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: ассистент _____ М. Ю. Уздяев

Оптимальные системы

Практическое задание №3

Шкалы дальности РЛС, Вариант 2

выполнил студент группы 3721

11.01.2021, А. А. Булыгин

Санкт-Петербург, 2020

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Булыгин		
Проверил		Уздяев		
Н. контр.				
Утв.		Савельев		

ГУАП.3721.0С-003

Оптимальные системы
Практическое задание №3

Лит.	Лист	Листов
Р	1	77

ГУАП

Копировал

Формат А4

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 3. Шкалы дальности РЛС, вариант 2. Имеется байтовый массив in с данными оцифрованного видеосигнала (n байт) с приемника РЛС. Для дальнейшей передачи по сети желательно уплотнить передаваемую информацию путем объединения каждой группы из соседних m байтов в один, равный максимальному значению в этой группе. Необходимо разработать алгоритм, выполняющий упаковку.

```
int pack_scale(const u8 *in, int n, int m, u8* result);
```

Возвращаемое значение равно числу байтов выходного массива result, в котором складываются упакованные данные (предполагается, что n кратно m).

Примечание. При m=1 алгоритм, очевидно, вырождается в простое копирование памяти.

Дополнительные вопросы:

1) Зависит ли оптимальный алгоритм от параметра m? Для каких значений m можно обойтись общей реализацией упаковки? С чем это может быть связано?
2) Как вы можете объяснить разницу в скорости для четных и нечетных m?

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 _xm_opt_N, где N обозначает номер шага оптимизации, xm – кратность количества соседних объединяющихся байт. Если функция предназначена для определённого m, то x не прописывается, а пишется только m. Имя имеет также приставку cxx_, где cxx — серия ядра процессора, например, c62_pack_scale_x2_opt_2 для шага оптимизации 2, ядра c62, m в функции кратно 2.

2. Базовый алгоритм выполнения задания

2.1. Эталонный алгоритм

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

Листинг 1: Заголовочный файл

```
#include <stdio.h>
#include <string.h>

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;
```

Листинг 2: Эталонный алгоритм уплотнения сигнала РЛС

```
// Эталонная программа для уплотнения РЛС сигнала (ядро C62)
#include "pack_type.h"

int c62_pack_scale_no_opt(const u8 *in, int n, int m, u8* result)
{
   int k;
   u8 max;
   int count = 0;
   while (n >= m) {
      max = 0;
      for (k = 0; k < m; k++) {
         u8 x = *in++;
         max = (x > max) ? x : max;
      }
      *result++ = max;
      n -= m;
      count++;
   }
   return count;
}
```

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="c62_pack_scale_no_opt.pp"  "../c62_pack_scale_no_opt.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. В данном задании основным считается внутренний цикл, так как компилятор создаёт расписание для организации конвейера для внутреннего цикла.

Листинг 3: Фидбэк эталонного алгоритма, собранного для C62x

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c62_pack_scale_no_opt.c
;*      Loop source line                 : 10
;*      Loop opening brace source line   : 10
;*      Loop closing brace source line   : 13
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound     : 1
;*      Partitioned Resource Bound(*)    : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     1*       0
;*      .S units                     0        1*
;*      .D units                     1*       0
;*      .M units                     0        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)          1        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1*       1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 3
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      For further improvement on this loop, try option -mh3
;*
;*      Minimum safe trip count       : 3
;*----------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP PROLOG
```

```
           B       .S2     $C$L5             ; |10| (P) <0,1>
           LDBU    .D1T1   *A4++,A3          ; |11| (P) <1,0>
           B       .S2     $C$L5             ; |10| (P) <1,1>

           SUB     .L1X    B0,3,A2
||         SUB     .L2     B0,3,B0
||         LDBU    .D1T1   *A4++,A3          ; |11| (P) <2,0>

;** --------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_no_opt$8$B:

           CMPGTU  .L1     A5,A3,A1          ; |11| <0,5>  ^
|| [ B0]   B       .S2     $C$L5             ; |10| <2,1>

   [ A2]   SUB     .L1     A2,1,A2           ; <0,6>
|| [!A1]   MV      .S1     A3,A5             ; |11| <0,6>  ^
|| [ B0]   SUB     .L2     B0,1,B0           ; |10| <3,0>
|| [ A2]   LDBU    .D1T1   *A4++,A3          ; |11| <3,0>

$C$DW$L$_c62_pack_scale_no_opt$8$E:
;** --------------------------------------------------------------------------*
$C$L6:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Предварительный анализ функции с эталонным алгоритмом показывает, что:

- Цикл не имеет эпилога, но пролог занимает 4 такта, что может негативно сказаться на скорости выполнения программы, так как данный цикл будет выполняться несколько раз в теле внешнего цикла. Из этого следует, что в оптимизированной программе необходимо добиться минимального времени выполнения оверхэда внутреннего цикла.
- Ядро цикла умещается в 2 пакета выборки инструкций (fetch packet). Использовано за 2 такта 37,5% вычислительных ресурсов ядра, это говорит об изначальной невысокой производительности алгоритма, из-за примитивности поиска максимума;
- За 1 итерацию ядра цикла обрабатывается 1 элемент массива;
- Сторона A более загружена, чем сторона B, это говорит о том, что в оптимизированной программе нужно произвести балансировку ресурсов по сторонам процессора.

3. Создание оптимального алгоритма (C62x)

3.1. Разворачивание цикла

Попробуем уменьшить оверхэд внутреннего цикла с помощью избавления от условия при использовании прагмы MUST_ITERATE и с помощью опции -mh, позволяющие производить спекулятивные чтения из памяти.

Листинг 4: Уменьшение оверхэда цикла

```
#include "pack_type.h"

int c62_pack_scale_x1_opt_1(const u8 *in, int n, int m, u8* result)
{
   int k;
   u8 max;
   int count = 0;
   while (n >= m) {
      max = 0;
#pragma MUST_ITERATE(1)
      for (k = 0; k < m; k++) {
         u8 x = *in++;
         max = (x > max) ? x : max;
      }
      *result++ = max;
      n -= m;
      count++;
   }
   return count;
}
```

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=3 -k
--preproc_with_compile --preproc_dependency="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

Позволяет компилятору составить следующее расписание:

Листинг 5: Уменьшение оверхэда цикла

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c62_pack_scale_opt.c
;*      Loop source line                 : 11
;*      Loop opening brace source line   : 11
;*      Loop closing brace source line   : 14
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound     : 1
;*      Partitioned Resource Bound(*)    : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     1*       0
;*      .S units                     0        1*
;*      .D units                     1*       0
;*      .M units                     0        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)          1        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1*       1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 3
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 3 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L18:    ; PIPED LOOP PROLOG
           B       .S2     $C$L19           ; |11| (P) <0,1>

           SUB     .L2     B7,2,B0
||         LDBU    .D1T1   *A4++,A3         ; |12| (P) <1,0>

   [ B0]   B       .S2     $C$L19           ; |11| (P) <1,1>

           ZERO    .L1     A5               ; |9|
|| [ B0]   SUB     .L2     B0,1,B0          ; |11| (P) <2,0>
||         LDBU    .D1T1   *A4++,A3         ; |12| (P) <2,0>

;** ----------------------------------------------------------------------------*
$C$L19:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_opt_1$4$B:

           CMPGTU  .L1     A5,A3,A1         ; |12| <0,5>  ^
|| [ B0]   B       .S2     $C$L19           ; |11| <2,1>

   [!A1]   MV      .L1     A3,A5            ; |12| <0,6>  ^
|| [ B0]   SUB     .L2     B0,1,B0          ; |11| <3,0>
||         LDBU    .D1T1   *A4++,A3         ; |12| <3,0>

$C$DW$L$_c62_pack_scale_opt_1$4$E:
;** ----------------------------------------------------------------------------*
$C$L20:    ; PIPED LOOP EPILOG
;** ----------------------------------------------------------------------------*
```

Прирост производительности составляет 126,3% для массива из 128 элементов и m=16:

Name	Calls	Incl Count Average
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	538.39
c62_pack_scale_x1_opt_1(unsigned char *, int, int, unsigned char *)	100	426.14

Рисунок 1: Сравнение эталона с программой, имеющие меньший оверхэд во внутреннем цикле(m = 16, n = 128)

Прирост производительности связан с отсутствием проверки на n=0 в начале функции из-за прагмы MUST_ITERATE(1), а также за счёт спекулятивных чтений данных. Можно значительно уменьшить время выполнения программы для m от 0 до 32 сведя двойной

цикл к одинарному, из-за чего оверхэд будет выполнятся всего 1 раз.

Листинг 6: Программа с одинарным циклом

```
#include "pack_type.h"

int c62_pack_scale_x1_opt_1(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   int count = 0;
   int countb = 0;
   u8 max = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k++) {
      u8 x = *in++;
      max = (x > max) ? x : max;
      count++;
      if (count == m) {
         *result++ = max;
         max = 0;
         count = 0;
         countb++;
      }
   }
   return countb;
}
```

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="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

Конвейер стал выглядеть следующим образом:

Листинг 7: Добавление прагмы разворачивания цикла

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c62_pack_scale_opt.c
;*      Loop source line                 : 10
;*      Loop opening brace source line   : 10
;*      Loop closing brace source line   : 20
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 3
;*      Unpartitioned Resource Bound     : 2
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     1        1
;*      .S units                     1        1
;*      .D units                     1        1
```

```
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             2        0
;*      Long read paths              1        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          3        2     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        1
;*      Bound(.L .S .D .LS .LSD)     2        2
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 3  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 2
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 2 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L13:    ; PIPED LOOP PROLOG

   [ B1]   B       .S2     $C$L14           ; |10| (P) <0,3>
||         LDBU    .D1T1   *A5++,A4         ; |11| (P) <1,0>

           NOP             1

           ZERO    .S1     A7               ; |8|
||         MV      .L1X    B4,A3
|| [ B1]   SUB     .L2     B1,1,B1          ; |10| (P) <1,2>

;** ---------------------------------------------------------------------------*
$C$L14:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_x1_opt_1$3$B:

           CMPEQ   .L2     B4,B6,B0         ; |14| <0,6>  ^
||         CMPGTU  .L1     A7,A4,A1         ; |11| <0,6>  ^
|| [ B1]   B       .S2     $C$L14           ; |10| <1,3>
||         LDBU    .D1T1   *A5++,A4         ; |11| <2,0>

   [ B0]   ADD     .L2     1,B5,B5          ; |18| <0,7>
|| [!A1]   MV      .L1     A4,A7            ; |11| <0,7>  ^
|| [ B0]   MVK     .S1     0x1,A3           ; |17| <0,7>  ^
|| [!B0]   ADD     .D1     1,A3,A3          ; |13| <0,7>  ^ Define a twin register

           MV      .L2X    A3,B4            ; |17| <0,8>  ^ Define a twin register
|| [ B0]   ZERO    .L1     A7               ; |16| <0,8>  ^
|| [ B0]   STB     .D1T1   A7,*A6++         ; |15| <0,8>  ^
|| [ B1]   SUB     .S2     B1,1,B1          ; |10| <2,2>

$C$DW$L$_c62_pack_scale_x1_opt_1$3$E:
;** ---------------------------------------------------------------------------*
$C$L15:    ; PIPED LOOP EPILOG
;** ---------------------------------------------------------------------------*
```

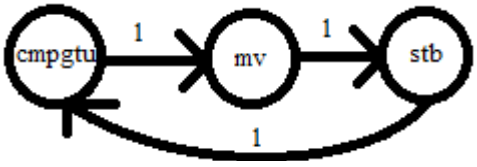

Рисунок 2: Граф петли зависимости по данным

Как видим, эпилог в цикле отсутствует, что уменьшает оверхэд функции. Итерационный интервал составляет 3 такта, за которые обрабатывается 1 элемент массива. Это связано с петлёй завязки по данным, присутствующей в цикле. Граф завязки по данным изображен на рисунке 2. При профилировании функции для массива из 128 элементов и m=16, получены результаты, изображенные на рисунке 3. Из данного рисунка следует, что прирост производительности составил 101,2 %. Такой малый прирост связан с тем, что в эталоне для обработки одного байта требуется 2 такта, а в оптимизированном варианте — 3, но из-за меньшего количества тактов, потраченного на оверхэд, удалось добиться меньшего времени выполнения оптимизированной программы. При уменьшении m прирост производительности полученной функции по сравнению с эталоном будет расти.

