МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

КАФЕДРА № 32 ЭЛЕКТРОМЕХАНИКИ И РОБОТОТЕХНИКИ

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ: _____

Преподаватель: доцент, к.т.н. _____ В.П. Дашевский

Оптимальные системы

Практическое задание №6

Алгоритм Герцля (Goertzel)

выполнил студент группы 3721

27.12.2020 _____ А. А. Булыгин

Санкт-Петербург, 2020

Содержание

1. Введение

Настоящий документ содержит отчет о выполнении практического задания по созданию и оптимизации программы на языке Си для процессоров семейства C6000. Разработка и отладка алгоритма выполнена в среде программирования Code Composer Studio версии 5.3.

Документ построен следующим образом. В первой части приведено начальное условие задачи.

В разделе 2 приводится описание программы и среды, в которой проводится оптимизация. Приводится текст эталонной программы до оптимизации и результаты его выполнения в симуляторе. Производительность алгоритма оценивается по результатам профилирования кода.

В разделах 3 и 4 представлено описание шагов, предпринятых для оптимизации алгоритма, анализируется обратная связь от компилятора, получаемая в результате этих действий для разной архитектуры ядра.

В разделе 5 представлены основные выводы. Приведено общее сравнение эталонного алгоритма с оптимизированными вариантами на всех предпринятых шагах. Сформулированы выводы, каков общий прирост быстродействия относительно эталонного алгоритма, какой шаг в процессе оптимизации дает наиболее значимый вклад в ускорение программы, и с какой особенностью архитектуры C6000 это связано.

В разделе выводов необходимо дать ответы на общие вопросы:

1. Во сколько раз удалось ускорить алгоритм по сравнению с эталоном?
2. Зависит ли ускорение от архитектуры ядра?
3. Какие дополнительные ограничения целесообразно наложить на входные данные для повышения производительности?

Кроме общих вопросов дать ответы на частные вопросы, посвященные каждому заданию (приведены в при постановке задачи в п.1.1.).

1.1. Постановка задачи оптимизации

Задание №6. Алгоритм Герцля (Goertzel). Написать функцию оценки энергии сигнала на частоте fs при частоте дискретизации f0 на основе алгоритма Герцля[4]. Для упрощения расчетов в целых числах в качестве аргумента функции передается значение косинуса приращения фазы в формате Q15.

```
void goertzel_power(s16 cos_Q15, const s16 *in, int n, s32 *power);
```

- cos_Q15 – косинус (см. ниже псевдокод) в формате Q15.
- in – указатель на массив входных данных
- n – количество данных в массиве
- power – указатель на массив ячеек, куда сохранить оцененную энергию сигнала.

Примечание. Псевдокод алгоритма Герцля для вычисления энергии сигнала:

Листинг 1: Псевдокод алгоритма Герцля

```
f0 – частота дискретизации
fs – частота сигнала
phi := 2 × π × fs / f0;
```

```
cof := 2 × cos(phi)
sprev := 0
sprev2 := 0
for each index i in range 0 to N-1 do
   s := x[i] + cof × sprev – sprev2
   sprev2 := sprev
   sprev := s
end
power := sprev2 × sprev2 + sprev × sprev - cof × sprev × sprev2
```

Для проверки себя на правильность реализации алгоритма сравните получаемую вами энергию с суммой квадратов значений сигнала, являющегося синусом на той же частоте (с таким же приростом фазы). **При правильном расчете обе энергии должны совпасть.** Используйте для расчета команды с насыщением (mpy → smpy, add → sadd и т. п.) Дополнительные вопросы:

1) Исследуйте вопрос, достаточно ли расчета одного алгоритма Герцля, чтобы загрузить все ресурсы процессора?

2) Какие ресурсы процессора определяют быстродействие в данном случае?

3) Сколько потоков алгоритма можно выполнять параллельно так, чтобы это было быстрее, чем последовательно один за другим?

4) Как это зависит от типа процессора (62x, 64x, 64x+)?

1.2. Общие принципы оформления задания

Для более гибкого управления ключами компиляции каждая функция размещается в отдельном файле. Имя файла соответствует имени функции, после которого добавляется суффикс:

- _no_opt.c – для эталонной функции;
- _opt.c – для оптимизированных функций

Файл с суффиксом _no_opt.c компилируется с ключом -O2 (обычная оптимизация) и предоставляет эталонную производительность.

Файл с эталонным алгоритмом компилируется также с ключом -k или другим ключом (например, -os), который позволяет получить обратную связь от компилятора. Обратная связь от компилятора для эталонного алгоритма приводится в разделе 2 после исходного текста эталонной функции.

Файл с оптимизированными функциями компилируется аналогично эталонному, с получением обратной связи от компилятора, но для каждого шага оптимизации. Для удобства исследования и повторения результатов каждый шаг оптимизации оформлен в виде отдельной функции с именем, заканчивающимся суффиксом _opt_N, где N обозначает номер шага оптимизации, и приставкой cxx_, где cxx — серия ядра процессора, например, c62_goertzel_power_opt_2 для шага оптимизации 2 и ядра c62.

2. Базовый алгоритм выполнения задания

2.1. Эталонный алгоритм

Для компактной записи типов и объявления использованных библиотек создадим файл “goertzel_type.h”.

Листинг 2: Заголовочный файл

```
#include <stdio.h>

typedef short s16;
typedef int s32;
```

При объявлении переменной для вычисления энергии сигнала использован тип int, так как при подготовке к вычислению энергии сигнала после цикла происходит нормировка s[i] и s[i-1] к формату Q8.7. При возведении в квадрат чисел данного формата результат занимает 30 бит и укладывается в тип int. Так как входные данные соответствуют типу short, то есть занимают 16 бит со знаком, а величина s объявлена типом int, было принято решение увеличить количество бит, которое занимает значение элемента входного массива с 16 до 24, произведя смещение входных данных типа Q0.15 на 8 бит влево, получив число формата Q0.23, что позволило согласовать входную величину и результат умножения s[i-1] на косинус в формате Q0.15 по количеству бит, выделенному для дробной части. Так как при умножении чисел формата Q0.15 на Q8.23 происходит переполнение ячейки памяти типа int, в которую записывается результат умножения, было решено изменить формат значения s[i-1] перед умножением с Q8.23 на Q8.7, чтобы результат умножения s[i-1] и cos в формате Q0.15 занимал максимум 31 бит. Результат умножения соответствует формату Q8.23.

Листинг 3: Эталонный программа для реализации алгоритма Герцля

```
// Эталонная программа алгоритма Герцля (ядро c62)
#include "goertzel_type.h"

void c62_goertzel_power_no_opt(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int s;
   int i;
   for (i = 0; i < n; i++) {
      int x = (in[i]) << 8;
      // Q0.15 << 8 => Q0.23
      s = 2 * (((sprev) >> 15) * cos_Q15) - sprev1 + x;
      // Q8.23 >> 15 => Q8.8
      // Q8.8 * Q0.15 = Q8.23
      sprev1 = sprev;
      sprev = s;
   }
   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   // Q8.23 >> 16 => Q8.7
   *power = ((sprev1 * sprev1)) + ((sprev * sprev))
```

```
        - (((sprev * cos_Q15) >> 14) * sprev1);
   // Q8.7 * Q8.7 => Q16.14
   // Q8.7 * Q0.15 >> 15 *2 = Q8.7 * Q0.15 >> 14 => Q8.7 * 2
}
```

2.2. Компиляция эталонного алгоритма

2.2.1. Сборка для процессоров семейства C62x

При компиляции исходного текста среда разработки вызывает компилятор со следующими опциями:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6200 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -k --preproc_with_compile
--preproc_dependency="goertzel_no_opt_c62.pp"  "../goertzel_no_opt_c62.c"
```

2.3. Листинг обратной связи компилятора

Примечание: в дальнейшем обратную связь от компилятора будем для краткости называть *фидбэк* в соответствии с англоязычным термином. В качестве фидбэка будем приводить основные части файла обратной связи компилятора.

2.3.1. Листинг для процессоров семейства C62x

Наибольший интерес представляет пролог, ядро и эпилог основного цикла.

Листинг 4: Фидбэк эталонного алгоритма, собранного для C62x

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../goertzel_no_opt_c62.c
;*      Loop source line                : 9
;*      Loop opening brace source line  : 9
;*      Loop closing brace source line  : 17
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 9
;*      Unpartitioned Resource Bound    : 2
;*      Partitioned Resource Bound(*)   : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     3*       1
;*      .D units                     1        0
;*      .M units                     1        1
;*      .X cross paths               1        2
;*      .T address paths             1        0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           1        1     (.L or .S unit)
;*      Addition ops (.LSD)          5        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             2        1
;*      Bound(.L .S .D .LS .LSD)     4*       1
```

```
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 9  Schedule found with 1 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 0
;*      Collapsed prolog stages       : 0
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG
;** ---------------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_goertzel_power_no_opt$4$B:
           SHR     .S1     A3,15,A6          ; |12| <0,0>  ^

           LDH     .D2T2   *B5++,B4          ; |12| <0,1>
||         MPYLH   .M1     A4,A6,A6          ; |12| <0,1>  ^

   [ B0]   SUB     .L2     B0,1,B0           ; |9| <0,2>
||         MPYSU   .M1     A4,A6,A6          ; |12| <0,2>

   [ B0]   B       .S2     $C$L2             ; |9| <0,3>
||         SHL     .S1     A6,16,A7          ; |12| <0,3>  ^

           ADD     .L1     A6,A7,A6          ; |12| <0,4>  ^
           ADD     .L1     A6,A6,A6          ; |12| <0,5>  ^

           MV      .L1     A3,A5             ; |15| <0,6> Define a twin register
||         SHL     .S2     B4,8,B4           ; |12| <0,6>
||         SUB     .S1     A6,A5,A3          ; |12| <0,6>  ^

           ADD     .L2X    B4,A3,B4          ; |12| <0,7>  ^
           MV      .L1X    B4,A3             ; |16| <0,8>  ^
$C$DW$L$_c62_goertzel_power_no_opt$4$E:
;** ---------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG
;** ---------------------------------------------------------------------------*
```

Предварительный анализ функции с эталонным алгоритмом показывает, что:

- Цикл не имеет пролога и эпилога.
- Ядро цикла умещается в 9 пакетов выборки инструкций (fetch packet). Использовано за 9 тактов 19% вычислительных ресурсов ядра, это говорит об изначальной невысокой производительности алгоритма, из-за зависимости расчёта последующих итераций от предыдущих;
- За 1 итерацию ядра цикла обрабатывается 1 элемент массива;
- В информации об организации конвейера наблюдается завязка по данным, равная 9 тактам, в оптимизированной программе стоит попробовать загрузить процессор так, чтобы ограничение завязки по данным было меньше ограничения по количеству вычислительных ресурсов процессора (граф представления завязки по данным изображён на рисунке 1);
- Сторона A более загружена, чем сторона B, это говорит о том, что в оптимизированной программе нужно произвести балансировку ресурсов по

сторонам процессора.

