+9. В случае N = 100 это дает 109 тактов выполнения, что соответствует почти 10-кратному ускорению выполнения задачи при оптимизации с использованием принципа программной конвейеризации.

«РАЗВОРАЧИВАНИЕ» ЦИКЛОВ

Принцип «разворачивания» циклов (Loop Unrolling) состоит в трансформации исходного неоптимизированного цикла в новый оптимизированный такой, что за одну итерацию нового цикла производится L итераций исходного цикла.

В простейшем случае такая трансформация основана на том, что процессор TMS320C6701 имеет двойной набор вычислительных блоков.

Пример:

Требуется осуществить N умножений $x_i * h_i$. Исходный цикл условно представляется командой:

MPYSP x_i , h_i

выполняемой N раз.

«РАЗВОРАЧИВАНИЕ» ЦИКЛОВ

Такой цикл выполняется как минимум N тактов.

При использовании второго умножителя процессора TMS320C67x переходим к «развернутому» циклу, представленному командой:

```
        MPYSP x_i,h_i
||      MPYSP x_i+1,h_i+1
```

выполняемой N/2 раз.

Оптимизированный цикл выполняется N/2 тактов. Таким образом, за счет использования принципа «разворачивания» цикла получили двухкратное ускорение процесса вычислений.

SPLOOP

Процессоры TMS320C66x снабжены специальным буфером циклов, куда записываются команды одной итерации цикла.

Специальные аппаратные модули отвечают за выборку команд из этого буфера и позволяют в процессе реализации цикла формировать не только команды ядра цикла, но и команды пролога и эпилога (фрагменты входа в цикл и выхода из него, содержащие не все команды ядра цикла)

```
- - - - - - - - - - - - - - - - - - - - - - - - -
          MVC         8,ILC              ;Do 8 loops
             NOP         3                  ;4 cycle for ILC to load
                SPLOOP      1                  ;Iteration interval is 1
                LDW         *A1++,A2            ;Load source
                NOP    4                       ;Wait for source to load
                MV     .L1X  A2,B2             ;Position data for write
                SPKERNEL     6,0                ;End loop and store value
||               STW         B2,*B0++
```

SPLOOP

Такой подход (обеспечиваемый дополнительными аппаратными компонентами) позволяет получить следующие преимущества.

1.Уменьшается размер программы (за счет отсутствия необходимости полностью расписывать пролог и эпилог).

2.Циклы на основе SPLOOP являются прерываемыми.

3.Поскольку работа с памятью не производится, увеличивается пропускная способность между ядром и памятью; сокращается мощность потребления.

4.Команда перехода становится не нужна. Блок .S может быть задействован под другие нужды.

Оптимизация КИХ-фильтра

```
;-----------
;LOOP KERNEL
;------------
loop_ker:
            LDDW    .D1T1    *A4++[1],A7:A6
||          LDDW    .D2T2    *B4++[1],B7:B6
||          MPYSP   .M1x     A7,B7,A8
||          MPYSP   .M2x     B6,A6,B8
||          ADDSP   .L1      A9,A8,A9
||          ADDSP   .L2      B9,B8,B9
||          [B0]SUB .S2      B0,2,B0
||          [B0]B   .S1      loop_ker
```

Оптимизация КИХ-фильтра

```
;------------
;  PROLOG
;-----------

    MV         .L2x       A6,B0
 || ZERO       .L1        A9
 || ZERO       .S2        B9
 || LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || SUB        .L2        B0,9,B0

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || SUB        .L2        B0,9,B0

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || [B0]SUB    .S2        B0,2,B0

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || [B0]SUB    .S2        B0,2,B0
 || [B0]B      .S1        loop_ker

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || MPYSP      .M1x       A7,B7,A8
 || MPYSP      .M2x       B6,A6,B8
 || [B0]SUB    .S2        B0,2,B0
 || [B0]B      .S1        loop_ker

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || MPYSP      .M1x       A7,B7,A8
 || MPYSP      .M2x       B6,A6,B8
 || [B0]SUB    .S2        B0,2,B0
 || [B0]B      .S1        loop_ker

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || MPYSP      .M1x       A7,B7,A8
 || MPYSP      .M2x       B6,A6,B8
 || [B0]SUB    .S2        B0,2,B0
 || [B0]B      .S1        loop_ker

    LDDW       .D1T1      *A4++[1],A7:A6
 || LDDW       .D2T2      *B4++[1],B7:B6
 || MPYSP      .M1x       A7,B7,A8
 || MPYSP      .M2x       B6,A6,B8
 || [B0]SUB    .S2        B0,2,B0
 || [B0]B      .S1        loop_ker
```

Оптимизация КИХ-фильтра

```
;------------
;   EPILOG
;-----------

    MPYSP    .M1x    A7,B7,A8
 || MPYSP    .M2x    B6,A6,B8
 || FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    MPYSP    .M1x    A7,B7,A8
 || MPYSP    .M2x    B6,A6,B8
 || FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    MPYSP    .M1x    A7,B7,A8
 || MPYSP    .M2x    B6,A6,B8
 || FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    MPYSP    .M1x    A7,B7,A8
 || MPYSP    .M2x    B6,A6,B8
 || FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    MPYSP    .M1x    A7,B7,A8
 || MPYSP    .M2x    B6,A6,B8
 || FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    FADDSP   .L1     A9,A8,A9
 || FADDSP   .L2     B9,B8,B9

    MV       .D1     A9,A5
 || MV       .D2     B9,B5

    FADDSP   .L1     A9,A5,A9
 || FADDSP   .L2     B9,B5,B9

    MV       .D1     A9,A5
 || MV       .D2     B9,B5

    NOP

    FADDSP   .L1     A9,A5,A9
 || FADDSP   .L2     B9,B5,B9

    NOP

    B        .S2     B3

   NOP       2

   FADDSP    .L1x    A9,B9,A4

   NOP       2
```