Name	Calls	Incl Count Average
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	538.39
c62_pack_scale_x1_opt_1(unsigned char *, int, int, unsigned char *)	100	532.20

Рисунок 3: Сравнение эталона с программой со сведением цикла к одинарному (n = 128, m = 16)

3.2. Ручное разворачивание цикла в 2 раза

Добиться максимальной скорости для m, кратных 2, удаётся путём разворачивания цикла вручную. При этом n также должно быть кратно m.

Листинг 8: Ручное разворачивание цикла в 2 раза

```
#include "pack_type.h"

int c62_pack_scale_x2_opt_2(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   int count = 0;
   int countb = 0;
   u8 max = 0;
   u8 max1 = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 2) {
      u8 x = *in++;
      u8 y = *in++;
      max = (x > max) ? x : max;
      max1 = (y > max1) ? y : max1;
      count += 2;
      if (count == m) {
         // конструкция для уменьшения завязки по данным
         if (max > max1)
```

```
            *result++ = max;
         if (max1 > max)
            *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb++;
      }
   }
   return countb;
}
```

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="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

Фидбэк стал выглядеть следующим образом:

Листинг 9: Фидбэк от компилятора для программы с ручным разворачиванием цикла

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c62_pack_scale_opt.c
;*      Loop source line                 : 31
;*      Loop opening brace source line   : 31
;*      Loop closing brace source line   : 47
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 5
;*      Unpartitioned Resource Bound     : 4
;*      Partitioned Resource Bound(*)    : 5
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3        2
;*      .S units                     0        2
;*      .D units                     2        2
;*      .M units                     0        0
;*      .X cross paths               2        2
;*      .T address paths             2        2
;*      Long read paths              1        1
;*      Long write paths             0        0
;*      Logical  ops (.LS)           2        1     (.L or .S unit)
;*      Addition ops (.LSD)          7        6     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             3        3
;*      Bound(.L .S .D .LS .LSD)     5*       5*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 5  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 safe trip count       : 2
;*--------------------------------------------------------------------------*
$C$L7:    ; PIPED LOOP PROLOG
           LDBU    .D1T1   *++A7(2),A3       ; |34| (P) <1,0>
           SUB     .L2     B9,2,B1

           ZERO    .S1     A5                ; |29|
||         CMPGTU  .L1     A4,A3,A1          ; |34| (P) <0,7>
|| [ B1]   B       .S2     $C$L8             ; |31| (P) <0,7>

;** --------------------------------------------------------------------------*
$C$L8:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_x2_opt_2$3$B:

           CMPEQ   .L2     B5,B4,B0          ; |37| <0,8>
|| [!A1]   MV      .S1     A3,A4             ; |34| <0,8>
||         CMPGTU  .L1     A5,A6,A1          ; |35| <0,8>  ^
||         LDBU    .D1T1   *+A7(1),A6        ; |35| <1,3>

   [ B0]   ADD     .L2     1,B6,B6           ; |45| <0,9>
|| [!B0]   ADD     .S2     2,B5,B5           ; |36| <0,9>
|| [!B0]   ZERO    .L1     A1                ; <0,9> Define a twin register
|| [!A1]   MV      .S1     A6,A5             ; |35| <0,9>  ^

   [!B0]   ZERO    .S1     A2                ; <0,10> Define a twin register
|| [ B0]   CMPGTU  .L1     A4,A5,A2          ; |38| <0,10>  ^
||         LDBU    .D1T1   *++A7(2),A3       ; |34| <2,0>

   [ B0]   MVK     .S2     0x2,B5            ; |44| <0,11>
|| [ B0]   CMPGTU  .L1     A5,A4,A1          ; |40| <0,11>
|| [ B0]   ZERO    .S1     A4                ; |42| <0,11>
|| [ A2]   STB     .D1T1   A4,*A8++          ; |39| <0,11>  ^
|| [ B1]   SUB     .L2     B1,1,B1           ; |31| <1,6>

   [ B0]   ZERO    .S1     A5                ; |43| <0,12>  ^
|| [ A1]   STB     .D1T1   A5,*A8++          ; |41| <0,12>  ^
||         CMPGTU  .L1     A4,A3,A1          ; |34| <1,7>
|| [ B1]   B       .S2     $C$L8             ; |31| <1,7>

$C$DW$L$_c62_pack_scale_x2_opt_2$3$E:
;** --------------------------------------------------------------------------*
$C$L9:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Из фидбэка следует, что ядро цикла загружено на 50%. За 5 таков обрабатывается 2 байта, что составляет удельную производительность в 2,5 такта на байт, что быстрее предыдущей программы на 120%. Удалось добиться одинакового количества необходимых ресурсов на сторонах процессора, которая совпадает с количеством тактов, которое занимает петля зависимости по данным. Видим прирост скорости по сравнению с эталоном в 157,7% для массива из 128 элементов и m=16.

Name	Calls	Incl Count Average
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	538.39
c62_pack_scale_x2_opt_2(unsigned char *, int, int, unsigned char *)	100	341.31

Рисунок 4: Сравнение эталона с программой, в которой использовано ручное разворачивание цикла в 2 раза (n = 128, m = 16)

3.3. Ручное разворачивание цикла в 4 раза

Больший прирост производительности позволяет получить программа с ручных разворачиванием цикла в 4 раза, представленная ниже. При этом накладывается ограничение на входной массив и на количество соседних байт в виде условия кратности четырём.

Листинг 10: Ручное разворачивание цикла в 4 раза

```
#include "pack_type.h"

int c62_pack_scale_x4_opt_3(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   int count = 0;
   int countb = 0;
   u8 max = 0;
   u8 max1 = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 4) {
      u8 x = *in++;
      u8 y = *in++;
      u8 z = *in++;
      u8 a = *in++;
      x = (x > z) ? x : z;
      y = (y > a) ? y : a;
      max = (x > max) ? x : max;
      max1 = (y > max1) ? y : max1;
      count += 4;
      if (count == m) {
         // конструкция для уменьшения завязки по данным
         if (max > max1)
            *result++ = max;
         if (max1 > max)
            *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb++;

      }
   }
   return countb;
}
```

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=12 -k
--preproc_with_compile --preproc_dependency="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 11: Фидбэк от компилятора для программы с ручным разворачиванием цикла в 4 раза

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c62_pack_scale_opt.c
;*      Loop source line                 : 31
;*      Loop opening brace source line   : 31
;*      Loop closing brace source line   : 48
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 5
;*      Unpartitioned Resource Bound     : 4
;*      Partitioned Resource Bound(*)    : 5
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3        2
;*      .S units                     0        2
;*      .D units                     2        2
;*      .M units                     0        0
;*      .X cross paths               2        2
;*      .T address paths             2        2
;*      Long read paths              1        1
;*      Long write paths             0        0
;*      Logical  ops (.LS)           2        1     (.L or .S unit)
;*      Addition ops (.LSD)          7        6     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             3        3
;*      Bound(.L .S .D .LS .LSD)     5*       5*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 5  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 2
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 4 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP PROLOG
           LDBU    .D1T1   *++A7(2),A3       ; |34| (P) <1,0>
           SUB     .L2     B9,1,B1

           ZERO    .S1     A5                ; |29|
||         CMPGTU  .L1     A4,A3,A1          ; |34| (P) <0,7>
|| [ B1]   B       .S2     $C$L5             ; |31| (P) <0,7>
```

```
;** --------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_x2_opt_2$3$B:

           CMPEQ   .L2     B5,B4,B0          ; |37| <0,8>
|| [!A1]   MV      .S1     A3,A4             ; |34| <0,8>
||         CMPGTU  .L1     A5,A6,A1          ; |35| <0,8>  ^
||         LDBU    .D1T1   *+A7(1),A6        ; |35| <1,3>

   [ B0]   ADD     .L2     1,B6,B6           ; |46| <0,9>
|| [!B0]   ADD     .S2     2,B5,B5           ; |36| <0,9>
|| [!B0]   ZERO    .L1     A1                ; <0,9> Define a twin register
|| [!A1]   MV      .S1     A6,A5             ; |35| <0,9>  ^

   [!B0]   ZERO    .S1     A2                ; <0,10> Define a twin register
|| [ B0]   CMPGTU  .L1     A4,A5,A2          ; |39| <0,10>  ^
||         LDBU    .D1T1   *++A7(2),A3       ; |34| <2,0>

   [ B0]   MVK     .S2     0x2,B5            ; |45| <0,11>
|| [ B0]   CMPGTU  .L1     A5,A4,A1          ; |41| <0,11>
|| [ B0]   ZERO    .S1     A4                ; |43| <0,11>
|| [ A2]   STB     .D1T1   A4,*A8++          ; |40| <0,11>  ^
|| [ B1]   SUB     .L2     B1,1,B1           ; |31| <1,6>

   [ B0]   ZERO    .S1     A5                ; |44| <0,12>  ^
|| [ A1]   STB     .D1T1   A5,*A8++          ; |42| <0,12>  ^
||         CMPGTU  .L1     A4,A3,A1          ; |34| <1,7>
|| [ B1]   B       .S2     $C$L5             ; |31| <1,7>

$C$DW$L$_c62_pack_scale_x2_opt_2$3$E:
;** --------------------------------------------------------------------------*
$C$L6:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Name	Calls	Incl Count Average
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	538.33
c62_pack_scale_x4_opt_3(unsigned char *, int, int, unsigned char *)	100	254.76

Рисунок 5: Сравнение эталона и программы с ручным разворачиванием цикла в 4 раза (m = 16; n = 128)

Как видим, необходимое количество тактов для итерационного интервала такое же, как и в прошлой программе, но теперь за 5 тактов обрабатывается 4 байта, что соответствует удельной производительности в 1,25 такта на элемент. Результаты профилирования для массива, состоящего из 128 элементов, и m=16 выглядят следующим образом: Прирост производительности составляет 211,3%.

3.4. Копирование памяти

Так как при m = 1 цикл поиска максимума вырождается в копирование памяти выгодно создать программу, которая будет копировать данные в выходной массив. На входной массив накладывается ограничение в виде кратности 4.

Листинг 12: Программа для копирования памяти

```
#include "pack_type.h"

int c62_pack_scale_1_opt_4(const u8 *in, int n, u8*restrict result)
{
   int i;
   _nassert(((int)(in) & 0x3)== 0);
   u32 *din = (u32 *) in;
   u32 *restrict dres = (u32 *) result;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 4) {
      u32 x = *din++;
      *dres++ = x;
   }
   return n;
}
```

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=24 -k
--preproc_with_compile --preproc_dependency="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 13: Фидбэк от компилятора для программы копирования памяти

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c62_pack_scale_opt.c
;*      Loop source line                 : 89
;*      Loop opening brace source line   : 89
;*      Loop closing brace source line   : 92
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 1
;*      Partitioned Resource Bound(*)    : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     1*       0
;*      .D units                     1*       1*
;*      .M units                     0        0
;*      .X cross paths               0        1*
;*      .T address paths             1*       1*
;*      Long read paths              0        1*
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        1     (.L or .S unit)
;*      Addition ops (.LSD)          0        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1*       1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
```

```
;*      Searching for software pipeline schedule at ...
;*         ii = 1  Schedule found with 7 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      Collapsed epilog stages       : 3
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 24 bytes
;*
;*      Minimum safe trip count       : 3
;*--------------------------------------------------------------------------*
$C$L23:    ; PIPED LOOP PROLOG

           SUB     .L2     B5,1,B0
||         LDW     .D1T1   *A4++,A3          ; |91| (P) <0,0>
||         B       .S1     $C$L24            ; |89| (P) <0,1>

           LDW     .D1T1   *A4++,A3          ; |91| (P) <1,0>
|| [ B0]   B       .S1     $C$L24            ; |89| (P) <1,1>
|| [ B0]   SUB     .L2     B0,1,B0           ; |89| (P) <2,0>

           LDW     .D1T1   *A4++,A3          ; |91| (P) <2,0>
|| [ B0]   B       .S1     $C$L24            ; |89| (P) <2,1>
|| [ B0]   SUB     .L2     B0,1,B0           ; |89| (P) <3,0>

           LDW     .D1T1   *A4++,A3          ; |91| (P) <3,0>
|| [ B0]   B       .S1     $C$L24            ; |89| (P) <3,1>
|| [ B0]   SUB     .L2     B0,1,B0           ; |89| (P) <4,0>

           MV      .L2X    A6,B5
||         LDW     .D1T1   *A4++,A3          ; |91| (P) <4,0>
|| [ B0]   B       .S1     $C$L24            ; |89| (P) <4,1>
|| [ B0]   SUB     .S2     B0,1,B0           ; |89| (P) <5,0>

;** --------------------------------------------------------------------------*
$C$L24:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_1_opt_4$3$B:

   [ A1]   SUB     .L1     A1,1,A1           ; <0,6>
|| [!A1]   STW     .D2T2   B4,*B5++          ; |91| <0,6>
||         MV      .L2X    A3,B4             ; |91| <1,5> Define a twin register
|| [ B0]   B       .S1     $C$L24            ; |89| <5,1>
|| [ B0]   SUB     .S2     B0,1,B0           ; |89| <6,0>
||         LDW     .D1T1   *A4++,A3          ; |91| <6,0>

$C$DW$L$_c62_pack_scale_1_opt_4$3$E:
;** --------------------------------------------------------------------------*
$C$L25:    ; PIPED LOOP EPILOG
;**--------------------------------------------------------------------------*
```

Как видно из фидбэка, за итерационный интервал программа загружает из памяти 4 байта. За 1 такт в цикле обрабатывается 4 элемента входного массива, что соответствует удельной производительности 0,25 тактов на элемент.

Name	Calls	Incl Count Average
c62_pack_scale_1_opt_4(unsigned char *, int, unsigned char *)	100	51.21
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	4498.35

Рисунок 6: Сравнение эталона и программы копирования памяти (n = 128, m = 1)

Из результатов профилирования следует, что прирост производительности при m=1 и составляет 8784,1% для массива из 128 элементов.

3.5. Программа, производящая упаковку по паре соседних байт

Для большего ускорения можно реализовать программы, производящие упаковку для 2, 3, 4 соседних байт. В данном пункте реализована программа, производящая упаковку по паре соседних байт. Цикл развёрнут в 2 раза для балансировки ресурсов по сторонам конвейера. От прагмы UNROLL было решено отказаться из-за увеличения оверхэда в 2 раза при ее использовании. Входной массив для использования данной программы должен быть кратен 4.

Листинг 14: Программа для упаковки массива по паре байт

```
#include "pack_type.h"

int c62_pack_scale_2_opt_5(const u8 *in, int n, u8*restrict result)
{
   int i;
   int count = 0;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 4) {
      u8 x = *in++;
      u8 y = *in++;
      *result++ = (x > y) ? x : y;
      u8 a = *in++;
      u8 b = *in++;
      *result++ = (a > b) ? a : b;
      count += 2;
   }
   return count;
}
```

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=8 -k
--preproc_with_compile --preproc_dependency="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 15: Фидбэк от компилятора для программы, выполняющей упаковку по паре байт

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../c62_pack_scale_opt.c
;*      Loop source line                : 85
;*      Loop opening brace source line  : 85
;*      Loop closing brace source line  : 93
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 1
;*      Unpartitioned Resource Bound    : 3
;*      Partitioned Resource Bound(*)   : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     1        1
;*      .S units                     1        0
;*      .D units                     3*       3*
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             3*       3*
;*      Long read paths              1        1
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          2        3     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        1
;*      Bound(.L .S .D .LS .LSD)     3*       3*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 3  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 2
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 8 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L17:    ; PIPED LOOP PROLOG

           LDBU    .D1T1   *++A3(4),A6       ; |86| (P) <0,0>
||         LDBU    .D2T2   *++B4(4),B7       ; |89| (P) <0,0>
||         B       .S1     $C$L30            ; |85| (P) <0,3>

           MV      .L1X    B9,A4
||         ADD     .L2     B1,B1,B5          ; |92|
||         LDBU    .D1T1   *+A3(1),A5        ; |86| (P) <0,1>
||         LDBU    .D2T2   *+B4(1),B6        ; |89| (P) <0,1>

           SUB     .L2     B1,1,B1
||         ADD     .S2     1,B9,B5
||         MVK     .S1     0x1,A2            ; init prolog collapse predicate
||         MV      .L1X    B5,A7             ; |92| Define a twin register
```

```
;** --------------------------------------------------------------------------*
$C$L18:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_opt_5_2$3$B:

           CMPGTU  .L1     A5,A6,A1          ; |86| <0,6>
||         CMPGTU  .L2     B6,B7,B0          ; |89| <0,6>
|| [ B1]   B       .S1     $C$L30            ; |85| <1,3>
||         LDBU    .D1T1   *++A3(4),A6       ; |86| <2,0>
||         LDBU    .D2T2   *++B4(4),B7       ; |89| <2,0>

   [!A1]   MV      .S1     A6,A5             ; |86| <0,7>  ^
|| [!B0]   MV      .S2     B7,B6             ; |89| <0,7>  ^
||         LDBU    .D1T1   *+A3(1),A5        ; |86| <2,1>
||         LDBU    .D2T2   *+B4(1),B6        ; |89| <2,1>

   [ A2]   SUB     .L1     A2,1,A2           ; <0,8>
|| [!A2]   STB     .D1T1   A5,*++A4(2)       ; |86| <0,8>
|| [!A2]   STB     .D2T2   B6,*++B5(2)       ; |89| <0,8>
|| [ B1]   SUB     .L2     B1,1,B1           ; |85| <2,2>

$C$DW$L$_c62_pack_scale_opt_5_2$3$E:
;** --------------------------------------------------------------------------*
$C$L19:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Ядро цикла загружено на 54%. За 3 такта обрабатывается 4 элемента массива, что составляет 0,75 такта на элемент. При профилировании для m=2 получаем прирост производительности 2394,7% для массива из 128 элементов.

Name	Calls	Incl Count Average
c62_pack_scale_2_opt_5(unsigned char *, int, unsigned char *)	100	118.36
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char...	100	2834.34

Рисунок 7: Сравнение эталона и программы, производящей упаковку данных по паре байт (n = 128, m = 2)

3.6. Программа, производящая упаковку по 3 байта

В данном пункте представлена программа, упаковывающая 3 соседних байта в 1 для повышения производительности. Прагма UNROLL использована для балансировки ресурсов по сторонам конвейера.

Листинг 16: Программа для упаковки массива по три байта

```
#include "pack_type.h"

int c62_pack_scale_3_opt_6(const u8 *in, int n, u8*restrict result)
{
   int i;
   int count = 0;
   u8 max;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
   for (i=0; i<n; i+=3){
      u8 x = *in++;
```

```
        u8 y = *in++;
        u8 z = *in++;
        max = (z > y) ? z : y;
        *result++ = (x > max) ? x : max;
        count++;
    }
    return count;
}
```

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=18 -k
--preproc_with_compile --preproc_dependency="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 17: Фидбэк от компилятора для программы, выполняющей упаковку по три байта

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c62_pack_scale_opt.c
;*      Loop source line                 : 120
;*      Loop opening brace source line   : 120
;*      Loop closing brace source line   : 127
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 1
;*      Unpartitioned Resource Bound     : 4
;*      Partitioned Resource Bound(*)    : 4
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     2        2
;*      .S units                     1        0
;*      .D units                     4*       4*
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             4*       4*
;*      Long read paths              1        1
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          4        5     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             2        1
;*      Bound(.L .S .D .LS .LSD)     4*       4*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 4  Register is live too long
;*         ii = 4  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 3
;*      Prolog not entirely removed
```

```
;*      Collapsed prolog stages       : 2
;*
;*      Minimum required memory pad   : 18 bytes
;*
;*      Minimum safe trip count       : 1 (after unrolling)
;*--------------------------------------------------------------------------*
$C$L18:    ; PIPED LOOP PROLOG

           SUB     .L1     A7,2,A4
||         ADD     .L2X    3,A3,B8
||         LDBU    .D1T1   *++A3(6),A6       ; |121| (P) <0,0>

           ZERO    .L1     A0
||         LDBU    .D1T1   *+A3(2),A8        ; |121| (P) <0,1>
||         B       .S1     $C$L19            ; |120| (P) <0,9>

           ADD     .S2     1,B1,B1
||         AND     .L1X    -2,B3,A10         ; |126|
||         ADD     .L2X    -1,A7,B9
||         SET     .S1     A0,0xf,0xf,A2     ; init prolog collapse predicate
||         LDBU    .D1T1   *+A3(1),A7        ; |121| (P) <0,2>

;** --------------------------------------------------------------------------*
$C$L19:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_3_opt_6$5$B:

   [!A2]   STB     .D1T1   A5,*++A4(2)       ; |121| <0,11>
|| [!B0]   MV      .L2     B2,B4             ; |121| <0,11>  ^
|| [ B0]   MV      .S2     B5,B4             ; |121| <0,11>  ^
||         MV      .S1     A6,A9             ; |121| <1,7> Split a long life
||         CMPGTU  .L1     A7,A8,A1          ; |121| <1,7>
||         LDBU    .D2T2   *++B8(6),B6       ; |121| <2,3>

           CMPGTU  .L2     B4,B7,B0          ; |121| <0,12>
|| [ B1]   SUB     .S2     B1,1,B1           ; |120| <1,8>
|| [ A1]   MV      .L1     A7,A5             ; |121| <1,8>  ^
|| [!A1]   MV      .S1     A8,A5             ; |121| <1,8>  ^
||         LDBU    .D2T2   *+B8(1),B5        ; |121| <2,4>
||         LDBU    .D1T1   *++A3(6),A6       ; |121| <3,0>

   [ A2]   MPYSU   .M1     2,A2,A2           ; <0,13>
|| [!B0]   MV      .L2     B7,B4             ; |121| <0,13>  ^
||         CMPGTU  .L1     A5,A9,A1          ; |121| <1,9>
|| [ B1]   B       .S1     $C$L19            ; |120| <1,9>
||         LDBU    .D2T2   *+B8(2),B2        ; |121| <2,5>
||         LDBU    .D1T1   *+A3(2),A8        ; |121| <3,1>

   [!A2]   STB     .D2T2   B4,*++B9(2)       ; |121| <0,14>
||         MV      .S2     B6,B7             ; |121| <1,10> Split a long life
|| [!A1]   MV      .L1     A9,A5             ; |121| <1,10>  ^
||         CMPGTU  .L2     B5,B2,B0          ; |121| <1,10>
||         LDBU    .D1T1   *+A3(1),A7        ; |121| <3,2>

$C$DW$L$_c62_pack_scale_3_opt_6$5$E:
;** --------------------------------------------------------------------------*
$C$L20:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

За 4 такта обрабатывается 6 байт, что соответствует удельной производительности 0,67 такта на элемент. При профилировании для m=3 прирост производительности составляет 956,2% для массива из 120 элементов.

Name	Calls	Incl Count Average
c62_pack_scale_3_opt_6(unsigned char *, int, unsigned char *)	100	166.78
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	1577.39

Рисунок 8: Сравнение эталона и программы, производящей упаковку по 3 байта, для m = 3 и n = 128

3.7. Программа, производящая упаковку по 4 байта

Для увеличения скорости расчёта для m=4 реализуем программу без вложенного цикла, упаковывающую входной массив по 4 байта. Прагма UNROLL использована для балансировки ресурсов по сторонам конвейера. Входной массив должен быть кратен 8.

Листинг 18: Программа для упаковки массива по четыре байта