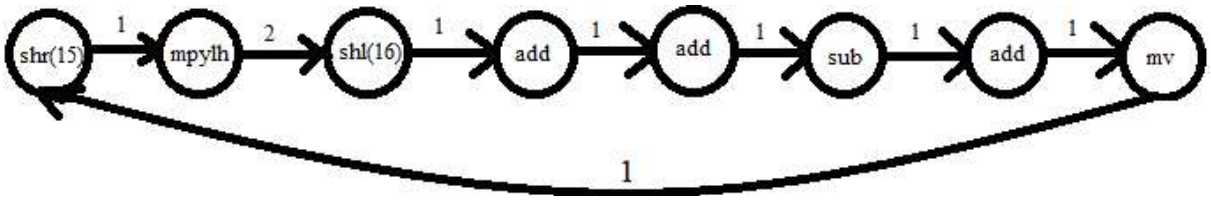

Рисунок 1: Петля зависимости по данным для эталонного алгоритма

Из рисунка следует, что в петле зависимости по данным присутствуют инструкции, такие как, вычитание последовательно со сложением, смещение на 16 бит влево для дублирования младшей части регистра в старшую, а также умножение на 2 с помощью инструкции add (add .L1 A6,A6,A6). Перечисленные инструкции можно упразднить (дублирование и умножение на 2) или сократить время вычисления (последовательное вычитание), в результате чего можно получить петлю завязки по данным в 5 тактов.

3. Создание оптимального алгоритма (C62x)

3.1. Разворачивание цикла

Попробуем развернуть цикл с помощью прагмы MUST_ITERATE для лучшей балансировки ресурсов по сторонам конвейера

Листинг 5: Добавление прагмы кратности 2

```
// Оптимизированная программа алгоритма Герцля (ядро c62)
#include "goertzel_type.h"

void c62_goertzel_power_opt_1(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int s;
   int i;
#pragma MUST_ITERATE(2, 2)
   for (i = 0; i < n; i++) {
      int x = (in[i]) << 8;
      s = 2 * (((sprev) >> 15) * cos_Q15) - sprev1 + x;
      sprev1 = sprev;
      sprev = s;
   }

   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   *power = (sprev1 * sprev1 + sprev * sprev
         - ((sprev * cos_Q15) >> 14) * sprev1);
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6200 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -k --preproc_with_compile
--preproc_dependency="goertzel_opt.pp"  "../goertzel_opt.c"
```

Позволяет компилятору составить следующее расписание:

Листинг 6: Добавление прагмы кратности 2

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../goertzel_opt_c62.c
;*      Loop source line                 : 10
;*      Loop opening brace source line   : 10
;*      Loop closing brace source line   : 15
;*      Known Minimum Trip Count         : 2
;*      Known Max Trip Count Factor      : 2
;*      Loop Carried Dependency Bound(^) : 9
;*      Unpartitioned Resource Bound     : 2
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     3*       1
;*      .D units                     1        0
;*      .M units                     1        1
;*      .X cross paths               1        2
;*      .T address paths             1        0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           1        1     (.L or .S unit)
;*      Addition ops (.LSD)          5        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             2        1
;*      Bound(.L .S .D .LS .LSD)     4*       1
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 9  Schedule found with 1 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 0
;*      Collapsed prolog stages       : 0
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L12:    ; PIPED LOOP PROLOG
;** ----------------------------------------------------------------------------*
$C$L13:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_goertzel_power_opt_1$3$B:
           SHR     .S1     A3,15,A6          ; |12| <0,0>  ^

           LDH     .D2T2   *B5++,B4          ; |12| <0,1>
||         MPYLH   .M1     A4,A6,A6          ; |12| <0,1>  ^

   [ B0]   SUB     .L2     B0,1,B0           ; |10| <0,2>
||         MPYSU   .M1     A4,A6,A6          ; |12| <0,2>

   [ B0]   B       .S2     $C$L13            ; |10| <0,3>
||         SHL     .S1     A6,16,A7          ; |12| <0,3>  ^

           ADD     .L1     A6,A7,A6          ; |12| <0,4>  ^
           ADD     .L1     A6,A6,A6          ; |12| <0,5>  ^

           MV      .L1     A3,A5             ; |13| <0,6> Define a twin register
||         SHL     .S2     B4,8,B4           ; |12| <0,6>
```

```
||         SUB     .S1     A6,A5,A3          ; |12| <0,6>  ^

           ADD     .L2X    B4,A3,B4          ; |12| <0,7>  ^
           MV      .L1X    B4,A3             ; |14| <0,8>  ^
$C$DW$L$_c62_goertzel_power_opt_1$3$E:
;** --------------------------------------------------------------------------*
$C$L14:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Как видим из фидбэка, цикл не разворачивается и пролога и эпилога по-прежнему нет. Прирост производительности составляет 2,1% для массива из 100 элементов:

Name	Calls	Incl Count Average
c62_goertzel_power_no_opt(short, short *, int, int *)	100	1148.00
c62_goertzel_power_opt_1(short, short *, int, int *)	100	1124.19

Рисунок 2: Сравнение эталона с простым разворачиванием цикла

Прирост производительности связан с отсутствием проверки на n=0 в начале функции из-за прагмы MUST_ITERATE(2, ,2). При этом завязка по данным соответствует петле зависимости по данным для эталонного алгоритма.

Укажем компилятору развернуть цикл с помощью прагмы UNROLL:

Листинг 7: Добавление прагмы разворачивания цикла

```
void c62_goertzel_power_opt_1(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int s;
   int i;
#pragma UNROLL(2)
   for (i = 0; i < n; i++) {
      int x = (in[i]) << 8;
      s = 2 * (((sprev) >> 15) * cos_Q15) - sprev1 + x;
      sprev1 = sprev;
      sprev = s;
   }

   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   *power = (sprev1 * sprev1 + sprev * sprev
        - ((sprev * cos_Q15) >> 14) * sprev1);
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6200 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 --speculate_loads=56 -k
--preproc_with_compile --preproc_dependency="goertzel_opt_c62.pp"
"../goertzel_opt_c62.c"
```

Конвейер стал выглядеть следующим образом:

Листинг 8: Добавление прагмы разворачивания цикла

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../goertzel_opt_c62.c
;*      Loop source line                 : 10
;*      Loop opening brace source line   : 10
;*      Loop closing brace source line   : 15
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 17
;*      Unpartitioned Resource Bound     : 5
;*      Partitioned Resource Bound(*)    : 7
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        1
;*      .S units                     3        4
;*      .D units                     2        0
;*      .M units                     2        2
;*      .X cross paths               6        2
;*      .T address paths             1        1
;*      Long read paths              0        0
;*      Long write paths             0        1
;*      Logical  ops (.LS)           4        1     (.L or .S unit)
;*      Addition ops (.LSD)         10        3     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             4        3
;*      Bound(.L .S .D .LS .LSD)     7*       3
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 17 Schedule found with 2 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 1
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 4 bytes
;*      Minimum threshold value       : -mh56
;*
;*      Minimum safe trip count       : 1 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L12:    ; PIPED LOOP PROLOG

           MV      .L1X    B5,A4
||         MV      .S1     A5,A3
||         ADDU    .L2     B0,B8,B7:B6       ; |10| (P) <0,0>
||         LDH     .D2T2   *++B9(4),B4       ; |12| (P) <0,6>
||         MPYLH   .M1     A8,A6,A6          ; |12| (P) <0,6>  ^

;** ---------------------------------------------------------------------------*
$C$L13:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_goertzel_power_opt_1$5$B:

           LDH     .D2T2   *+B9(2),B4        ; |12| <0,7>
||         MPYSU   .M1     A8,A6,A5          ; |12| <0,7>
```

```
           SHL     .S1     A6,16,A6          ; |12| <0,8>  ^
           ADD     .L1     A5,A6,A5          ; |12| <0,9>  ^
           ADD     .L1     A5,A5,A5          ; |12| <0,10>  ^

           SHL     .S2     B4,8,B5           ; |12| <0,11>
||         SUB     .L1     A5,A4,A4          ; |12| <0,11>  ^

           SHL     .S2     B4,8,B5           ; |12| <0,12>
||         ADD     .L2X    B5,A4,B4          ; |12| <0,12>  ^

           MV      .L1X    B4,A4             ; |13| <0,13>
||         SHR     .S2     B4,15,B4          ; |12| <0,13>  ^

           MPYSU   .M1X    A8,B4,A5          ; |12| <0,14>
||         MPYLH   .M2X    A8,B4,B4          ; |12| <0,14>  ^

           NOP             1

           MV      .L2     B6,B0             ; |10| <0,16>
||         SHL     .S2     B4,16,B4          ; |12| <0,16>  ^

           ADD     .S2X    A5,B4,B4          ; |12| <0,17>  ^
||         ADDU    .L2     B0,B8,B7:B6       ; |10| <1,0>

   [ B0]   B       .S1     $C$L13            ; |10| <0,18>
||         ADD     .L2     B4,B4,B4          ; |12| <0,18>  ^

           SUB     .L1X    B4,A3,A3          ; |12| <0,19>  ^
           ADD     .L1X    B5,A3,A5          ; |12| <0,20>  ^
           NOP             1

           MV      .L1     A5,A3             ; |14| <0,22> Split a long life(pre-
sched)
||         SHR     .S1     A5,15,A6          ; |12| <1,5>  ^

           ADD     .L1     2,A7,A7           ; |10| <0,23>
||         LDH     .D2T2   *++B9(4),B4       ; |12| <1,6>
||         MPYLH   .M1     A8,A6,A6          ; |12| <1,6>  ^

$C$DW$L$_c62_goertzel_power_opt_1$5$E:
;** --------------------------------------------------------------------------*
$C$L14:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Как видно из рисунка, теперь за 17 тактов обрабатывается 2 элемента массива, что составляет 8,5 тактов на 1 элемент, а в эталонной программе 1 элемент обрабатывался за 9 тактов. При этом сторона B более загружена в процентном соотношении, по сравнению с программой использующей прагму MUST_ITERATE. На рисунке 3 изображены результаты, полученные в профайлере. В профайлере получили прирост производительности на 3,5% для массива из 100 элементов.

Name	Calls	Incl Count Average
c62_goertzel_power_no_opt(short, short *, int, int *)	100	1148.00
c62_goertzel_power_opt_1(short, short *, int, int *)	100	1109.38

Рисунок 3: Добавление прагмы разворачивания цикла

3.2. Использование интринсиков

Заменим операторы сложения вычитания и умножения на интринсики sadd, ssub и smpy.

Листинг 9: Использование интринсиков

```
#include "goertzel_type.h"

void c62_goertzel_power_opt_2(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int i, s;
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      s = _sadd(_ssub(x, sprev1), _smpy((sprev) >> 15, cos_Q15));
      sprev1 = sprev;
      sprev = s;
   }
   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   *power = (sprev1 * sprev1 + sprev * sprev
         - (_smpy(sprev, cos_Q15) >> 15) * sprev1);
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6200 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -speculate_loads=2 -k
--preproc_with_compile --preproc_dependency="goertzel_opt_c62.pp"
"../goertzel_opt_c62.c"
```

Фидбэк стал выглядеть следующим образом:

Листинг 10: Фидбэк от компилятора для программы, использующей интринсики

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../goertzel_opt_c62.c
;*      Loop source line                : 27
;*      Loop opening brace source line  : 27
;*      Loop closing brace source line  : 32
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 5
;*      Unpartitioned Resource Bound    : 2
;*      Partitioned Resource Bound(*)   : 2
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     2*       0
;*      .S units                     2*       1
;*      .D units                     1        0
;*      .M units                     1        0
;*      .X cross paths               0        0
;*      .T address paths             1        0
;*      Long read paths              0        0
;*      Long write paths             0        0
```

```
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          2        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             2*       1
;*      Bound(.L .S .D .LS .LSD)     3*       1
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 5  Schedule found with 2 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 1
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 2 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L8:    ; PIPED LOOP PROLOG

           MV      .L2X    A7,B4
|| [ A1]   B       .S1     $C$L9             ; |27| (P) <0,3>

;** ---------------------------------------------------------------------------*
$C$L9:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_goertzel_power_opt_2$4$B:
           SHR     .S2     B4,15,B5          ; |29| <0,4>  ^

           SMPY    .M2     B5,B6,B5          ; |29| <0,5>  ^
||         SSHL    .S1     A4,8,A4           ; |29| <0,5>
||         LDH     .D1T1   *A5++,A4          ; |29| <1,0>

           MV      .S1X    B4,A3             ; |30| <0,6>
||         SSUB    .L1     A4,A3,A4          ; |29| <0,6>

           SADD    .L2X    A4,B5,B5          ; |29| <0,7>  ^
|| [ A1]   SUB     .L1     A1,1,A1           ; |27| <1,2>

           MV      .L2     B5,B4             ; |31| <0,8>  ^
|| [ A1]   B       .S1     $C$L9             ; |27| <1,3>

$C$DW$L$_c62_goertzel_power_opt_2$4$E:
;** ---------------------------------------------------------------------------*
$C$L10:    ; PIPED LOOP EPILOG
;** ---------------------------------------------------------------------------*
```

В информации об организации конвейера количество тактов, определяемое петлёй зависимости по данным, равно 5. Компилятору удалось составить расписание с такой завязкой по данным вследствие ускоренного умножения, по сравнению с эталонной программой. Также удалось избавиться от дублирования значения, полученного при умножении в верхнюю половину слова, которое занимало 2 такта. Граф полученной петли завязки по данным изображён на рисунке 4. Длина петли является наименьшей, так как такое расписание соответствует количеству простых операций в алгоритме Герцля (умножение, смещение, сложение, запись значения в переменную).

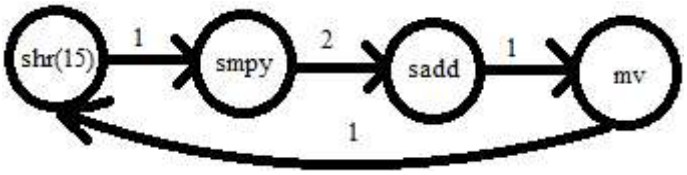

Рисунок 4: Граф петли зависимости по данным для оптимизированной программы

Name	Calls	Incl Count Average
c62_goertzel_power_no_opt(short, short *, int, int *)	100	1148.00
c62_goertzel_power_opt_1(short, short *, int, int *)	100	1107.43
c62_goertzel_power_opt_2(short, short *, int, int *)	100	638.13

Рисунок 5: Сравнение эталона с программой, в которой использованы интринсики

Видим прирост скорости по сравнению с эталоном на 79,9% для массива из 100 элементов. за 5 тактов использовано 10 юнитов процессора, что составляет загрузку конвейера в 25%. Так как остаётся завязка по данным, то можно применить прагму разворачивания цикла для уменьшения влияния завязки по данным.

Листинг 11: Использование интринсиков и прагмы UNROLL

```
#include "goertzel_type.h"

void c62_goertzel_power_opt_2(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int i, s;
#pragma UNROLL(2)
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      s = _sadd(_ssub(x, sprev1), _smpy((sprev) >> 15, cos_Q15));
      sprev1 = sprev;
      sprev = s;
   }
   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   *power = (sprev1 * sprev1 + sprev * sprev
         - (_smpy(sprev, cos_Q15) >> 15) * sprev1);
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6200 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -k -speculate_loads=56
--preproc_with_compile --preproc_dependency="goertzel_opt_c62.pp"
"../goertzel_opt_c62.c"
```

Конвейер стал выглядеть следующим образом:

Листинг 12: Фидбэк от компилятора для программы, использующей интринсики и прагму UNROLL

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../goertzel_opt_c62.c
;*      Loop source line                : 28
;*      Loop opening brace source line  : 28
;*      Loop closing brace source line  : 33
;*      Loop Unroll Multiple            : 2x
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 9
;*      Unpartitioned Resource Bound    : 3
;*      Partitioned Resource Bound(*)   : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3*       2
;*      .S units                     3*       2
;*      .D units                     2        0
;*      .M units                     2        0
;*      .X cross paths               1        1
;*      .T address paths             1        1
;*      Long read paths              0        0
;*      Long write paths             0        1
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          2        3     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             3*       2
;*      Bound(.L .S .D .LS .LSD)     4*       3*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 9  Schedule found with 2 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 1
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 4 bytes
;*      Minimum threshold value       : -mh56
;*
;*      Minimum safe trip count       : 1 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L8:    ; PIPED LOOP PROLOG
           LDH     .D1T1   *+A5(2),A6        ; |30| (P) <0,4>
           NOP             1

           MV      .L1X    B8,A4
||         MV      .S2X    A7,B5
||         ADDU    .L2     B0,B6,B9:B8       ; |28| (P) <0,0>

;** ----------------------------------------------------------------------------*
$C$L9:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_goertzel_power_opt_2$5$B:
           SHR     .S2     B4,15,B7          ; |30| <0,7>  ^

           MV      .L2     B8,B0             ; |28| <0,8>
```

```
||          SMPY    .M2     B7,B5,B7          ; |30| <0,8>  ^
||          SSHL    .S1     A6,8,A7           ; |30| <0,8>

            SSHL    .S1     A6,8,A3           ; |30| <0,9>
||          SSUB    .L1     A7,A3,A6          ; |30| <0,9>
||          ADDU    .L2     B0,B6,B9:B8       ; |28| <1,0>

            SSUB    .L1X    A3,B4,A6          ; |30| <0,10>
|| [ B0]   B       .S1     $C$L9             ; |28| <0,10>
||          SADD    .L2X    A6,B7,B4          ; |30| <0,10>  ^

            MV      .L1X    B4,A3             ; |31| <0,11>
||          SHR     .S2     B4,15,B4          ; |30| <0,11>  ^

            SMPY    .M2     B4,B5,B4          ; |30| <0,12>  ^
||          LDH     .D1T1   *++A5(4),A6       ; |30| <1,3>

            LDH     .D1T1   *+A5(2),A6        ; |30| <1,4>
            SADD    .L2X    A6,B4,B4          ; |30| <0,14>  ^
            ADD     .L1     2,A4,A4           ; |28| <0,15>
$C$DW$L$_c62_goertzel_power_opt_2$5$E:
;** --------------------------------------------------------------------------*
$C$L10:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

В информации об организации конвейера значение количества тактов петли завязки по данным увеличилось до 9 и итерационный интервал стал равен 9, но при этом ядро содержит две итерации исходного цикла. Сторона B более загружена в процентном соотношении по сравнению с программой, не использующей прагму UNROLL. В информации, полученной при профилировании, можно увидеть, что программа получила ускорение на 120,7%.

Name	Calls	Incl Count Average
c62_goertzel_power_no_opt(short, short *, int, int *)	100	1148.00
c62_goertzel_power_opt_1(short, short *, int, int *)	100	1135.61
c62_goertzel_power_opt_2(short, short *, int, int *)	100	520.17

Рисунок 6: Результаты профилирования программы, использующей интринсики и прагму разворачивания цикла

3.3. Параллельное вычисление нескольких мощностей сигнала

Для эталонной и полученных оптимизированных программ остаётся актуальной проблема завязки по данным в 5 тактов, которая связана с тем, что для вычисления следующих итераций нужны результаты, полученные в предыдущих итерациях, из-за чего процессор не может проводить вычисления для нескольких элементов массива параллельно. Для того, чтобы нивелировать данное негативное явление, можно считать несколько мощностей одного сигнала параллельно. Это полезно для выполнения некоторых задач, например, для детектора DTMF сигнала необходимо оценивать энергию 8 частот. Реализуем получение нескольких мощностей для разных частот входного сигнала на ядре C62. На данном ядре удалось максимально быстро посчитать 6 мощностей сигнала с помощью алгоритма Герцля, этого достаточно, чтобы устранить негативное влияние петли

завязки по данным в 5 тактов.

Листинг 13: Параллельное вычисление 6 мощностей

```
#include "goertzel_type.h"

void c62_goertzel_power_opt_3(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power) {
   int s[6], i, k, sprev[6], sprev1[6];
   int cosx[6];
   for (k = 0; k < 6; k++) {
      cosx[k] = cos_Q15[k];
      sprev[k] = 0;
      sprev1[k] = 0;
   }
#pragma MUST_ITERATE(2, ,2)
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      for (k = 0; k < 6; k++) {
         s[k] = _sadd(_ssub(x, sprev1[k]), _smpy((sprev[k]) >> 15, cosx[k]));
         sprev1[k] = sprev[k];
         sprev[k] = s[k];
      }
   }
   for (k = 0; k < 6; k++) {
      sprev[k] = (sprev[k]) >> 16;
      sprev1[k] = (sprev1[k]) >> 16;
      power[k] = (_mpy(sprev1[k], sprev1[k])
            + _mpy(sprev[k], sprev[k])
            - _mpy((_smpy(sprev[k], cosx[k]) >> 15), sprev1[k]));
   }
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6200 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 --speculate_loads=4 -k
--preproc_with_compile --preproc_dependency="goertzel_opt_c62.pp"
"../goertzel_opt_c62.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 14: Фидбэк от компилятора для параллельного вычисления 6 мощностей сигнала