```
#include "pack_type.h"

int c62_pack_scale_4_opt_7(const u8 *in, int n, u8*restrict result)
{
   int i;
   int count = 0;
   u8 max, max1;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
   for (i=0; i<n; i+=4){
      u8 x = *in++;
      u8 y = *in++;
      u8 z = *in++;
      u8 a = *in++;
      max = (z > a) ? z : a;
      max1 = (x > y) ? x : y;
      *result++ = (max > max1) ? max : max1;
      count++;
   }
   return count;
}
```

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=24 -k
--preproc_with_compile --preproc_dependency="c62_pack_scale_opt.pp"
"../c62_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 19: Фидбэк от компилятора для программы, выполняющей упаковку по четыре байта

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
```

```
;*
;*      Loop found in file               : ../c62_pack_scale_opt.c
;*      Loop source line                 : 137
;*      Loop opening brace source line   : 137
;*      Loop closing brace source line   : 146
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 3
;*      Unpartitioned Resource Bound     : 5
;*      Partitioned Resource Bound(*)    : 5
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3        3
;*      .S units                     1        0
;*      .D units                     5*       5*
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             5*       5*
;*      Long read paths              1        1
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          6        7     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             2        2
;*      Bound(.L .S .D .LS .LSD)     5*       5*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 5  Cannot allocate machine registers
;*                   Regs Live Always   :  2/3  (A/B-side)
;*                   Max Regs Live      : 10/12
;*                   Max Cond Regs Live :  2/3
;*         ii = 5  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 3
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 24 bytes
;*
;*      Minimum safe trip count       : 1 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L10:    ; PIPED LOOP PROLOG

           LDBU    .D2T2   *-B4(2),B8        ; |138| (P) <0,3>
||         LDBU    .D1T1   *-A4(7),A3        ; |138| (P) <0,3>  ^

           NOP             1

           LDBU    .D2T2   *+B4(7),B10       ; |138| (P) <1,0>
||         LDBU    .D1T1   *+A4(2),A3        ; |138| (P) <1,0>

           CMPGTU  .L2     B8,B10,B1         ; |138| (P) <0,6>
||         CMPGTU  .L1     A6,A3,A1          ; |138| (P) <0,6>
||         LDBU    .D2T2   *++B4(8),B8       ; |138| (P) <1,1>
||         LDBU    .D1T1   *+A4(3),A6        ; |138| (P) <1,1>
```

```
   [ B1]   MV      .L2     B8,B7           ; |138| (P) <0,7>
|| [!B1]   MV      .S2     B10,B7          ; |138| (P) <0,7>
|| [ A1]   MV      .L1     A6,A9           ; |138| (P) <0,7>
|| [!A1]   MV      .S1     A3,A9           ; |138| (P) <0,7>
||         LDBU    .D2T2   *-B4(3),B9      ; |138| (P) <1,2>
||         LDBU    .D1T1   *A4++(8),A7     ; |138| (P) <1,2>  ^

           SUB     .S2     B2,1,B2
||         SUB     .S1     A5,2,A5
||         CMPGTU  .L2     B8,B9,B0        ; |138| (P) <0,8>
||         CMPGTU  .L1     A3,A7,A1        ; |138| (P) <0,8>  ^
||         LDBU    .D2T2   *-B4(2),B8      ; |138| (P) <1,3>
||         LDBU    .D1T1   *-A4(7),A3      ; |138| (P) <1,3>  ^

           AND     .L1X    -2,B5,A10       ; |145|
||         ADD     .L2X    1,A5,B5
|| [!B0]   MV      .S2     B9,B6           ; |138| (P) <0,9>
|| [ B0]   MV      .D2     B8,B6           ; |138| (P) <0,9>
|| [ A1]   MV      .S1     A3,A0           ; |138| (P) <0,9>
|| [!A1]   MV      .D1     A7,A0           ; |138| (P) <0,9>  ^

           CMPGTU  .L2     B6,B7,B0        ; |138| (P) <0,10>
|| [ B2]   B       .S1     $C$L11          ; |137| (P) <0,10>
||         CMPGTU  .L1     A0,A9,A2        ; |138| (P) <0,10>
|| [ B2]   SUB     .S2     B2,1,B2         ; |137| (P) <1,5>
||         LDBU    .D2T2   *+B4(7),B10     ; |138| (P) <2,0>
||         LDBU    .D1T1   *+A4(2),A3      ; |138| (P) <2,0>

;** --------------------------------------------------------------------------*
$C$L11:    ; PIPED LOOP KERNEL
$C$DW$L$_c62_pack_scale_4_opt_7$4$B:

   [!B0]   MV      .S2     B7,B3           ; |138| <0,11>
|| [!A2]   MV      .S1     A9,A8           ; |138| <0,11>
||         CMPGTU  .L2     B8,B10,B1       ; |138| <1,6>
||         CMPGTU  .L1     A6,A3,A1        ; |138| <1,6>
||         LDBU    .D2T2   *++B4(8),B8     ; |138| <2,1>
||         LDBU    .D1T1   *+A4(3),A6      ; |138| <2,1>

   [ B1]   MV      .L2     B8,B7           ; |138| <1,7>
|| [!B1]   MV      .S2     B10,B7          ; |138| <1,7>
|| [ A1]   MV      .L1     A6,A9           ; |138| <1,7>
|| [!A1]   MV      .S1     A3,A9           ; |138| <1,7>
||         LDBU    .D2T2   *-B4(3),B9      ; |138| <2,2>
||         LDBU    .D1T1   *A4++(8),A7     ; |138| <2,2>  ^

   [ B0]   MV      .S2     B6,B3           ; |138| <0,13>
|| [ A2]   MV      .S1     A0,A8           ; |138| <0,13>
||         CMPGTU  .L2     B8,B9,B0        ; |138| <1,8>
||         CMPGTU  .L1     A3,A7,A1        ; |138| <1,8>  ^
||         LDBU    .D2T2   *-B4(2),B8      ; |138| <2,3>
||         LDBU    .D1T1   *-A4(7),A3      ; |138| <2,3>  ^

           STB     .D2T2   B3,*++B5(2)     ; |138| <0,14>
||         STB     .D1T1   A8,*++A5(2)     ; |138| <0,14>
|| [!B0]   MV      .L2     B9,B6           ; |138| <1,9>
|| [ B0]   MV      .S2     B8,B6           ; |138| <1,9>
```

```
||  [ A1]   MV      .L1     A3,A0           ; |138| <1,9>
||  [!A1]   MV      .S1     A7,A0           ; |138| <1,9>  ^

            CMPGTU  .L2     B6,B7,B0        ; |138| <1,10>
||  [ B2]   B       .S1     $C$L11          ; |137| <1,10>
||          CMPGTU  .L1     A0,A9,A2        ; |138| <1,10>
||  [ B2]   SUB     .S2     B2,1,B2         ; |137| <2,5>
||          LDBU    .D2T2   *+B4(7),B10     ; |138| <3,0>
||          LDBU    .D1T1   *+A4(2),A3      ; |138| <3,0>

$C$DW$L$_c62_pack_scale_4_opt_7$4$E:
;** --------------------------------------------------------------------------*
$C$L12:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

За 5 тактов обрабатывается 8 элементов массива, что соответствует удельной производительности в 0,625 такта на элемент. При профилировании для m=4 массива из 128 элементов получаем следующие результаты:

Name	Calls	Incl Count Average
c62_pack_scale_4_opt_7(unsigned char *, int, unsigned char *)	100	135.63
c62_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	1330.39

Рисунок 9: Сравнение эталона и программы, производящей упаковку по 4 байта, для m = 4, n = 128

Прирост производительности составляет 980,9%.

4. Создание оптимального алгоритма (C64x)

4.1. Запуск программы с ручным разворачиванием цикла в 4 раза на ядре C64

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

Листинг 20: Программа с ручным разворачиванием цикла в 4 раза

```c
#include "pack_type.h"

int c64_pack_scale_x4_opt_1(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   int count = 0;
   int countb = 0;
   u8 max = 0;
   u8 max1 = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 4) {
      u8 x = *in++;
      u8 y = *in++;
      u8 z = *in++;
      u8 a = *in++;
      x = (x > z) ? x : z;
```

```
      y = (y > a) ? y : a;
      max = (x > max) ? x : max;
      max1 = (y > max1) ? y : max1;
      count += 4;
      if (count == m) {
         // структура для уменьшения завязки по данным
         if (max > max1)
            *result++ = max;
         if (max1 > max)
            *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb++;

      }
   }
   return countb;
}
```

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 --speculate_loads=12 -k
--preproc_with_compile --preproc_dependency="c64_pack_scale_opt.pp"
"../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 21: Фидбэк от компилятора для ручного разворачивания цикла в 4 раза

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../c64_pack_scale_opt.c
;*      Loop source line                : 11
;*      Loop opening brace source line  : 11
;*      Loop closing brace source line  : 33
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 5
;*      Unpartitioned Resource Bound    : 5
;*      Partitioned Resource Bound(*)   : 5
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     2        5*
;*      .S units                     3        2
;*      .D units                     4        2
;*      .M units                     0        0
;*      .X cross paths               0        3
;*      .T address paths             4        2
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          2        5     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             3        4
```

```
;*      Bound(.L .S .D .LS .LSD)     4        5*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 5  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 3
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 12 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L15:    ; PIPED LOOP PROLOG
           LDBU    .D1T1   *A7++(4),A5       ; |18| (P) <1,0>
           NOP             1
           LDBU    .D1T1   *-A7(3),A3        ; |19| (P) <1,2>

           MV      .L2X    A6,B8
||         MAXU4   .L1     A5,A16,A5         ; |18| (P) <0,8>
||         LDBU    .D1T1   *-A7(2),A6        ; |18| (P) <1,3>

           MAXU4   .L2X    B4,A4,B4          ; |19| (P) <0,9>
||         EXTU    .S1     A5,24,24,A4       ; |18| (P) <0,9>
||         LDBU    .D1T2   *-A7(1),B4        ; |19| (P) <1,4>

           LDBU    .D1T1   *A7++(4),A5       ; |18| (P) <2,0>

           SUB     .L1     A3,1,A0
||         EXTU    .S2     B4,24,24,B4       ; |19| (P) <0,11>

           ZERO    .S2     B5                ; |9|
||         CMPEQ   .L2     B7,B8,B0          ; |21| (P) <0,12>
|| [ A0]   BDEC    .S1     $C$L16,A0         ; |11| (P) <0,12>
||         MAXU4   .L1     A9,A4,A4          ; |18| (P) <0,12>  ^
||         LDBU    .D1T1   *-A7(3),A3        ; |19| (P) <2,2>

;** ----------------------------------------------------------------------------*
$C$L16:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_x4_opt_1$3$B:

   [!B0]   ADD     .S2     4,B7,B7           ; |20| <0,13>
||         MAXU4   .L2     B5,B4,B9          ; |19| <0,13>  ^
||         EXTU    .S1     A4,24,24,A9       ; |18| <0,13>  ^
||         MAXU4   .L1     A6,A5,A4          ; |18| <1,8>
||         LDBU    .D1T1   *-A7(2),A6        ; |18| <2,3>

   [ B0]   ADD     .L1     1,A8,A8           ; |30| <0,14>
|| [ B0]   MVK     .D2     0x4,B7            ; |29| <0,14>
||         EXTU    .S2     B9,24,24,B5       ; |19| <0,14>  ^
||         MAXU4   .L2X    B4,A3,B4          ; |19| <1,9>
||         EXTU    .S1     A4,24,24,A4       ; |18| <1,9>
||         LDBU    .D1T2   *-A7(1),B4        ; |19| <2,4>

   [!B0]   ZERO    .S2     B1                ; <0,15>
|| [!B0]   ZERO    .D2     B2                ; <0,15>
```

```
||	[ B0]   CMPGTU  .L2X    A9,B5,B2          ; |23| <0,15>  ^
||          LDBU    .D1T1   *A7++(4),A5       ; |18| <3,0>

    [ B0]   CMPGTU  .L2X    B5,A9,B1          ; |25| <0,16>
||  [ B0]   ZERO    .L1     A9                ; |27| <0,16>  ^
||  [ B2]   STB     .D2T1   A9,*B6++          ; |24| <0,16>  ^
||          EXTU    .S2     B4,24,24,B4       ; |19| <1,11>

    [ B0]   ZERO    .S2     B5                ; |28| <0,17>  ^
||  [ B1]   STB     .D2T2   B5,*B6++          ; |26| <0,17>  ^
||          CMPEQ   .L2     B7,B8,B0          ; |21| <1,12>
||  [ A0]   BDEC    .S1     $C$L16,A0         ; |11| <1,12>
||          MAXU4   .L1     A9,A4,A4          ; |18| <1,12>  ^
||          LDBU    .D1T1   *-A7(3),A3        ; |19| <3,2>

$C$DW$L$_c64_pack_scale_x4_opt_1$3$E:
;** --------------------------------------------------------------------------*
$C$L17:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Можно отметить, что вычислительные ресурсы распределены не равномерно по сторонам конвейера, но удельная производительно такая же, как и для ядра C62. Это связано с тем, что ядро C64 использует встроенные инструкции сравнения _maxu4, так как входные данные и переменная для записи максимума объявлены типом unsigned char, который занимает 8 бит.

Эталонная программа для ядра C64:

Листинг 22: Эталонный алгоритм для уплотнения РЛС сигнала