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../goertzel_opt_c62.c
;*      Loop source line                : 49
;*      Loop opening brace source line  : 49
;*      Loop closing brace source line  : 56
;*      Known Minimum Trip Count        : 2
;*      Known Max Trip Count Factor     : 2
;*      Loop Carried Dependency Bound(^) : 5
;*      Unpartitioned Resource Bound    : 6
;*      Partitioned Resource Bound(*)   : 6
;*      Resource Partition:
```

```
;*                                A-side   B-side
;*      .L units                     6*       6*
;*      .S units                     4        4
;*      .D units                     1        0
;*      .M units                     3        3
;*      .X cross paths               0        3
;*      .T address paths             1        0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          7        6     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             5        5
;*      Bound(.L .S .D .LS .LSD)     6*       6*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 6  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      Collapsed epilog stages       : 1
;*
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 2 bytes
;*      Minimum threshold value       : -mh4
;*
;*      Minimum safe trip count       : 2
;*----------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG AND PROLOG

           STW     .D2T2   B6,*B5++          ; |45| (E) <1,5>
||         LDH     .D1T1   *A5++,A3          ; |44| (E) <3,1>

           STW     .D2T2   B6,*B4++          ; |46| (E) <1,6>
||         STW     .D1T1   A3,*A4++          ; |44| (E) <1,6>

           STW     .D2T2   B6,*B5++          ; |45| (E) <2,5>

           STW     .D2T2   B6,*B4++          ; |46| (E) <2,6>
||         STW     .D1T1   A3,*A4++          ; |44| (E) <2,6>

           MVC     .S2     B8,CSR            ; interrupts on
||         STW     .D2T2   B6,*B5++          ; |45| (E) <3,5>

           STW     .D2T2   B6,*B4++          ; |46| (E) <3,6>
||         STW     .D1T1   A3,*A4++          ; |44| (E) <3,6>

           LDW     .D2T2   *+SP(20),B13
           LDW     .D2T2   *+SP(72),B2
           LDW     .D2T1   *+SP(68),A10
           LDW     .D2T1   *+SP(60),A11
           LDW     .D2T2   *+SP(56),B10
           LDW     .D2T2   *+SP(64),B3
           LDW     .D2T1   *+SP(52),A13
           LDW     .D2T2   *+SP(24),B7
           LDW     .D2T2   *+SP(44),B11
           LDW     .D2T2   *+SP(48),B6
```

```
           LDW     .D2T2   *+SP(40),B8
           LDW     .D2T1   *+SP(36),A8
           LDW     .D2T2   *+SP(12),B5
           LDW     .D2T2   *+SP(16),B9
           LDW     .D2T2   *+SP(32),B0
           LDW     .D2T2   *+SP(28),B4
           LDW     .D2T2   *+SP(4),B12
           LDW     .D2T2   *+SP(8),B1
   .dwpsn  file "../goertzel_opt_c62.c",line 50,column 7,is_stmt

           SUB     .S1     A1,2,A1
||         MV      .L1X    B13,A7
||         MVC     .S2     CSR,B13
||         LDH     .D1T1   *A12++,A3         ; |52| (P) <0,0>

           MV      .L1X    B5,A9
||         AND     .L2     -2,B13,B5

           MV      .L1X    B4,A0
||         MVC     .S2     B5,CSR            ; interrupts off

           MV      .L1X    B12,A2

           MV      .L1X    B11,A6
||         SHR     .S2     B1,15,B12         ; |52| (P) <0,6>  ^
||         LDH     .D1T1   *A12++,A5         ; |52| (P) <1,0>
||         SHR     .S1     A7,15,A4          ; |52| (P) <0,6>  ^

           SMPY    .M2     B12,B10,B5        ; |52| (P) <0,7>  ^
||         SMPY    .M1     A4,A10,A5         ; |52| (P) <0,7>  ^
||         SSHL    .S1     A3,8,A14          ; |52| (P) <0,7>

           SHR     .S2     B7,15,B4          ; |52| (P) <0,8>  ^
||         SHR     .S1     A2,15,A3          ; |52| (P) <0,8>  ^
||         SSUB    .L1     A14,A8,A4         ; |52| (P) <0,8>
||         SSUB    .L2X    A14,B0,B11        ; |52| (P) <0,8>

;** --------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_goertzel_power_opt_3$5$B:

           SHR     .S1     A9,15,A15         ; |52| <0,9>  ^
||         MV      .D2     B1,B0             ; |53| <0,9>
||         SMPY    .M2     B4,B2,B4          ; |52| <0,9>  ^
|| [ A1]   B       .S2     $C$L4             ; |49| <0,9>
||         SADD    .L2     B11,B5,B12        ; |52| <0,9>  ^
||         SMPY    .M1     A3,A13,A15        ; |52| <0,9>  ^
||         MV      .D1     A7,A6             ; |53| <0,9>
||         SSUB    .L1     A14,A6,A3         ; |52| <0,9>

           MV      .D1     A9,A8             ; |53| <0,10>
||         SMPY    .M1     A15,A11,A14       ; |52| <0,10>  ^
||         MV      .D2     B7,B6             ; |53| <0,10>
||         SSUB    .L2X    A14,B6,B5         ; |52| <0,10>
||         SHR     .S2     B9,15,B11         ; |52| <0,10>  ^
||         SADD    .L1     A3,A5,A3          ; |52| <0,10>  ^
|| [ A1]   SUB     .S1     A1,1,A1           ; |49| <1,4>
```

```
            MV      .S2     B9,B8           ; |53| <0,11>
||          SSUB    .L2X    A14,B8,B11      ; |52| <0,11>
||          MV      .D2     B12,B1          ; |54| <0,11>  ^
||          SMPY    .M2     B11,B3,B5       ; |52| <0,11>  ^
||          MV      .S1     A2,A0           ; |53| <0,11> Define a twin register
||          SSUB    .L1     A14,A0,A3       ; |52| <0,11>
||          MV      .D1     A3,A7           ; |54| <0,11>  ^

            SADD    .L2     B5,B4,B4        ; |52| <0,12>  ^
||          SADD    .L1     A3,A15,A15      ; |52| <0,12>  ^
||          SHR     .S2     B1,15,B12       ; |52| <1,6>  ^
||          SHR     .S1     A7,15,A3        ; |52| <1,6>  ^
||          LDH     .D1T1   *A12++,A5       ; |52| <2,0>

            SADD    .L1     A4,A14,A3       ; |52| <0,13>  ^
||          MV      .S2     B4,B7           ; |54| <0,13>  ^
||          SADD    .L2     B11,B5,B4       ; |52| <0,13>  ^
||          MV      .D1     A15,A2          ; |54| <0,13>  ^
||          SMPY    .M2     B12,B10,B5      ; |52| <1,7>  ^
||          SMPY    .M1     A3,A10,A5       ; |52| <1,7>  ^
||          SSHL    .S1     A5,8,A14        ; |52| <1,7>

            MV      .D1     A3,A9           ; |54| <0,14>  ^
||          MV      .D2     B4,B9           ; |54| <0,14>  ^
||          SHR     .S2     B7,15,B4        ; |52| <1,8>  ^
||          SHR     .S1     A2,15,A3        ; |52| <1,8>  ^
||          SSUB    .L1     A14,A8,A4       ; |52| <1,8>
||          SSUB    .L2X    A14,B0,B11      ; |52| <1,8>

$C$DW$L$_c62_goertzel_power_opt_3$5$E:
;*--------------------------------------------------------------------------*
```

Как видим, параллельное вычисление нескольких мощностей позволяет компилятору составить расписание, в котором ii определяется не завязкой по данным, а требуемыми вычислительными ресурсами процессора. За 6 тактов выполняется 12 инструкций на .L юнитах. Ядро цикла загружено на 40/48=83,3%.

Name	Calls	Incl Count Average	Incl Count Total
c62_goertzel_power_no_opt(short, short *, int, int *)	600	1137.17	682303
c62_goertzel_power_opt_1(short, short *, int, int *)	100	1134.61	113461
c62_goertzel_power_opt_2(short, short *, int, int *)	100	586.26	58626
c62_goertzel_power_opt_3(short *, short *, int, int *)	100	734.50	73450

Рисунок 7: Результаты профайлера для программы, рассчитывающей 6 алгоритмов Герцля параллельно

В итоге количество тактов в полученной программе, производящей обработку входного массива из 100 элементов, меньше в среднем на 402 такта, но при этом эталонная программа за 1137,17 тактов считает только одну мощность. Если полное количество тактов разделить не на 600, а на 100, то получим 682303/100=6823,03 тактов на последовательный расчёт 6 мощностей, в то время как параллельный расчёт 6 мощностей требует 734,5 тактов. Таким образом, прирост производительности составляет 928,94%.

4. Создание оптимального алгоритма (C64x)

4.1. Запуск программы, вычисляющей 6 мощностей параллельно, на ядре C64

Проведем отладку программы из пункта 3.3 и посмотрим, насколько быстро программа работает на ядре C64.

Листинг 15: Программа для параллельного вычисления 6 мощностей сигнала

```
#include "goertzel_type.h"

void c64_goertzel_power_opt_1(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power) {
   int s[6], i, k, sprev[6], sprev1[6];
   short cosx[6];
   for (k = 0; k < 6; k++) {
      cosx[k] = cos_Q15[k];
      sprev[k] = 0;
      sprev1[k] = 0;
   }
#pragma MUST_ITERATE(2, ,2)
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      for (k = 0; k < 6; k++) {
         s[k] = _sadd(_ssub(x, sprev1[k]), _smpy((sprev[k]) >> 15, cosx[k]));
         sprev1[k] = sprev[k];
         sprev[k] = s[k];
      }
   }
   for (k = 0; k < 6; k++) {
      sprev[k] = (sprev[k]) >> 16;
      sprev1[k] = (sprev1[k]) >> 16;
      power[k] = (_mpy(sprev1[k], sprev1[k])
            + _mpy(sprev[k], sprev[k])
            - _mpy((_smpy(sprev[k], cosx[k]) >> 15), sprev1[k]));
   }
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6400 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -k --preproc_with_compile
--preproc_dependency="goertzel_opt_c64.pp"  "../goertzel_opt_c64.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 16: Фидбэк от компилятора для вычисления 6 мощностей параллельно

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../goertzel_opt_c64.c
;*      Loop source line                : 13
;*      Loop opening brace source line  : 13
```

```
;*      Loop closing brace source line   : 20
;*      Known Minimum Trip Count         : 2
;*      Known Max Trip Count Factor      : 2
;*      Loop Carried Dependency Bound(^) : 5
;*      Unpartitioned Resource Bound     : 6
;*      Partitioned Resource Bound(*)    : 6
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3        3
;*      .S units                     5        3
;*      .D units                     1        0
;*      .M units                     3        3
;*      .X cross paths               0        3
;*      .T address paths             1        0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           3        3     (.L or .S unit)
;*      Addition ops (.LSD)          6        6     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             6*       5
;*      Bound(.L .S .D .LS .LSD)     6*       5
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 6  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*      Collapsed prolog stages       : 2
;*      Minimum required memory pad   : 0 bytes
;*
;*      Minimum safe trip count       : 2
;*--------------------------------------------------------------------------*
$C$L6:    ; PIPED LOOP PROLOG
;** --------------------------------------------------------------------------*
$C$L7:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_goertzel_power_opt_1$3$B:

   [!A1]   MV      .D1     A7,A8            ; |17| <0,9>
|| [!A1]   SMPY    .M1     A17,A18,A3       ; |16| <0,9>  ^
|| [ A0]   BDEC    .S1     $C$L7,A0         ; |13| <0,9>
|| [!A1]   SADD    .L1     A6,A3,A24        ; |16| <0,9>  ^
|| [!A1]   MV      .S2     B8,B9            ; |17| <0,9>
|| [!A1]   SMPY    .M2     B4,B7,B21        ; |16| <0,9>  ^
|| [!A1]   SSUB    .L2X    A21,B9,B4        ; |16| <0,9>

   [!A1]   MV      .D1     A4,A5            ; |17| <0,10>
|| [!A1]   SSUB    .L1     A21,A5,A20       ; |16| <0,10>
|| [!A1]   SADD    .S1     A20,A3,A17       ; |16| <0,10>  ^
|| [!A1]   MV      .D2     B16,B17          ; |17| <0,10>
|| [!A1]   SADD    .S2     B4,B21,B4        ; |16| <0,10>  ^
|| [!A1]   SSUB    .L2X    A21,B17,B22      ; |16| <0,10>

   [!A1]   MV      .D1     A24,A9           ; |18| <0,11>  ^
|| [!A1]   SADD    .L1     A20,A3,A3        ; |16| <0,11>  ^
|| [!A1]   MV      .D2     B4,B8            ; |18| <0,11>  ^
|| [!A1]   SSUB    .L2X    A21,B19,B20      ; |16| <0,11>
|| [!A1]   SADD    .S2     B22,B20,B4       ; |16| <0,11>  ^
|| [!B0]   SSHL    .S1     A6,8,A21         ; |16| <1,5>
```

```
   [ A1]   MPYSU   .M1     2,A1,A1           ; <0,12>
|| [!A1]   MV      .L1     A17,A7            ; |18| <0,12>  ^
|| [!A1]   SADD    .L2     B20,B21,B20       ; |16| <0,12>  ^
|| [!A1]   MV      .D2     B4,B16            ; |18| <0,12>  ^
|| [!B0]   SHR     .S1     A9,15,A6          ; |16| <1,6>  ^
|| [!B0]   SHR     .S2     B8,15,B4          ; |16| <1,6>  ^
||         LDH     .D1T1   *A23++,A6         ; |16| <2,0>

   [!A1]   MV      .D1     A3,A4             ; |18| <0,13>  ^
|| [!A1]   MV      .L2     B18,B19           ; |17| <0,13>
|| [!A1]   MV      .D2     B20,B18           ; |18| <0,13>  ^
|| [!B0]   SSUB    .L1     A21,A8,A20        ; |16| <1,7>
|| [!B0]   SMPY    .M1     A6,A22,A3         ; |16| <1,7>  ^
|| [!B0]   SHR     .S1     A7,15,A3          ; |16| <1,7>  ^
|| [!B0]   SMPY    .M2     B4,B5,B21         ; |16| <1,7>  ^
|| [!B0]   SHR     .S2     B16,15,B4         ; |16| <1,7>  ^

   [ B0]   SUB     .L2     B0,1,B0           ; <0,14>
|| [!B0]   MV      .D1     A9,A16            ; |17| <1,8> Define a twin register
|| [!B0]   SMPY    .M1     A3,A19,A3         ; |16| <1,8>  ^
|| [!B0]   SHR     .S1     A4,15,A17         ; |16| <1,8>  ^
|| [!B0]   SSUB    .L1     A21,A16,A6        ; |16| <1,8>
|| [!B0]   SHR     .S2     B18,15,B4         ; |16| <1,8>  ^
|| [!B0]   SMPY    .M2     B4,B6,B20         ; |16| <1,8>  ^

$C$DW$L$_c64_goertzel_power_opt_1$3$E:
;*--------------------------------------------------------------------------*
```

Можно отметить, что в ядре цикла на 1 инструкцию больше, по сравнению с ядром C62.

Эталонная программа для ядра C64:

Листинг 17: Эталонный программа для реализации алгоритма Герцля

```c
// Эталонная программа алгоритма Герцля (ядро c64)
#include "goertzel_type.h"

void c64_goertzel_power_no_opt(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int s;
   int i;
   for (i = 0; i < n; i++) {
      int x = (in[i]) << 8;
      // Q0.15 << 8 => Q0.23
      s = 2 * (((sprev) >> 15) * cos_Q15) - sprev1 + x;
      // Q8.23 >> 15 => Q8.8
      // Q8.8 * Q0.15 = Q8.23
      sprev1 = sprev;
      sprev = s;
   }
   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   // Q8.23 >> 16 => Q8.7
   *power = ((sprev1 * sprev1)) + ((sprev * sprev))
         - (((sprev * cos_Q15) >> 14) * sprev1);
   // Q8.7 * Q8.7 => Q16.14
```

```
   // Q8.7 * Q0.15 >> 15 *2 = Q8.7 * Q0.15 >> 14 => Q8.7 * 2
}
```

Компиляции с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6400 --abi=coffabi -O2
-g --include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -k --preproc_with_compile
--preproc_dependency="goertzel_no_opt_c64.pp"  "../goertzel_no_opt_c64.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 18: Фидбэк от компилятора для эталонной программы на ядре C64

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../goertzel_no_opt_c64.c
;*      Loop source line                 : 9
;*      Loop opening brace source line   : 9
;*      Loop closing brace source line   : 17
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 9
;*      Unpartitioned Resource Bound     : 2
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     2        1
;*      .D units                     1        0
;*      .M units                     1        0
;*      .X cross paths               0        0
;*      .T address paths             1        0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          5        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        1
;*      Bound(.L .S .D .LS .LSD)     3*       1
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 9  Schedule found with 1 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 0
;*      Collapsed prolog stages       : 0
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG
;** ----------------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_goertzel_power_no_opt$4$B:
          SHR     .S1     A4,15,A7          ; |12| <0,0>  ^

          LDH     .D1T1   *A3++,A7          ; |12| <0,1>
||        MPYLI   .M1     A5,A7,A9:A8       ; |12| <0,1>  ^
```

```
           NOP             1
   [ B0]   BDEC    .S2     $C$L2,B0          ; |9| <0,3>
           NOP             1
           ADD     .L1     A8,A8,A8          ; |12| <0,5>  ^

           MV      .L1     A4,A6             ; |15| <0,6> Define a twin register
||         SHL     .S1     A7,8,A7           ; |12| <0,6>
||         SUB     .D1     A8,A6,A4          ; |12| <0,6>  ^

           ADD     .L1     A7,A4,A4          ; |12| <0,7>  ^
           NOP             1
$C$DW$L$_c64_goertzel_power_no_opt$4$E:
;** --------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

c64_goertzel_power_no_opt(short, short *, int, int *)	600	934.84	560904
c64_goertzel_power_opt_1(short *, short *, int, int *)	100	708.44	70844

Рисунок 9: Запуск оптимизированной программы для расчёта 6 мощностей на процессоре с ядром C64

Из рисунка 9 следует, что эталонная программа на ядре C64 работает быстрее, чем эталонная программа на ядре C62. Это связано с использованием компилятором команд, позволяющих производить расчёт параметров в оверхэде быстрее, но при этом итерационный интервал остался равен 9 тактам, как и петля завязки по данным, что видно из фидбэка эталонной программы. Для расчёта 6 мощностей, соответствующих входному массиву из 100 элементов, эталонной программе нужно в среднем 560904/100=5609,04 такта, а оптимизированной программе для это нужно в среднем 708,44 такта, оптимизированная программа быстрее на 691,75%. При этом оптимизированная программа работает быстрее на 3,68% для обработки массива из 100 элементов, по сравнению с этой же программой на ядре C62, это связано с тем, что не ускоряемая часть программы на ядре C64 вычисляется быстрее, так как ядра циклов для разных процессоров практически аналогичны, за исключением инструкции BDEC.

4.2. Расчёт 8 мощностей сигнала параллельно

Так как для детектор DTMF сигнала построен на анализе 8 базовых частот, то на ядре C64 имеет смысл попробовать посчитать 8 мощностей сигнала с помощью алгоритма Герцля параллельно, для ядра C62 эта задача не была решена.

Листинг 19: Программа, считающая 8 мощностей сигнала параллельно

```
#include "goertzel_type.h"

void c64_goertzel_power_opt_2(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power) {
   int s[8], i, k, sprev[8], sprev1[8];
   short cosx[8];
   for (k = 0; k < 8; k++) {
      cosx[k] = cos_Q15[k];
      sprev[k] = 0;
```

```
      sprev1[k] = 0;
   }
#pragma MUST_ITERATE(2, ,2)
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      for (k = 0; k < 8; k++) {
         s[k] = _sadd(_ssub(x, sprev1[k]), _smpy((sprev[k]) >> 15, cosx[k]));
         sprev1[k] = sprev[k];
         sprev[k] = s[k];
      }
   }
   for (k = 0; k < 8; k++) {
      sprev[k] = (sprev[k]) >> 16;
      sprev1[k] = (sprev1[k]) >> 16;
      power[k] = (_mpy(sprev1[k], sprev1[k])
            + _mpy(sprev[k], sprev[k])
            - _mpy((_smpy(sprev[k], cosx[k]) >> 15), sprev1[k]));
   }
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6400 --abi=coffabi -O2 -g
--include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -k
--preproc_dependency="goertzel_opt_c64.pp"  "../goertzel_opt_c64.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 20: Фидбэк программы, рассчитывающий 8 мощностей сигнала параллельно