```c
// Эталонная программа для уплотнения РЛС сигнала (ядро C64)
#include "pack_type.h"

int c64_pack_scale_no_opt(const u8 *in, int n, int m, u8* result)
{
   int k;
   u8 max;
   int count = 0;
   while (n >= m) {
      max = 0;
      for (k = 0; k < m; k++) {
         u8 x = *in++;
         max = (x > max) ? x : max;
      }
      *result++ = max;
      n -= m;
      count++;
   }
   return count;
}
```

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="c64_pack_scale_no_opt.pp"  "../c64_pack_scale_no_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 23: Фидбэк от компилятора для эталонной программы на ядре C64

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../c64_pack_scale_no_opt.c
;*      Loop source line                : 10
;*      Loop opening brace source line  : 10
;*      Loop closing brace source line  : 13
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound    : 1
;*      Partitioned Resource Bound(*)   : 1
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     1*       0
;*      .S units                     1*       1*
;*      .D units                     1*       0
;*      .M units                     0        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)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1*       1*
;*      Bound(.L .S .D .LS .LSD)     1*       1*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      For further improvement on this loop, try option -mh3
;*
;*      Minimum safe trip count       : 3
;*----------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP PROLOG
   [ B0]   BDEC    .S2     $C$L5,B0          ; |10| (P) <0,1>
           LDBU    .D1T1   *A4++,A3          ; |11| (P) <0,0>
   [ B0]   BDEC    .S2     $C$L5,B0          ; |10| (P) <1,1>
```

```
           MVK     .L1     0x1,A0            ; init prolog collapse predicate
||         LDBU    .D1T1   *A4++,A3          ; |11| (P) <1,0>

;** --------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_no_opt$8$B:

           MAXU4   .L1     A5,A3,A3          ; |11| <0,5>  ^
|| [ B0]   BDEC    .S2     $C$L5,B0          ; |10| <2,1>

   [ A0]   SUB     .L1     A0,1,A0           ; <0,6>
|| [!A0]   EXTU    .S1     A3,24,24,A5       ; |11| <0,6>  ^
||         LDBU    .D1T1   *A4++,A3          ; |11| <3,0>

$C$DW$L$_c64_pack_scale_no_opt$8$E:
;** --------------------------------------------------------------------------*
$C$L6:    ; PIPED LOOP EPILOG
           MAXU4   .L1     A5,A3,A3          ; |11| (E) <1,5>  ^
           EXTU    .S1     A3,24,24,A5       ; |11| (E) <1,6>  ^
           MAXU4   .L1     A5,A3,A3          ; |11| (E) <2,5>  ^

           MVC     .S2     B5,CSR            ; interrupts on
||         EXTU    .S1     A3,24,24,A5       ; |11| (E) <2,6>  ^

;** --------------------------------------------------------------------------*
```

Name	Calls	Incl Count Average
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	546.38
c64_pack_scale_x4_opt_1(unsigned char *, int, int, unsigned char *)	100	186.40

Рисунок 10: Сравнение эталона и программы с разворачиванием цикла в 4 раза (m = 16, n = 128)

Из рисунка 10 следует, что эталонная программа на ядре C64 работает медленнее, чем эталонная программа на ядре C62. Это связано с использованием компилятором команд, увеличивающих эпилог цикла. Прирост скорости составляет 293,12%.

4.2. Использование интринсиков

Ядро C64 позволяет использовать интринсик _maxu4, производящий сравнение 4 пар байт, что сокращает количество тактов для обработки 1 элемента массива. Также использованы интринсики _packl4 и _packh4, позволяющие уплотнить расписание совместно с ручным разворачиванием цикла. В данной программе адрес входного массива должен быть кратен 16.

Листинг 24: Программа, использующая интринсики _maxu4, _packl4, _packh4

```
#include "pack_type.h"

int c64_pack_scale_x4_opt_2(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   u32 max = 0;
   u32 max1 = 0;
```

```
   u32 max2 = 0;
   u32 max3 = 0;
   int count = 0;
   int countb = 0;
   int m1 = m >> 2;
   u32 *din = (u32 *) in;
   u32 *dres = (u32 *) result;
   int m12 = _mpy(m1, 2);
   int m13 = _mpy(m1, 3);
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 16) {
      u32 v = *(din + m13);
      u32 z = *(din + m12);
      u32 y = *(din + m1);
      u32 x = *din++;
      max = _maxu4(max, v);
      max1 = _maxu4(max1, z);
      max2 = _maxu4(max2, y);
      max3 = _maxu4(max3, x);
      count += 4;
      if (count == m) {
         u32 l = _packl4(max, max1);
         u32 h = _packh4(max, max1);
         u32 maxlh = _maxu4(l, h);
         u32 l1 = _packl4(max2, max3);
         u32 h1 = _packh4(max2, max3);
         u32 maxlh1 = _maxu4(l1, h1);
         u32 pack16l = _packl4(maxlh, maxlh1);
         u32 pack16h = _packh4(maxlh, maxlh1);
         *dres++ = _maxu4(pack16l, pack16h);
         max = 0;
         max1 = 0;
         max2 = 0;
         max3 = 0;
         count = 0;
         countb += 4;
         din += m13;
      }
   }
   return countb;
}
```

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="c64_pack_scale_opt.pp"  "../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 25: Фидбэк программы, использующей _maxu4, _packl4, _packh4

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../c64_pack_scale_opt.c
```

```
;*      Loop source line                 : 51
;*      Loop opening brace source line   : 51
;*      Loop closing brace source line   : 79
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 3
;*      Unpartitioned Resource Bound     : 7
;*      Partitioned Resource Bound(*)    : 7
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     7*       7*
;*      .S units                     0        1
;*      .D units                     4        1
;*      .M units                     0        0
;*      .X cross paths               3        4
;*      .T address paths             4        1
;*      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)             4        4
;*      Bound(.L .S .D .LS .LSD)     6        5
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 7  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      Minimum safe trip count       : 3
;*----------------------------------------------------------------------------*
$C$L11:    ; PIPED LOOP PROLOG

           LDW     .D1T1   *A3,A4           ; |55| (P) <0,0>
||         ADD     .L1     A3,A9,A4         ; |54| (P) <0,0>

           LDW     .D1T1   *A4,A4           ; |54| (P) <0,1>
||         ADD     .L2X    A3,B7,B16        ; |53| (P) <0,1>

           MV      .L2X    A8,B16
||         LDW     .D2T2   *B16,B18         ; |53| (P) <0,2>

           MV      .L1X    B18,A16
||         ADD     .S1     A17,A3,A8        ; |52| (P) <0,3>

           CMPEQ   .L1X    B16,A16,A1       ; |61| (P) <0,4>  ^
||         LDW     .D1T1   *A8,A4           ; |52| (P) <0,4>

   [ A1]   ADD     .L2     4,B17,B17        ; |76| (P) <0,5>
||         MAXU4   .L1     A7,A4,A7         ; |59| (P) <0,5>
```

```
   [!A1]   ADD     .S1     4,A3,A3           ; |55| (P) <0,6>
||         MAXU4   .L1     A6,A4,A6          ; |58| (P) <0,6>
|| [ A1]   ADD     .D1     4,A8,A3           ; |77| (P) <0,6>

           MVD     .M1     A1,A0             ; |61| (P) <0,7> Split a long life
|| [ A1]   MVK     .L2     0x4,B16           ; |75| (P) <0,7>  ^
|| [ A1]   PACKH4  .L1     A6,A7,A8          ; |70| (P) <0,7>
||         LDW     .D1T1   *A3,A5            ; |55| (P) <1,0>
||         ADD     .S1     A3,A9,A4          ; |54| (P) <1,0>

   [ A1]   PACKL4  .L1     A6,A7,A19         ; |70| (P) <0,8>
||         MAXU4   .L2     B9,B18,B9         ; |57| (P) <0,8>
||         LDW     .D1T1   *A4,A5            ; |54| (P) <1,1>
||         ADD     .S2X    A3,B7,B18         ; |53| (P) <1,1>

   [!A1]   ADD     .S2     4,B16,B16         ; |60| (P) <0,9>  ^
||         MAXU4   .L2X    B8,A4,B8          ; |56| (P) <0,9>
||         LDW     .D2T2   *B18,B18          ; |53| (P) <1,2>

           MV      .S1X    B6,A4
||         SUB     .S2     B0,3,B0
|| [ A1]   MAXU4   .L1     A19,A8,A18        ; |70| (P) <0,10>
|| [ A1]   PACKL4  .L2     B8,B9,B6          ; |70| (P) <0,10>
||         ADD     .D1     A17,A3,A8         ; |52| (P) <1,3>

;** --------------------------------------------------------------------------*
$C$L12:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_x4_opt_2$6$B:

   [ A0]   ZERO    .S1     A7                ; |74| <0,11>
|| [ A0]   PACKH4  .L2     B8,B9,B5          ; |70| <0,11>
||         CMPEQ   .L1X    B16,A16,A1        ; |61| <1,4>  ^
||         LDW     .D1T1   *A8,A5            ; |52| <1,4>

   [ A0]   ZERO    .S1     A6                ; |73| <0,12>
|| [ B0]   BDEC    .S2     $C$L12,B0         ; |51| <0,12>
|| [ A0]   MAXU4   .L2     B6,B5,B5          ; |70| <0,12>
|| [ A1]   ADD     .D2     4,B17,B17         ; |76| <1,5>
||         MAXU4   .L1     A7,A5,A7          ; |59| <1,5>

   [ A0]   PACKH4  .L2X    B5,A18,B4         ; |70| <0,13>
|| [!A1]   ADD     .S1     4,A3,A3           ; |55| <1,6>
||         MAXU4   .L1     A6,A5,A6          ; |58| <1,6>
|| [ A1]   ADD     .D1     4,A8,A3           ; |77| <1,6>

   [ A0]   ZERO    .D2     B9                ; |72| <0,14>
|| [ A0]   PACKL4  .L2X    B5,A18,B5         ; |70| <0,14>
||         MVD     .M1     A1,A0             ; |61| <1,7> Split a long life
|| [ A1]   MVK     .S2     0x4,B16           ; |75| <1,7>  ^
|| [ A1]   PACKH4  .L1     A6,A7,A8          ; |70| <1,7>
||         LDW     .D1T1   *A3,A5            ; |55| <2,0>
||         ADD     .S1     A3,A9,A5          ; |54| <2,0>

   [ A0]   ZERO    .S2     B8                ; |71| <0,15>
||         MV      .S1X    B4,A20            ; |70| <0,15> Define a twin register
|| [ A1]   PACKL4  .L1     A6,A7,A19         ; |70| <1,8>
||         MAXU4   .L2     B9,B18,B9         ; |57| <1,8>
```

```
||         LDW     .D1T1   *A5,A5           ; |54| <2,1>
||         ADD     .D2X    A3,B7,B18        ; |53| <2,1>

   [ A0]   MAXU4   .L1X    B5,A20,A5        ; |70| <0,16>
|| [!A1]   ADD     .S2     4,B16,B16        ; |60| <1,9>  ^
||         MAXU4   .L2X    B8,A5,B8         ; |56| <1,9>
||         LDW     .D2T2   *B18,B18         ; |53| <2,2>

   [ A0]   STW     .D1T1   A5,*A4++         ; |70| <0,17>
|| [ A1]   MAXU4   .L1     A19,A8,A18       ; |70| <1,10>
|| [ A1]   PACKL4  .L2     B8,B9,B6         ; |70| <1,10>
||         ADD     .S1     A17,A3,A8        ; |52| <2,3>

$C$DW$L$_c64_pack_scale_x4_opt_2$6$E:
;** --------------------------------------------------------------------------*
$C$L13:    ; PIPED LOOP EPILOG

   [ A0]   ZERO    .S1     A7               ; |74| (E) <1,11>
|| [ A0]   PACKH4  .L2     B8,B9,B5         ; |70| (E) <1,11>
||         CMPEQ   .L1X    B16,A16,A1       ; |61| (E) <2,4>  ^
||         LDW     .D1T1   *A8,A5           ; |52| (E) <2,4>

   [ A0]   ZERO    .S1     A6               ; |73| (E) <1,12>
|| [ A0]   MAXU4   .L2     B6,B5,B5         ; |70| (E) <1,12>
|| [ A1]   ADD     .S2     4,B17,B17        ; |76| (E) <2,5>
||         MAXU4   .L1     A7,A5,A7         ; |59| (E) <2,5>

   [ A0]   PACKH4  .L2X    B5,A18,B4        ; |70| (E) <1,13>
|| [!A1]   ADD     .S1     4,A3,A3          ; |55| (E) <2,6>
||         MAXU4   .L1     A6,A5,A6         ; |58| (E) <2,6>
|| [ A1]   ADD     .D1     4,A8,A3          ; |77| (E) <2,6>

   [ A0]   ZERO    .S2     B9               ; |72| (E) <1,14>
|| [ A0]   PACKL4  .L2X    B5,A18,B5        ; |70| (E) <1,14>
||         MVD     .M1     A1,A0            ; |61| (E) <2,7> Split a long life
|| [ A1]   MVK     .D2     0x4,B16          ; |75| (E) <2,7>  ^
|| [ A1]   PACKH4  .L1     A6,A7,A8         ; |70| (E) <2,7>

   [ A0]   ZERO    .S2     B8               ; |71| (E) <1,15>
||         MV      .S1X    B4,A20           ; |70| (E) <1,15> Define a twin
register
|| [ A1]   PACKL4  .L1     A6,A7,A19        ; |70| (E) <2,8>
||         MAXU4   .L2     B9,B18,B9        ; |57| (E) <2,8>

   [ A0]   MAXU4   .L1X    B5,A20,A5        ; |70| (E) <1,16>
|| [!A1]   ADD     .S2     4,B16,B16        ; |60| (E) <2,9>  ^
||         MAXU4   .L2X    B8,A5,B8         ; |56| (E) <2,9>

           MVC     .S2     B19,CSR          ; interrupts on
|| [ A0]   STW     .D1T1   A5,*A4++         ; |70| (E) <1,17>
|| [ A1]   MAXU4   .L1     A19,A8,A18       ; |70| (E) <2,10>
|| [ A1]   PACKL4  .L2     B8,B9,B6         ; |70| (E) <2,10>

;** --------------------------------------------------------------------------*
```

В данной программе ядро цикла загружено на 60,7%, за 7 тактов обрабатывается 16

элементов, а в предыдущей - за 5 тактов 4 элемента. Результаты профилирования для обработки массива из 128 элементов и m=16 представлены ниже на рисунке 11. Прирост скорости составил 474,8%.

Name	Calls	Incl Count Average
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	558.26
c64_pack_scale_x4_opt_2(unsigned char *, int, int, unsigned char *)	100	117.57

Рисунок 11: Сравнение эталона и программы, использующей maxu4 (n = 128, m = 16)

4.3. Загрузка элементов массива с помощью инструкции lddw

Максимальное количество байт, которое возможно загрузить 1 инструкцией, равняется 8, с помощью инструкции lddw. Входной массив объявлен типом char, который занимает 8 бит. Если загружать элементы массива с выравниванием указателя к типу long long, то компилятор составит расписание с использованием инструкции lddw. Это позволяет выполнить 1 инструкции maxu4, который использует операнды типа int. При этом ограничение, накладываемое на входной массив и на количество соседних элементов, увеличивается до условия кратности восьми. Прагма UNROLL применена для лучшей балансировки по сторонам конвейера. Максимальная скорость работы программы достигается для массива, адрес которого кратен 16.

Листинг 26: Программа, производящая загрузку по 8 байт

```
#include "pack_type.h"

int c64_pack_scale_x8_opt_3(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   u32 max = 0;
   u32 maxs32 = 0;
   int count = 0;
   int countb = 0;
   u64 *din = (u64 *) in;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 8) {
      u64 x = *din++;
      maxs32 = _maxu4(maxs32, x >> 32);
      max = _maxu4(max, x);
      count += 8;
      if (count == m) {
         max = _maxu4(max, maxs32);
         max = _maxu4(max, max >> 16);
         *result++ = _maxu4(max, max >> 8);
         max = 0;
         maxs32 = 0;
         count = 0;
         countb++;
      }
   }
   return countb;
}
```

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 --speculate_loads=32 -k
--preproc_with_compile --preproc_dependency="c64_pack_scale_opt.pp"
"../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 27: Фидбэк программы, производящей загрузку 8 байт

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../c64_pack_scale_opt.c
;*      Loop source line                : 92
;*      Loop opening brace source line  : 92
;*      Loop closing brace source line  : 106
;*      Loop Unroll Multiple            : 2x
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 6
;*      Unpartitioned Resource Bound    : 6
;*      Partitioned Resource Bound(*)   : 6
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     6*       6*
;*      .S units                     2        3
;*      .D units                     2        2
;*      .M units                     0        0
;*      .X cross paths               1        4
;*      .T address paths             3        1
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          6        4     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             4        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 entirely removed
;*      Collapsed epilog stages       : 1
;*
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 16 bytes
;*      Minimum threshold value       : -mh32
;*
;*      Minimum safe trip count       : 2 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP PROLOG

           CMPEQ   .L2X    A16,B4,B0         ; |97| (P) <0,6>
||         LDDW    .D1T1   *A9++(16),A5:A4   ; |94| (P) <1,0>
```

```
   [ B0]   ADD     .S2     1,B9,B9           ; |104| (P) <0,7>
|| [!B0]   ADD     .S1     8,A16,A16         ; |96| (P) <0,7>
||         MAXU4   .L2X    B5,A4,B7          ; |95| (P) <0,7>
||         MAXU4   .L1     A17,A5,A8         ; |94| (P) <0,7>  ^

           ROTL    .M2     B0,0,B2           ; |97| (P) <0,8> Split a long life
|| [ B0]   MVK     .L1     0x8,A16           ; |103| (P) <0,8>
||         LDDW    .D1T1   *-A9(8),A7:A6     ; |94| (P) <1,2>

   [ B0]   ZERO    .S1     A8                ; |102| (P) <0,9>
|| [ B0]   ZERO    .S2     B7                ; |101| (P) <0,9>  ^
|| [ B0]   MAXU4   .L2X    B7,A8,B17         ; |98| (P) <0,9>  ^
||         CMPEQ   .L1     A16,A3,A1         ; |97| (P) <0,9>

           ROTL    .M1     A1,0,A0           ; |97| (P) <0,10> Split a long life
|| [ A1]   ADD     .D2     1,B16,B16         ; |104| (P) <0,10>
|| [ B2]   SHRU    .S2     B17,16,B7         ; |99| (P) <0,10>
||         MAXU4   .L2X    B7,A6,B5          ; |95| (P) <0,10>  ^
|| [ A1]   MVK     .L1     0x8,A16           ; |103| (P) <0,10>
|| [!A1]   ADD     .S1     8,A16,A16         ; |96| (P) <0,10>

           SUB     .S2     B1,2,B1
|| [ B2]   MAXU4   .L2     B17,B7,B8         ; |99| (P) <0,11>
||         MAXU4   .L1     A8,A7,A17         ; |94| (P) <0,11>

;** --------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_x8_opt_3$8$B:

   [ A0]   ZERO    .D2     B5                ; |101| <0,12>
|| [ B1]   BDEC    .S2     $C$L4,B1          ; |92| <0,12>
|| [ A0]   MAXU4   .L1X    B5,A17,A18        ; |98| <0,12>  ^
|| [ A0]   ZERO    .S1     A17               ; |102| <0,12>  ^
||         CMPEQ   .L2X    A16,B4,B0         ; |97| <1,6>
||         LDDW    .D1T1   *A9++(16),A5:A4   ; |94| <2,0>

   [ B2]   SHRU    .S2     B8,8,B17          ; |100| <0,13>
|| [ A0]   SHRU    .S1     A18,16,A4         ; |99| <0,13>
|| [ B0]   ADD     .D2     1,B9,B9           ; |104| <1,7>
|| [!B0]   ADD     .D1     8,A16,A16         ; |96| <1,7>
||         MAXU4   .L2X    B5,A4,B7          ; |95| <1,7>
||         MAXU4   .L1     A17,A5,A8         ; |94| <1,7>  ^

   [ B2]   MAXU4   .L2     B8,B17,B17        ; |100| <0,14>
|| [ A0]   MAXU4   .L1     A18,A4,A5         ; |99| <0,14>
||         ROTL    .M2     B0,0,B2           ; |97| <1,8> Split a long life
|| [ B0]   MVK     .S1     0x8,A16           ; |103| <1,8>
||         LDDW    .D1T1   *-A9(8),A7:A6     ; |94| <2,2>

   [ B2]   STB     .D2T2   B17,*B6++         ; |100| <0,15>
|| [ A0]   SHRU    .S1     A5,8,A4           ; |100| <0,15>
|| [ B0]   ZERO    .D1     A8                ; |102| <1,9>
|| [ B0]   ZERO    .S2     B7                ; |101| <1,9>  ^
|| [ B0]   MAXU4   .L2X    B7,A8,B17         ; |98| <1,9>  ^
||         CMPEQ   .L1     A16,A3,A1         ; |97| <1,9>
```

```
   [ A0]   MAXU4   .L1     A5,A4,A6          ; |100| <0,16>
||         ROTL    .M1     A1,0,A0           ; |97| <1,10> Split a long life
|| [ A1]   ADD     .D2     1,B16,B16         ; |104| <1,10>
|| [ B2]   SHRU    .S2     B17,16,B7         ; |99| <1,10>
||         MAXU4   .L2X    B7,A6,B5          ; |95| <1,10>  ^
|| [ A1]   MVK     .S1     0x8,A16           ; |103| <1,10>
|| [!A1]   ADD     .D1     8,A16,A16         ; |96| <1,10>

   [ A0]   STB     .D2T1   A6,*B6++          ; |100| <0,17>
|| [ B2]   MAXU4   .L2     B17,B7,B8         ; |99| <1,11>
||         MAXU4   .L1     A8,A7,A17         ; |94| <1,11>

$C$DW$L$_c64_pack_scale_x8_opt_3$8$E:
;** --------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Из фидбэка следует, что за 6 тактов обрабатывается 16 байт, что соответствует удельной производительности 0,375 тактов на байт. Результаты профилирования для массива из 128 элементов и m=16:

Name	Calls	Incl Count Average
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	546.38
c64_pack_scale_x8_opt_3(unsigned char *, int, int, unsigned char *)	100	96.61

Рисунок 12: Сравнение эталона и программы, производящей загрузку 8 байт 1 инструкцией (m = 16; n = 128)

Прирост скорости составляет 565,6%.

4.4. Программа, производящая загрузку 2 байт 1 командой

Для увеличения универсальности конечной программы, которая заключает в себе объединение функций с различными ограничениями по кратности входного массива и количества соседних байт, полезно реализовать программу загружающую 2 байта за 1 инструкцию. Данная программа является ускоренной версией программы из пункта 3.2. Для обработки входного массива модно прописать диспетчер, который с помощью условия будет вызывать функцию, соответствующую кратности входного массива, с помощью чего можно добиться максимальной скорости для каждой длины входного массива и количества соседних байт. Массив в данной программе должен быть кратен 2m.

Листинг 28: Программа, производящая загрузку 2 байт

```
#include "pack_type.h"

int c64_pack_scale_x2_opt_4(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   u32 max = 0;
   u32 max1 = 0;
   int count = 0;
   int countb = 0;
   int m1 = m >> 1;
   u16 *din = (u16 *) in;
#pragma UNROLL(2)
```