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../goertzel_opt_c64.c
;*      Loop source line                 : 39
;*      Loop opening brace source line   : 39
;*      Loop closing brace source line   : 46
;*      Known Minimum Trip Count         : 2
;*      Known Max Trip Count Factor      : 2
;*      Loop Carried Dependency Bound(^) : 5
;*      Unpartitioned Resource Bound     : 8
;*      Partitioned Resource Bound(*)    : 8
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     4        4
;*      .S units                     6        4
;*      .D units                     1        0
;*      .M units                     4        4
;*      .X cross paths               0        4
;*      .T address paths             1        0
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           4        4     (.L or .S unit)
;*      Addition ops (.LSD)          8        8     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             7        6
;*      Bound(.L .S .D .LS .LSD)     8*       7
;*
```

```
;*      Searching for software pipeline schedule at ...
;*         ii = 8  Schedule found with 2 iterations in parallel
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*      Collapsed prolog stages       : 1
;*      Minimum required memory pad   : 0 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG
;** ---------------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_goertzel_power_opt_2$3$B:

   [!B0]   MV      .D1     A5,A4             ; |43| <0,6>
|| [!B0]   SHR     .S1     A9,15,A24         ; |42| <0,6>  ^
|| [!B0]   SSUB    .L1     A3,A4,A23         ; |42| <0,6>
|| [!B0]   SHR     .S2     B19,15,B4         ; |42| <0,6>  ^
|| [!B0]   SMPY    .M2     B4,B9,B16         ; |42| <0,6>  ^

   [!B0]   SHR     .S1     A17,15,A24        ; |42| <0,7>  ^
|| [!B0]   SMPY    .M1     A24,A20,A27       ; |42| <0,7>  ^
|| [!B0]   MV      .D1     A7,A6             ; |43| <0,7>
|| [!B0]   SSUB    .L1     A3,A6,A27         ; |42| <0,7>
|| [!B0]   SMPY    .M2     B4,B17,B4         ; |42| <0,7>  ^
|| [!B0]   SHR     .S2     B21,15,B4         ; |42| <0,7>  ^
|| [!B0]   MV      .D2     B23,B22           ; |43| <0,7>
|| [!B0]   SSUB    .L2X    A3,B22,B5         ; |42| <0,7>

   [!B0]   SMPY    .M1     A24,A22,A26       ; |42| <0,8>  ^
|| [!B0]   SADD    .L1     A27,A26,A24       ; |42| <0,8>  ^
|| [!B0]   MV      .D2     B21,B20           ; |43| <0,8>
|| [ A0]   BDEC    .S1     $C$L2,A0          ; |39| <0,8>
|| [!B0]   SSUB    .L2X    A3,B20,B6         ; |42| <0,8>
|| [!B0]   SMPY    .M2     B4,B7,B26         ; |42| <0,8>  ^
|| [!B0]   SADD    .S2     B5,B6,B5          ; |42| <0,8>  ^
||         LDH     .D1T1   *A21++,A3         ; |42| <1,0>

   [!B0]   SADD    .S1     A23,A25,A23       ; |42| <0,9>  ^
|| [!B0]   MV      .D1     A9,A8             ; |43| <0,9>
|| [!B0]   SSUB    .L1     A3,A8,A25         ; |42| <0,9>
|| [!B0]   MV      .S2     B25,B24           ; |43| <0,9>
|| [!B0]   SSUB    .L2X    A3,B24,B5         ; |42| <0,9>
|| [!B0]   MV      .D2     B5,B23            ; |44| <0,9>  ^

   [!B0]   SSUB    .L1     A3,A16,A25        ; |42| <0,10>
|| [!B0]   SADD    .S1     A25,A27,A23       ; |42| <0,10>  ^
|| [!B0]   MV      .D1     A23,A5            ; |44| <0,10>  ^
|| [!B0]   SADD    .L2     B6,B26,B6         ; |42| <0,10>  ^
||         SHR     .S2     B23,15,B26        ; |42| <1,2>  ^

   [!B0]   SADD    .L1     A25,A26,A24       ; |42| <0,11>  ^
|| [!B0]   MV      .D1     A24,A7            ; |44| <0,11>  ^
|| [!B0]   MV      .D2     B19,B18           ; |43| <0,11>
|| [!B0]   SADD    .S2     B5,B16,B5         ; |42| <0,11>  ^
|| [!B0]   SSUB    .L2X    A3,B18,B16        ; |42| <0,11>
```

```
||         SHR     .S1     A5,15,A3          ; |42| <1,3>  ^
||         SMPY    .M2     B26,B8,B6         ; |42| <1,3>  ^

   [!B0]   MV      .L1     A17,A16           ; |43| <0,12> Define a twin register
|| [!B0]   SADD    .L2     B16,B4,B4         ; |42| <0,12>  ^
|| [!B0]   MV      .S2     B5,B25            ; |44| <0,12>  ^
|| [!B0]   MV      .D2     B6,B21            ; |44| <0,12>  ^
||         SMPY    .M1     A3,A18,A25        ; |42| <1,4>  ^
||         SHR     .S1     A7,15,A25         ; |42| <1,4>  ^

   [ B0]   SUB     .L2     B0,1,B0           ; <0,13>
|| [!B0]   MV      .L1     A24,A17           ; |44| <0,13>  ^
|| [!B0]   MV      .D1     A23,A9            ; |44| <0,13>  ^
|| [!B0]   MV      .D2     B4,B19            ; |44| <0,13>  ^
||         SMPY    .M1     A25,A19,A26       ; |42| <1,5>  ^
||         SHR     .S2     B25,15,B4         ; |42| <1,5>  ^
||         SSHL    .S1     A3,8,A3           ; |42| <1,5>

$C$DW$L$_c64_goertzel_power_opt_2$3$E:
;*--------------------------------------------------------------------------*
```

В данной программе ядро цикла загружено на 52/64=81,25 %, а в предыдущей программе, где параллельно вычислялось 6 мощностей, загруженность ядра составила 41/48=85,42%, что говорит о том, что ядро цикла загружено менее плотно, но и в полученной программе, и в предыдущей за 1 такт обрабатываются данные для вычисления 1 мощности, так как в данной программе за 8 тактов обрабатывается 8 элементов, а в предыдущей - за 6 тактов 6 элементов.

c64_goertzel_power_no_opt(short, short *, int, int *)	800	934.38	747504
c64_goertzel_power_opt_1(short *, short *, int, int *)	100	713.36	71336
c64_goertzel_power_opt_2(short *, short *, int, int *)	100	1003.04	100304

Рисунок 10: Результаты после составления программы для расчёта 8 мощностей параллельно

Для расчёта 8 мощностей, при размере входного массива равного 100 элементам, эталонной программе нужно в среднем 747504/100=7475,04 такта, а оптимизированной программе для это нужно в среднем 1003,04 такта, оптимизированная программа быстрее на 645,24%.

5. Заключение

Для того, чтобы не происходило переполнения при расчётах, амплитуда входного массива должна быть ограничена 14 битами. Максимальная длина массива, при котором мощность рассчитывается с погрешностью не более 1%, равна 128 элементам. Если длина массива больше 128 элементов, то произойдёт переполнение величины s, записанной в формате Q8.23, так как для вычисления s необходимо, чтобы результат умножения s[i-1] в формате Q8.23 на косинус в формате Q0.15, с последующим умножением на 2, который является слагаемым суммы, составляющей s, укладывался в тип int.

Эталонный алгоритм был скомпилирован с обычным ключом оптимизации -O2. Его производительность взята за 100%. Дальнейшие оптимизации относятся к ускорению эталонного алгоритма за счет структурной реорганизации кода, накладывания дополнительных ограничений на входные данные и применения специализированных

инструкций и параллельному расчёту нескольких мощностей .

Основные факторы, ограничивающие скорость выполнения алгоритма:

- завязка по данным, не позволяющая обрабатывать несколько элементов массива параллельно, из-за чего компилятор не может загрузить все вычислительные ресурсы процессора.

Оптимизация алгоритма свелась к расчёту нескольких мощностей входного сигнала для реализации таких задач, как получение информации с помощью DTMF сигнала. В итоге удалось равномерно загрузить все основные блоки процессора: .D, .S и .L и добиться обработки одного входного элемента для каждой из восьми мощностей сигнала в каждом восьмом такте для ядра C64 и одного входного элемента для каждой из шести мощностей сигнала в каждом шестом такте для ядра C62.

Сравнительный анализ шагов оптимизации для ядер C62 и C64 представлен в таблице ниже.

Таблица 1: Сравнение производительности программ, реализующих алгоритм Герцля, для C62 и C64

Имя функции	Тактов (n=100)	Тактов (n=50)	Прирост на n=50	Тактов на элемент	Произв-сть (n=100)	Произв-сть на элемент	Кол-во част-тот	Тактов на элем. для одной частоты	Произв-сть на элемент для одной частоты
c62_goertzel_power_no_opt	**1137,17**	**587,17**	**550**	**11**	**100,00%**	**100,00%**	**1**	**11**	**100,00%**
c62_goertzel_power_opt_1	1109,87	584,87	525	10,5	102,46%	104,76%	1	10,5	104,76%
c62_goertzel_power_opt_2	515,22	290,22	225	4,5	220,72%	244,44%	1	4,5	244,44%
c62_goertzel_power_opt_3	789,94	489,94	300	6	143,96%	183,33%	6	1	1100,00%
c64_goertzel_power_no_opt	**934,38**	**484,38**	**450**	**9**	**100,00%**	**100,00%**	**1**	**9**	**100,00%**
c64_goertzel_power_opt_1	713,36	413,36	300	6	130,98%	150,00%	6	1	900,00%
c64_goertzel_power_opt_2	1003,04	603,04	400	8	93,43%	112,50%	8	1	900,00%

Как видно из таблицы, последний вариант функции `c62_goertzel_power_opt_3` дает прирост производительности, около 1,83 раз при n=100, и удельный прирост в 11 раз по сравнению с эталонным алгоритмом для вычисления мощности сигнала на 1 частоте для ядра C62. Вариант функции `c64_goertzel_power_opt_1` даёт прирост производительности, около 1,31 раза при n=100, и удельный прирост в 9 раз по сравнению с эталонным алгоритмом для вычисления мощности сигнала на 1 частоте для ядра C64. При расчёте восьми мощностей сигнала на разных частотах получился такой же удельный прирост производительности в 9 раз, но за счёт увеличенного оверхэда программа работает медленнее программы для расчёта 6 частот параллельно. Программу `c64_goertzel_power_opt_2` выгодно применять, когда количество частот, необходимых для расчёта кратно 8 и не кратно 6.

5.1. Общие вопросы по заданию

5.1.1. Во сколько раз удалось ускорить алгоритм по сравнению с эталоном?

Как видно из таблицы 1, удалось добиться ускорения в 1,43 раз по сравнению с эталоном для ядра C62 и в 1,38 раз на ядре C64, что соответствует удельному росту производительности в 11 раз на один элемент входных данных (длину вектора) для ядра C62 и 9 раз для ядра C64, при расчёте 6 мощностей сигнала параллельно.

5.1.2. Зависит ли ускорение от архитектуры ядра?

Нет, в алгоритме Герцля для ядер C62 и C64 удалось одинаковой удельной производительности при расчёте нескольких мощностей сигнала параллельно. Оба ядра вышли на пиковую мощность, определяемую вычислительными ресурсами. За счёт увеличенного количества регистров на ядре C64 удалось организовать расчёт 8 мощностей параллельно.

5.1.3. Какие дополнительные ограничения целесообразно наложить на входные данные для повышения производительности?

Для приведенного алгоритма необходимо наложить следующие ограничения.

1. Адрес массива должен быть кратен 2, так как команда LDH читает из памяти 16-битные числа, их адрес должен быть кратен 2. Для максимальной скорости требуется массив с числом элементов, кратным 4. Это следует из анализа оптимизированного кода.

5.2. Дополнительные вопросы по заданию

5.2.1. Достаточно ли расчета одного алгоритма Герцля, чтобы загрузить все ресурсы процессора?

Из результатов оптимизации следует, что из-за завязки по данным, возникающей из-за зависимости последующих расчётов от предыдущих, не удаётся загрузить ресурсы процессора полностью, в некоторых тактах ни один исполнительный юнит процессора вообще не выполняет полезной работы. Для устранения этого эффекта применено параллельное вычисление нескольких мощностей входного сигнала для разных частот.

5.2.2. Какие ресурсы процессора определяют быстродействие в данном случае?

Для расчёта алгоритма Герцля необходимо получать результаты предыдущих расчётов, чтобы переходить к следующим. Так как получение результата из предыдущей итерации составляет минимум 5 тактов, ограничение заключается в завязке по данным, из-за которой не удаётся обеспечить все юниты процессора полезной работой в каждом такте. Петля завязки по данным ограничивает быстродействие в данном случае.

5.2.3. Сколько потоков алгоритма можно выполнять параллельно так, чтобы это было быстрее, чем последовательно один за другим?

Для ядра C62 удалось выполнить максимум 6 потоков алгоритма параллельно. Для ядра C64 удалось выполнить 8 потоков параллельно, что может пригодиться для ускорения задачи детектора DTMF сигнала. Но расчёт 8 потоков не даёт прироста скорости, по сравнению с расчётом 6 потоков.

5.2.4. Как это зависит от типа процессора (62x, 64x, 64x+)?

Это зависит от набора инструкций выбранного процессора, наличия нескольких вычислительных ядер процессора, которые присутствуют в ядре 64x+, а также от задержки, необходимой для выборки данных из памяти.

Список использованной литературы

1. *TMS320C6000 Optimizing C Compiler Tutorial* // SPRU425
2. *Hand-Tuning Loops and Control Code on the TMS320C6000* // SPRA666
3. *TMS320C6000 programmer's guide* // SPRU198k
4. Goertzel algorithm // https://en.wikipedia.org/wiki/Goertzel_algorithm

Приложение. Исходные тексты программ

В данном разделе представлены листинги всех исходных модулей разработанной программы, а также вывод программы при запуске.

Листинг 21: `goertzel_type.h`