```
#pragma MUST_ITERATE(2, ,2)
   for (k = 0; k < n; k += 4) {
      u32 y = *(din + m1);
      u32 x = *din++;
      max = _maxu4(max, y);
      max1 = _maxu4(max1, x);
      count += 2;
      if (count == m) {
         *result++ = _maxu4(max1, max1 >> 8);
         *result++ = _maxu4(max, max >> 8);
         count = 0;
         countb += 2;
         din += m1;
         max = 0;
         max1 = 0;
      }
   }
   return countb;
}
```

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="c64_pack_scale_opt.pp"  "../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 29: Фидбэк программы, производящей загрузку 2 байт

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c64_pack_scale_opt.c
;*      Loop source line                 : 120
;*      Loop opening brace source line   : 120
;*      Loop closing brace source line   : 135
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 6
;*      Unpartitioned Resource Bound     : 7
;*      Partitioned Resource Bound(*)    : 7
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     4        6
;*      .S units                     3        2
;*      .D units                     4        4
;*      .M units                     0        0
;*      .X cross paths               0        2
;*      .T address paths             6        2
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          9        8     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             4        4
```

```
;*      Bound(.L .S .D .LS .LSD)     7*       7*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 7  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      Minimum safe trip count       : 3 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L23:    ; PIPED LOOP PROLOG

   [!B2]   ADD     .L1     2,A5,A5           ; |122| (P) <0,2>
||         LDHU    .D1T1   *A8++,A3          ; |121| (P) <0,3>
|| [ B2]   MVK     .L2     0x2,B5            ; |129| (P) <0,3>

   [ B2]   ADD     .S2     2,B9,B9           ; |130| (P) <0,2>
|| [ B2]   MV      .L1     A8,A5             ; |131| (P) <0,4>
||         CMPEQ   .L2     B5,B8,B2          ; |126| (P) <0,4>

           MV      .L2X    A3,B6
||         MVD     .M2     B2,B0             ; |126| (P) <0,5> Split a long life
|| [ B1]   ADD     .L1     A7,A8,A8          ; (P) <0,5>
||         LDHU    .D1T1   *A5,A3            ; |122| (P) <0,5>
|| [!B2]   ADD     .S2     2,B5,B5           ; |125| (P) <0,5>

   [ B2]   ADD     .L1     2,A8,A5           ; |131| (P) <0,6>
|| [ B2]   MVK     .S2     0x2,B5            ; |129| (P) <0,6>
||         MAXU4   .L2X    B6,A9,B4          ; |124| (P) <0,6>  ^
||         LDHU    .D1T1   *A8,A16           ; |121| (P) <0,6>

   [!B2]   ADD     .L1     2,A5,A5           ; |122| (P) <0,7>
|| [ B1]   SHRU    .S2     B4,8,B17          ; |127| (P) <0,7>  ^
||         CMPEQ   .L2     B5,B8,B2          ; |126| (P) <1,0>

   [ B1]   ZERO    .S2     B4                ; |133| (P) <0,8>  ^
|| [ B1]   MAXU4   .L2     B4,B17,B17        ; |127| (P) <0,8>  ^
||         MAXU4   .L1     A6,A3,A9          ; |123| (P) <0,8>  ^
||         MVD     .M2     B2,B1             ; |126| (P) <1,1> Split a long life
|| [!B2]   ADD     .D2     2,B5,B5           ; |125| (P) <1,1>
||         LDHU    .D1T1   *A5,A3            ; |122| (P) <1,1>

   [ B0]   ADD     .L2     2,B16,B16         ; |130| (P) <0,9>
|| [ B1]   STB     .D2T2   B17,*B7++         ; |127| (P) <0,9>
|| [ B1]   SHRU    .S1     A9,8,A4           ; |128| (P) <0,9>  ^
|| [ B2]   ADD     .S2     2,B9,B9           ; |130| (P) <1,2>
|| [!B2]   ADD     .L1     2,A5,A5           ; |122| (P) <1,2>
||         ADD     .D1     A7,A5,A8          ; |121| (P) <1,2>

;** ---------------------------------------------------------------------------*
$C$L24:    ; PIPED LOOP KERNEL
```

```
$C$DW$L$_c64_pack_scale_x2_opt_4$6$B:

   [ B1]   ZERO    .S1     A9              ; |132| <0,10>  ^
||         MAXU4   .L2X    B4,A3,B6        ; |124| <0,10>  ^
|| [ B1]   MAXU4   .L1     A9,A4,A3        ; |128| <0,10>  ^
||         LDHU    .D1T1   *A8++,A3        ; |121| <1,3>
|| [ B2]   MVK     .S2     0x2,B5          ; |129| <1,3>

   [ B1]   STB     .D2T1   A3,*B7++        ; |128| <0,11>
|| [ A0]   B       .S1     $C$L24          ; |120| <0,11>
|| [ B0]   SHRU    .S2     B6,8,B4         ; |127| <0,11>  ^
||         MAXU4   .L1     A9,A16,A6       ; |123| <0,11>  ^
|| [ B2]   MV      .D1     A8,A5           ; |131| <1,4>
||         CMPEQ   .L2     B5,B8,B2        ; |126| <1,4>

   [ B0]   ZERO    .S2     B6              ; |133| <0,12>  ^
|| [ B0]   MAXU4   .L2     B6,B4,B4        ; |127| <0,12>  ^
|| [ B0]   SHRU    .S1     A6,8,A9         ; |128| <0,12>  ^
||         MVD     .M2     B2,B0           ; |126| <1,5> Split a long life
|| [ B1]   ADD     .L1     A7,A8,A8        ; <1,5>
||         LDHU    .D1T1   *A5,A3          ; |122| <1,5>
|| [!B2]   ADD     .D2     2,B5,B5         ; |125| <1,5>

   [ B0]   STB     .D2T2   B4,*B7++        ; |127| <0,13>
|| [ B0]   MAXU4   .L1     A6,A9,A3        ; |128| <0,13>  ^
|| [ B2]   ADD     .S1     2,A8,A5         ; |131| <1,6>
|| [ B2]   MVK     .S2     0x2,B5          ; |129| <1,6>
||         MAXU4   .L2X    B6,A3,B4        ; |124| <1,6>  ^
||         LDHU    .D1T1   *A8,A16         ; |121| <1,6>

   [ B0]   ZERO    .S1     A6              ; |132| <0,14>  ^
|| [ B0]   STB     .D2T1   A3,*B7++        ; |128| <0,14>
|| [!B2]   ADD     .L1     2,A5,A5         ; |122| <1,7>
|| [ B1]   SHRU    .S2     B4,8,B17        ; |127| <1,7>  ^
||         CMPEQ   .L2     B5,B8,B2        ; |126| <2,0>

   [ A0]   SUB     .S1     A0,2,A0         ; |120| <1,8>
|| [ B1]   ZERO    .D2     B4              ; |133| <1,8>  ^
|| [ B1]   MAXU4   .L2     B4,B17,B17      ; |127| <1,8>  ^
||         MAXU4   .L1     A6,A3,A9        ; |123| <1,8>  ^
||         MVD     .M2     B2,B1           ; |126| <2,1> Split a long life
|| [!B2]   ADD     .S2     2,B5,B5         ; |125| <2,1>
||         LDHU    .D1T1   *A5,A3          ; |122| <2,1>

   [ B0]   ADD     .S2     2,B16,B16       ; |130| <1,9>
|| [ B1]   STB     .D2T2   B17,*B7++       ; |127| <1,9>
|| [ B1]   SHRU    .S1     A9,8,A4         ; |128| <1,9>  ^
|| [ B2]   ADD     .L2     2,B9,B9         ; |130| <2,2>
|| [!B2]   ADD     .L1     2,A5,A5         ; |122| <2,2>
||         ADD     .D1     A7,A5,A8        ; |121| <2,2>

$C$DW$L$_c64_pack_scale_x2_opt_4$6$E:
;** --------------------------------------------------------------------------*
$C$L25:    ; PIPED LOOP EPILOG

   [ B1]   ZERO    .S1     A9              ; |132| (E) <1,10>  ^
||         MAXU4   .L2X    B4,A3,B6        ; |124| (E) <1,10>  ^
```

```
||	[ B1]	MAXU4	.L1	A9,A4,A3	; |128| (E) <1,10>  ^
||		LDHU	.D1T1	*A8++,A3	; |121| (E) <2,3>
||	[ B2]	MVK	.S2	0x2,B5	; |129| (E) <2,3>

	[ B1]	STB	.D2T1	A3,*B7++	; |128| (E) <1,11>
||	[ B0]	SHRU	.S2	B6,8,B4	; |127| (E) <1,11>  ^
||		MAXU4	.L1	A9,A16,A6	; |123| (E) <1,11>  ^
||	[ B2]	MV	.S1	A8,A5	; |131| (E) <2,4>
||		CMPEQ	.L2	B5,B8,B2	; |126| (E) <2,4>

	[ B0]	ZERO	.S2	B6	; |133| (E) <1,12>  ^
||	[ B0]	MAXU4	.L2	B6,B4,B4	; |127| (E) <1,12>  ^
||	[ B0]	SHRU	.S1	A6,8,A9	; |128| (E) <1,12>  ^
||		MVD	.M2	B2,B0	; |126| (E) <2,5> Split a long life
||	[ B1]	ADD	.L1	A7,A8,A8	; (E) <2,5>
||		LDHU	.D1T1	*A5,A3	; |122| (E) <2,5>
||	[!B2]	ADD	.D2	2,B5,B5	; |125| (E) <2,5>

	[ B0]	STB	.D2T2	B4,*B7++	; |127| (E) <1,13>
||	[ B0]	MAXU4	.L1	A6,A9,A3	; |128| (E) <1,13>  ^
||	[ B2]	ADD	.S1	2,A8,A5	; |131| (E) <2,6>
||	[ B2]	MVK	.S2	0x2,B5	; |129| (E) <2,6>
||		MAXU4	.L2X	B6,A3,B4	; |124| (E) <2,6>  ^
||		LDHU	.D1T1	*A8,A4	; |121| (E) <2,6>

	[ B0]	ZERO	.L1	A6	; |132| (E) <1,14>  ^
||	[ B0]	STB	.D2T1	A3,*B7++	; |128| (E) <1,14>
||	[!B2]	ADD	.S1	2,A5,A5	; |122| (E) <2,7>
||	[ B1]	SHRU	.S2	B4,8,B17	; |127| (E) <2,7>  ^

	[ B1]	ZERO	.S2	B4	; |133| (E) <2,8>  ^
||	[ B1]	MAXU4	.L2	B4,B17,B17	; |127| (E) <2,8>  ^
||		MAXU4	.L1	A6,A3,A9	; |123| (E) <2,8>  ^

	[ B0]	ADD	.L2	2,B16,B16	; |130| (E) <2,9>
||	[ B1]	STB	.D2T2	B17,*B7++	; |127| (E) <2,9>
||	[ B1]	SHRU	.S1	A9,8,A4	; |128| (E) <2,9>  ^

		MVC	.S2	B18,CSR	; interrupts on
||	[ B1]	ZERO	.S1	A9	; |132| (E) <2,10>  ^
||	[ B1]	MAXU4	.L1	A9,A4,A3	; |128| (E) <2,10>  ^
||		MAXU4	.L2X	B4,A3,B6	; |124| (E) <2,10>  ^

;** --------------------------------------------------------------------------*
```

Так как цикл развернулся, для максимальной скорости адрес входного массива должен быть кратен 8. Полученная программа обрабатывает 8 байт за 7 тактов. Результаты профилирования для массива из 128 элементов и m=16:

Name	Calls	Incl Count Average
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	546.38
c64_pack_scale_x2_opt_4(unsigned char *, int, int, unsigned char *)	100	135.45

Рисунок 13: Сравнение эталона и программы, производящей загрузку 2 байт 1 инструкцией (n = 128, m = 16)

Прирост скорости составляет 403,4%.

4.5. Программа, производящая копирование памяти

Данная программа является адаптированной версией программы из пункта 3.4, осуществляющей копирование памяти, для ядра C64. В программе произведено ручное разворачивание цикла для балансировки ресурсов по сторонам конвейера.

Листинг 30: Программа, производящая копирование памяти

```
#include "pack_type.h"

int c64_pack_scale_1_opt_5(const u8 *in, int n, u8*restrict result)
{
   _nassert(((int)(in) & 0x7) == 0);
   int i;
   u64 *din = (u64 *) in;
   u64 *dres = (u64 *) result;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 16) {
      *dres++ = *din++;
      *dres++ = *din++;
   }
   return n;
}
```

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 --speculate_loads=32 -k
--preproc_with_compile --preproc_dependency="c64_pack_scale_opt.pp"
"../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 31: Фидбэк программы, производящей копирование памяти

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c64_pack_scale_opt.c
;*      Loop source line                 : 145
;*      Loop opening brace source line   : 145
;*      Loop closing brace source line   : 148
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 2
;*      Partitioned Resource Bound(*)    : 2
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                     1        0
;*      .D units                     2*       2*
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             2*       2*
;*      Long read paths              0        0
```

```
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        0
;*      Bound(.L .S .D .LS .LSD)     1        1
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 2
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 32 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L47:    ; PIPED LOOP PROLOG

   [ A0]   BDEC    .S1     $C$L48,A0         ; |145| (P) <0,0>
||         LDDW    .D1T1   *A3++(16),A7:A6   ; |146| (P) <0,0>
||         LDDW    .D2T2   *++B6(16),B5:B4   ; |147| (P) <0,0>

           NOP             1

   [ A0]   BDEC    .S1     $C$L48,A0         ; |145| (P) <1,0>
||         LDDW    .D1T1   *A3++(16),A7:A6   ; |146| (P) <1,0>
||         LDDW    .D2T2   *++B6(16),B5:B4   ; |147| (P) <1,0>

           SUB     .L2X    A4,8,B7
;** ---------------------------------------------------------------------------*
$C$L48:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_1_opt_5$3$B:

   [ A0]   BDEC    .S1     $C$L48,A0         ; |145| <2,0>
||         LDDW    .D1T1   *A3++(16),A7:A6   ; |146| <2,0>
||         LDDW    .D2T2   *++B6(16),B5:B4   ; |147| <2,0>

           STDW    .D1T1   A7:A6,*A4++(16)   ; |146| <0,5>
||         STDW    .D2T2   B5:B4,*++B7(16)   ; |147| <0,5>

$C$DW$L$_c64_pack_scale_1_opt_5$3$E:
;** ---------------------------------------------------------------------------*
$C$L49:    ; PIPED LOOP EPILOG
;** ---------------------------------------------------------------------------*
```

За 2 такта программа копирует 16 байт, что соответствует удельной производительности 0,125 такта на элемент. При профилировании программы при m=1 и массиве из 128 элементов получаем:

Name	Calls	Incl Count Average
c64_pack_scale_1_opt_5(unsigned char *, int, unsigned char *)	100	32.13
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	4498.34

Рисунок 14: Сравнение эталона и программы, производящей копирование памяти (n = 128, m = 1)

Прирост производительности составляет 14000,4%.

4.6. Программа, производящая упаковку по паре соседних байт

Для дополнительного увеличения производительности можно реализовать частный случай программы для m=2, который не имеет вложенного цикла, из-за чего оверхэд цикла выполняется 1 раз за всё выполнение программы. Прагма UNROLL использована для балансировки вычислительных ресурсов по сторонам конвейера. Для максимальной скорости адрес массива должен быть кратен 16.

Листинг 32: Программа, производящая упаковку по паре соседних байт