```
#include <stdio.h>

typedef short s16;
typedef int s32;
```

Листинг 22: main.c

```
#include "goertzel_type.h"
#define N 128
short A[N];

void c62_goertzel_power_no_opt(s16 cos_Q15, const s16 *in, int n, s32 *power);
void c64_goertzel_power_no_opt(s16 cos_Q15, const s16 *in, int n, s32 *power);
void c62_goertzel_power_opt_1(s16 cos_Q15, const s16 *in, int n, s32 *power);
void c62_goertzel_power_opt_2(s16 cos_Q15, const s16 *in, int n, s32 *power);
void c62_goertzel_power_opt_3(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power);
void c64_goertzel_power_opt_1(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power);
void c64_goertzel_power_opt_2(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power);

void print_goertzel(long check, s32 power, char *prefix) {
   float checkf = (float) check / 8388608; // 8388608 = 2^23
   float powerf = (float) power / 16384; // 16384 = 2^14
   float delta = checkf - powerf;
   float epsilon = 200 * delta / (checkf + powerf);
   float gp = powerf / checkf * 100;
   printf("goertzel_power%s: %.4f(%.3f%%), ", prefix, powerf, gp);
   printf("difference: %.3f%%", epsilon);
   if (epsilon <= 1)
      printf("(OK!).\n");
   else
      printf("(ERROR!).\n");
}

void print_multi_goertzel(long check, int *power, int *origin,
      char *prefix, char *prefix1, int n, s16 *cos) {
   int i;
   float checkf = (float) check / 8388608;
   float powerf = (float) power[0] / 16384;
   float delta = checkf - powerf;
   float epsilon = 200 * delta / (checkf + powerf);
   printf("-------------------------------------------------\n");
   for (i = 0; i < n; i++) {
      float powerfi = (float) power[i] / 16384;
      float originfi = (float) origin[i] / 16384;
```

```
      float gp = powerfi / originfi * 100;
      float cosf = (float) cos[i] / 32768;
      float deltai = originfi - powerfi;
      float epsiloni = 200 * deltai / (powerfi + originfi);
      printf("cos[%d] = %f:\n", i, cosf);
      printf("goertzel_power%s: %.4f(%.1f%%);\n", prefix, powerfi, gp);
      printf("goertzel_power_no_opt%s: %.4f(100%%);\n",prefix1, originfi);
      printf("difference: %.2f%%.\n\n", epsiloni);
   }
   float coscheck = (float) cos[0] / 32768;
   float gpc = powerf / checkf * 100;
   printf("CHECK:\n");
   printf("cos[0] = %f:\n", coscheck);
   printf("total_power: %.4f(100%%);\n", checkf);
   printf("goertzel_power%s: %.4f(%.3f%%);\n", prefix, powerf, gpc);
   printf("difference: %.3f%%", epsilon);
   if (epsilon <= 1)
      printf("(OK!).\n\n");
   else
      printf("(ERROR!).\n\n");
}

long total_power(short *in, int n) {
   int i;
   long total = 0;
   for (i = 0; i < n; i++) {
      short x = in[i];
      total += x * x;
      // Q0.15 * Q0.15 => Q0.30
   }
   return (total * n / 2) >> 7;
   // Q0.30 >> 7 => 8.23
}

int main(void) {
   int i, numberg4, numberg5, k, g162[6], g164[8], g2, g3, g4[6], g5[6], g6[8];
   long check;
   float checkf;
   numberg4 = sizeof(g4) / sizeof(g4[0]);
   numberg5 = sizeof(g6) / sizeof(g6[0]);
   short cosx[] = { 16000, 14000, 12000, 10000, 8000, 6000, 4000, 2000 };
   float cosf = (float) cosx[0] / 32768; // 32768 = 2^15
   A[0] = 0;
   A[1] = 5000;
   for (i = 2; i < N; i++) {
      A[i] = 2 * ((A[i - 1] * cosx[0]) >> 15) - A[i - 2];
   }
   check = total_power(A, N);
   for (i = 0; i < 100; i++) {
      for (k = 0; k < 6; k++) {
         c62_goertzel_power_no_opt(cosx[k], A, N, &g162[k]);
      }
      for (k = 0; k < 8; k++) {
         c64_goertzel_power_no_opt(cosx[k], A, N, &g164[k]);
      }
      c62_goertzel_power_opt_1(cosx[0], A, N, &g2);
      c62_goertzel_power_opt_2(cosx[0], A, N, &g3);
      c62_goertzel_power_opt_3(cosx, A, N, g4);
```

```
      c64_goertzel_power_opt_1(cosx, A, N, g5);
      c64_goertzel_power_opt_2(cosx, A, N, g6);

   }
   int amp = -32768;
   for (i = 1; i < N; i++) {
      if (amp < A[i])
         amp = A[i];
   }
   float ampf = (float) amp / 32768;
   checkf = (float) check / 8388608;
   printf("amplitude: %.5f, quantity of elements: %d:\n\n", ampf, N);
   printf("cos=%.3f:\n", cosf);
   printf("total_power: %.4f (100%%)\n", checkf);
   print_goertzel(check, g162[0], "_no_opt_c62");
   print_goertzel(check, g2, "_opt_1_c62");
   print_goertzel(check, g3, "_opt_2_c62");
   print_multi_goertzel(check, g4, g162, "_opt_3_c62", "_c62", numberg4, cosx);
   printf("-------------------------------------------------\n");
   printf("cos=%.3f:\n", cosf);
   printf("total_power: %.4f (100%%)\n", checkf);
   print_goertzel(check, g164[0], "_no_opt_c64");
   print_multi_goertzel(check, g5, g164, "_opt_1_c64", "_c64", numberg4, cosx);
   print_multi_goertzel(check, g6, g164, "_opt_2_c64", "_c64",  numberg5, cosx);
   return 0;
}
```

Листинг 23: goertzel_no_opt_c62.c

```
// Эталонная программа алгоритма Герцля (ядро c62)
#include "goertzel_type.h"

void c62_goertzel_power_no_opt(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int s;
   int i;
   for (i = 0; i < n; i++) {
      int x = (in[i]) << 8;
      // Q0.15 << 8 => Q0.23
      s = 2 * (((sprev) >> 15) * cos_Q15) - sprev1 + x;
      // Q8.23 >> 15 => Q8.8
      // Q8.8 * Q0.15 = Q8.23
      sprev1 = sprev;
      sprev = s;
   }
   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   // Q8.23 >> 16 => Q8.7
   *power = ((sprev1 * sprev1)) + ((sprev * sprev))
         - (((sprev * cos_Q15) >> 14) * sprev1);
   // Q8.7 * Q8.7 => Q16.14
   // Q8.7 * Q0.15 >> 15 *2 = Q8.7 * Q0.15 >> 14 => Q8.7 * 2
}
```

Листинг 24: goertzel_no_opt_c64.c

```c
// Эталонная программа алгоритма Герцля (ядро c64)
#include "goertzel_type.h"

void c64_goertzel_power_no_opt(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int s;
   int i;
   for (i = 0; i < n; i++) {
      int x = (in[i]) << 8;
      // Q0.15 << 8 => Q0.23
      s = 2 * (((sprev) >> 15) * cos_Q15) - sprev1 + x;
      // Q8.23 >> 15 => Q8.8
      // Q8.8 * Q0.15 = Q8.23
      sprev1 = sprev;
      sprev = s;
   }
   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   // Q8.23 >> 16 => Q8.7
   *power = ((sprev1 * sprev1)) + ((sprev * sprev))
         - (((sprev * cos_Q15) >> 14) * sprev1);
   // Q8.7 * Q8.7 => Q16.14
   // Q8.7 * Q0.15 >> 15 *2 = Q8.7 * Q0.15 >> 14 => Q8.7 * 2
}
```

Листинг 25: goertzel_opt_c62.c

```c
// Оптимизированная программа алгоритма Герцля (ядро c62)
#include "goertzel_type.h"

void c62_goertzel_power_opt_1(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int s;
   int i;
#pragma UNROLL(2)
   for (i = 0; i < n; i++) {
      int x = (in[i]) << 8;
      s = 2 * (((sprev) >> 15) * cos_Q15) - sprev1 + x;
      sprev1 = sprev;
      sprev = s;
   }

   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   *power = (sprev1 * sprev1 + sprev * sprev
         - ((sprev * cos_Q15) >> 14) * sprev1);
}

void c62_goertzel_power_opt_2(s16 cos_Q15, const s16 *in, int n, s32 *power) {
   int sprev = 0;
   int sprev1 = 0;
   int i, s;
#pragma UNROLL(2)
```

```
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      s = _sadd(_ssub(x, sprev1), _smpy((sprev) >> 15, cos_Q15));
      sprev1 = sprev;
      sprev = s;
   }
   sprev = (sprev) >> 16;
   sprev1 = (sprev1) >> 16;
   *power = (sprev1 * sprev1 + sprev * sprev
         - (_smpy(sprev, cos_Q15) >> 15) * sprev1);
}

void c62_goertzel_power_opt_3(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power) {
   int s[6], i, k, sprev[6], sprev1[6];
   int cosx[6];
   for (k = 0; k < 6; k++) {
      cosx[k] = cos_Q15[k];
      sprev[k] = 0;
      sprev1[k] = 0;
   }
#pragma MUST_ITERATE(2, ,2)
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      for (k = 0; k < 6; k++) {
         s[k] = _sadd(_ssub(x, sprev1[k]), _smpy((sprev[k]) >> 15, cosx[k]));
         sprev1[k] = sprev[k];
         sprev[k] = s[k];
      }
   }
   for (k = 0; k < 6; k++) {
      sprev[k] = (sprev[k]) >> 16;
      sprev1[k] = (sprev1[k]) >> 16;
      power[k] = (_mpy(sprev1[k], sprev1[k])
            + _mpy(sprev[k], sprev[k])
            - _mpy((_smpy(sprev[k], cosx[k]) >> 15), sprev1[k]));
   }
}
```

Листинг 26: goertzel_opt_c64.c

```
// Оптимизированная программа алгоритма Герцля (Ядро c64)
#include "goertzel_type.h"

void c64_goertzel_power_opt_1(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power) {
   int s[6], i, k, sprev[6], sprev1[6];
   short cosx[6];
   for (k = 0; k < 6; k++) {
      cosx[k] = cos_Q15[k];
      sprev[k] = 0;
      sprev1[k] = 0;
   }
#pragma MUST_ITERATE(2, ,2)
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
```

```
      for (k = 0; k < 6; k++) {
         s[k] = _sadd(_ssub(x, sprev1[k]), _smpy((sprev[k]) >> 15, cosx[k]));
         sprev1[k] = sprev[k];
         sprev[k] = s[k];
      }
   }
   for (k = 0; k < 6; k++) {
      sprev[k] = (sprev[k]) >> 16;
      sprev1[k] = (sprev1[k]) >> 16;
      power[k] = (_mpy(sprev1[k], sprev1[k])
            + _mpy(sprev[k], sprev[k])
            - _mpy((_smpy(sprev[k], cosx[k]) >> 15), sprev1[k]));
   }
}

void c64_goertzel_power_opt_2(const s16 *cos_Q15, const s16 *in, int n,
      s32 *power) {
   int s[8], i, k, sprev[8], sprev1[8];
   short cosx[8];
   for (k = 0; k < 8; k++) {
      cosx[k] = cos_Q15[k];
      sprev[k] = 0;
      sprev1[k] = 0;
   }
#pragma MUST_ITERATE(2, ,2)
   for (i = 0; i < n; i++) {
      int x = _sshl(in[i], 8);
      for (k = 0; k < 8; k++) {
         s[k] = _sadd(_ssub(x, sprev1[k]), _smpy((sprev[k]) >> 15, cosx[k]));
         sprev1[k] = sprev[k];
         sprev[k] = s[k];
      }
   }
   for (k = 0; k < 8; k++) {
      sprev[k] = (sprev[k]) >> 16;
      sprev1[k] = (sprev1[k]) >> 16;
      power[k] = (_mpy(sprev1[k], sprev1[k])
            + _mpy(sprev[k], sprev[k])
            - _mpy((_smpy(sprev[k], cosx[k]) >> 15), sprev1[k]));
   }
}
```

Вывод программы при запуске:

```
amplitude: 0.17480, quantity of elements: 100:

cos=0.488:
total_power: 76.3368 (100%)
goertzel_power_no_opt_c62: 76.1508(99.756%), difference: 0.244%(OK!).
goertzel_power_opt_1_c62: 76.1508(99.756%), difference: 0.244%(OK!).
goertzel_power_opt_2_c62: 76.1508(99.756%), difference: 0.244%(OK!).
-------------------------------------------------
cos[0] = 0.488281:
goertzel_power_opt_3_c62: 76.1508(100.0%);
goertzel_power_no_opt_c62: 76.1508(100%);
difference: 0.00%.
```

```
cos[1] = 0.427246:
goertzel_power_opt_3_c62: 0.5297(100.0%);
goertzel_power_no_opt_c62: 0.5297(100%);
difference: 0.00%.

cos[2] = 0.366211:
goertzel_power_opt_3_c62: 0.3529(100.0%);
goertzel_power_no_opt_c62: 0.3529(100%);
difference: 0.00%.

cos[3] = 0.305176:
goertzel_power_opt_3_c62: 0.2335(100.0%);
goertzel_power_no_opt_c62: 0.2335(100%);
difference: 0.00%.

cos[4] = 0.244141:
goertzel_power_opt_3_c62: 0.1494(100.0%);
goertzel_power_no_opt_c62: 0.1494(100%);
difference: 0.00%.

cos[5] = 0.183105:
goertzel_power_opt_3_c62: 0.0898(100.0%);
goertzel_power_no_opt_c62: 0.0898(100%);
difference: 0.00%.

CHECK:
cos[0] = 0.488281:
total_power: 76.3368(100%);
goertzel_power_opt_3_c62: 76.1508(99.756%);
difference: 0.244%(OK!).

-------------------------------------------------
cos=0.488:
total_power: 76.3368 (100%)
goertzel_power_no_opt_c64: 76.1508(99.756%), difference: 0.244%(OK!).
-------------------------------------------------
cos[0] = 0.488281:
goertzel_power_opt_1_c64: 76.1508(100.0%);
goertzel_power_no_opt_c64: 76.1508(100%);
difference: 0.00%.

cos[1] = 0.427246:
goertzel_power_opt_1_c64: 0.5297(100.0%);
goertzel_power_no_opt_c64: 0.5297(100%);
difference: 0.00%.

cos[2] = 0.366211:
goertzel_power_opt_1_c64: 0.3529(100.0%);
goertzel_power_no_opt_c64: 0.3529(100%);
difference: 0.00%.

cos[3] = 0.305176:
goertzel_power_opt_1_c64: 0.2335(100.0%);
goertzel_power_no_opt_c64: 0.2335(100%);
difference: 0.00%.

cos[4] = 0.244141:
```

```
goertzel_power_opt_1_c64: 0.1494(100.0%);
goertzel_power_no_opt_c64: 0.1494(100%);
difference: 0.00%.

cos[5] = 0.183105:
goertzel_power_opt_1_c64: 0.0898(100.0%);
goertzel_power_no_opt_c64: 0.0898(100%);
difference: 0.00%.

CHECK:
cos[0] = 0.488281:
total_power: 76.3368(100%);
goertzel_power_opt_1_c64: 76.1508(99.756%);
difference: 0.244%(OK!).

------------------------------------------------
cos[0] = 0.488281:
goertzel_power_opt_2_c64: 76.1508(100.0%);
goertzel_power_no_opt_c64: 76.1508(100%);
difference: 0.00%.

cos[1] = 0.427246:
goertzel_power_opt_2_c64: 0.5297(100.0%);
goertzel_power_no_opt_c64: 0.5297(100%);
difference: 0.00%.

cos[2] = 0.366211:
goertzel_power_opt_2_c64: 0.3529(100.0%);
goertzel_power_no_opt_c64: 0.3529(100%);
difference: 0.00%.

cos[3] = 0.305176:
goertzel_power_opt_2_c64: 0.2335(100.0%);
goertzel_power_no_opt_c64: 0.2335(100%);
difference: 0.00%.

cos[4] = 0.244141:
goertzel_power_opt_2_c64: 0.1494(100.0%);
goertzel_power_no_opt_c64: 0.1494(100%);
difference: 0.00%.

cos[5] = 0.183105:
goertzel_power_opt_2_c64: 0.0898(100.0%);
goertzel_power_no_opt_c64: 0.0898(100%);
difference: 0.00%.

cos[6] = 0.122070:
goertzel_power_opt_2_c64: 0.0542(100.0%);
goertzel_power_no_opt_c64: 0.0542(100%);
difference: 0.00%.

cos[7] = 0.061035:
goertzel_power_opt_2_c64: 0.0298(100.0%);
goertzel_power_no_opt_c64: 0.0298(100%);
difference: 0.00%.

CHECK:
cos[0] = 0.488281:
```

```
total_power: 76.3368(100%);
goertzel_power_opt_2_c64: 76.1508(99.756%);
difference: 0.244%(OK!).
```

Примечание: значения, полученные при вычислении мощности сигнала с помощью алгоритма Герцля и вычисления суммы квадратов значений сигнала, различаются на 0,2%, так как суммы считаются в конечном интервале времени, в течение которого частота сигнала не может быть точно установлена.

КОНЕЦ ДОКУМЕНТА