```
#include "pack_type.h"

int c64_pack_scale_2_opt_6(const u8 *in, int n, u8*restrict result)
{
   int i;
   int count = 0;
   u64 *din = (u64 *) in;
   u32 *dres = (u32 *) result;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 8) {
      u64 x = *din++;
      u32 l = _packl4(x >> 32, x);
      u32 h = _packh4(x >> 32, x);
      u32 max = _maxu4(h, l);
      *dres++ = max;
      count += 4;
   }
   return count;
}
```

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 --speculate_loads=48 -k
--preproc_with_compile --preproc_dependency="c64_pack_scale_opt.pp"
"../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 33: Фидбэк программы, производящей упаковку по паре байт

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
```

```
;*
;*      Loop found in file               : ../c64_pack_scale_opt.c
;*      Loop source line                 : 159
;*      Loop opening brace source line   : 159
;*      Loop closing brace source line   : 166
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 3
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3*       3*
;*      .S units                     0        1
;*      .D units                     3*       1
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             2        2
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             2        2
;*      Bound(.L .S .D .LS .LSD)     2        2
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 3  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 3
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 2
;*
;*      Minimum required memory pad   : 48 bytes
;*
;*      Minimum safe trip count       : 1 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L38:    ; PIPED LOOP PROLOG

           SUB     .L1     A3,8,A3
|| [ B0]   BDEC    .S2     $C$L39,B0         ; |159| (P) <0,4>

           NOP             1

           MVK     .L1     0x2,A0            ; init prolog collapse predicate
||         AND     .S1X    -8,B4,A8          ; |165|
||         ADD     .L2X    4,A3,B5
||         LDDW    .D1T1   *A7++(16),A5:A4   ; |164| (P) <0,0>

;** ---------------------------------------------------------------------------*
$C$L39:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_2_opt_6$4$B:

           MAXU4   .L1     A6,A4,A6          ; |164| <0,7>
||         PACKH4  .L2     B7,B6,B6          ; |164| <0,7>
|| [ B0]   BDEC    .S2     $C$L39,B0         ; |159| <1,4>
```

```
||         LDDW    .D1T2   *-A7(8),B7:B6     ; |164| <2,1>

   [!A0]   STW     .D1T1   A6,*++A3(8)       ; |164| <0,8>
||         MAXU4   .L2     B6,B4,B4          ; |164| <0,8>
||         PACKH4  .L1     A5,A4,A6          ; |164| <1,5>

   [ A0]   SUB     .S1     A0,1,A0           ; <0,9>
|| [!A0]   STW     .D2T2   B4,*++B5(8)       ; |164| <0,9>
||         PACKL4  .L1     A5,A4,A4          ; |164| <1,6>
||         PACKL4  .L2     B7,B6,B4          ; |164| <1,6>
||         LDDW    .D1T1   *A7++(16),A5:A4   ; |164| <3,0>

$C$DW$L$_c64_pack_scale_2_opt_6$4$E:
;** --------------------------------------------------------------------------*
$C$L40:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Результаты профилирования для массива из 128 элементов и m=2:

Name	Calls	Incl Count Average
c64_pack_scale_2_opt_6(unsigned char *, int, unsigned char *)	100	73.24
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	2834.30

Рисунок 15: Сравнение эталона и программы, производящей уплотнение по паре байт

Прирост скорости составляет 3869,88%. Из фидбэка следует, за 3 такта программа обрабатывает 16 байт.

4.7. Программа, производящая упаковку по 3 соседних байта

Реализация частного случая упаковки по 3 байта призвана ускорить расчёт при m=3 из-за отсутствия вложенного цикла. В данной программе подразумевается, что адрес входного массива кратен 12.

Листинг 34: Программа, производящая упаковку по три соседних байта

```
#include "pack_type.h"

nt c64_pack_scale_3_opt_7(const u8 *in, int n, u8*restrict result)
{
   int i;
   int count = 0;
   u32 m8 = 256;
   u32 *din = (u32 *) in;
   u32 *dres = (u32 *) result;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 12) {
      u32 x = *din++;
      u32 y = *din++;
      u32 z = *din++;
      u8 x0 = x;
      u8 x1 = x >> 8;
      u8 x2 = x >> 16;
      u8 y2 = y >> 16;
      u8 y3 = y >> 24;
```

```
        u8 z0 = z;
        u32 a = _mpy(x >> 24, m8) + x0;
        u32 b = _mpy(y, m8) + x1;
        u32 c = _mpy(y >> 8, m8) + x2;
        u32 d = _mpy(z >> 8, m8) + y2;
        u32 e = _mpy(z >> 16, m8) + y3;
        u32 f = _mpy(z >> 24, m8) + z0;
        a = _pack2(d, a);
        b = _pack2(e, b);
        c = _pack2(f, c);
        u32 max = _maxu4(a, b);
        max = _maxu4(max, c);
        *dres++ = max;
        count += 4;
    }
    return count;
}
```

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 --speculate_loads=36 -k
--preproc_with_compile --preproc_dependency="c64_pack_scale_opt.pp"
"../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 35: Фидбэк программы, производящей упаковку по три байта

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c64_pack_scale_opt.c
;*      Loop source line                 : 177
;*      Loop opening brace source line   : 177
;*      Loop closing brace source line   : 200
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 3
;*      Unpartitioned Resource Bound     : 5
;*      Partitioned Resource Bound(*)    : 5
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     2        1
;*      .S units                     5*       5*
;*      .D units                     2        1
;*      .M units                     3        3
;*      .X cross paths               3        5*
;*      .T address paths             2        2
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           2        1     (.L or .S unit)
;*      Addition ops (.LSD)          4        5     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             5*       4
;*      Bound(.L .S .D .LS .LSD)     5*       5*
;*
```

```
;*      Searching for software pipeline schedule at ...
;*         ii = 5  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 3
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 2
;*
;*      Minimum required memory pad   : 36 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L35:    ; PIPED LOOP PROLOG

           MV      .L2X    A6,B17
||         ZERO    .S2     B9
||         LDNDW   .D1T1   *A18++(12),A21:A20 ; |178| (P) <0,0>

           ZERO    .L2     B0                ; init loop condition
||         ZERO    .D2     B9                ; |172|
||         SET     .S2     B9,0xf,0xf,B1     ; init prolog collapse predicate

           ADD     .L2     12,B16,B9
||         SUB     .D2     B9,8,B16          ; undo prolog elim. side-effects
||         LDW     .D1T2   *-A18(4),B8       ; |198| (P) <0,2>
||         B       .S2     $C$L36            ; |177| (P) <0,12>

;** ----------------------------------------------------------------------------*
$C$L36:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_3_opt_7$3$B:

           ADD     .D1X    A6,B18,A16        ; |198| <0,13>
||         PACK2   .L1     A8,A7,A8          ; |198| <0,13>
||         MPY     .M2X    B7,A17,B4         ; |198| <1,8>
||         SHRU    .S1     A21,24,A6         ; |198| <1,8>
||         SHRU    .S2     B8,8,B6           ; |198| <1,8>

           ADD     .L2     4,B16,B16         ; |199| <0,14>
||         PACK2   .L1     A16,A3,A4         ; |198| <0,14>
||         SUB     .D2     B9,12,B9          ; |177| <1,9>
||         ADD     .D1     A4,A9,A3          ; |198| <1,9>
||         EXTU    .S2     B4,8,24,B7        ; |198| <1,9>
||         EXTU    .S1     A21,8,24,A9       ; |198| <1,9>
||         MPYHL   .M2X    B6,A17,B7         ; |198| <1,9>

           MAXU4   .L1     A8,A4,A4          ; |198| <0,15>
||         EXTU    .S2     B8,24,24,B8       ; |198| <1,10>
||         ADD     .L2     B7,B4,B6          ; |198| <1,10>
||         MPY     .M1X    B6,A17,A8         ; |198| <1,10>
||         SHRU    .S1     A20,8,A16         ; |198| <2,5>
||         MV      .D2X    A20,B4            ; |178| <2,5> Define a twin register
||         LDNDW   .D1T1   *A18++(12),A21:A20 ; |178| <3,0>

   [ B1]   MPYSU   .M2     2,B1,B1           ; <0,16>
||         MAXU4   .L1X    A4,B5,A5          ; |198| <0,16>
|| [!B0]   CMPLTU  .L2     B9,12,B0          ; |177| <1,11>
||         ADD     .D1     A5,A7,A7          ; |198| <1,11>
```

```
||          ADD     .D2     B8,B7,B5          ; |198| <1,11>
||          EXTU    .S1     A20,16,24,A4      ; |198| <2,6>
||          MPY     .M1     A21,A17,A9        ; |198| <2,6>
||          SHRU    .S2X    A21,8,B7          ; |198| <2,6>

   [!B1]    STW     .D2T1   A5,*B17++         ; |198| <0,17>
||          PACK2   .L2     B5,B6,B5          ; |198| <1,12>
||          ADD     .L1     A9,A8,A8          ; |198| <1,12>
|| [!B0]    B       .S2     $C$L36            ; |177| <1,12>
||          EXTU    .S1     A20,24,24,A5      ; |198| <2,7>
||          MPYHL   .M1     A16,A17,A7        ; |198| <2,7>
||          MPYHL   .M2X    B8,A17,B18        ; |198| <2,7>
||          LDW     .D1T2   *-A18(4),B8       ; |198| <3,2>

$C$DW$L$_c64_pack_scale_3_opt_7$3$E:
;** --------------------------------------------------------------------------*
$C$L37:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Из фидбэка следует, что за 5 тактов обрабатывается 12 элементов, что соответствует удельной производительности 0,42 такта на элемент. При профилировании программы с m=3 и массивом из 120 элементов получаем:

Name	Calls	Incl Count Average
c64_pack_scale_3_opt_7(unsigned char *, int, unsigned char *)	100	74.75
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	1617.35

Рисунок 16: Сравнение эталона и программы, производящей уплотнение по три байта (n = 120, m = 3)

Прирост производительности составляет 2163,7%.

4.8. Программа, производящая упаковку, используя 4 соседних байта

Данная программа реализуется для увеличения скорости для обработки массива с m, равным 4. Прагма UNROLL применена для балансировки ресурсов по сторонам конвейера, из-за чего на адрес входного массива накладывается ограничение в виде кратности 16.

Листинг 36: Программа, производящая упаковку по 4 байта

```
#include "pack_type.h"

int c64_pack_scale_4_opt_8(const u8 *in, int n, u8*restrict result)
{
   int i;
   u32 max;
   int count = 0;
   u64 *din = (u64 *) in;
   u16 *dres = (u16 *) result;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 8) {
      u64 x = *din++;
```

```
      u32 l = _packl4(x >> 32, x);
      u32 h = _packh4(x >> 32, x);
      max = _maxu4(l, h);
      max = _maxu4(max >> 8, max);
      *dres++ = _packl4(max, max);
      count += 2;
   }
   return count;
}
```

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 --speculate_loads=32 -k
--preproc_with_compile --preproc_dependency="c64_pack_scale_opt.pp"
"../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 37: Фидбэк программы, производящей упаковку по 4 байта

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../c64_pack_scale_opt.c
;*      Loop source line                 : 212
;*      Loop opening brace source line   : 212
;*      Loop closing brace source line   : 220
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 5
;*      Partitioned Resource Bound(*)    : 5
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     5*       5*
;*      .S units                     1        2
;*      .D units                     4        0
;*      .M units                     0        0
;*      .X cross paths               0        0
;*      .T address paths             2        2
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)          0        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             3        4
;*      Bound(.L .S .D .LS .LSD)     4        3
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 5  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 2
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
```

```
;*
;*      Minimum required memory pad    : 32 bytes
;*
;*      Minimum safe trip count        : 1 (after unrolling)
;*--------------------------------------------------------------------------*
$C$L30:    ; PIPED LOOP PROLOG

           SUB     .L1     A9,4,A9
||         MVK     .S1     0x1,A0            ; init prolog collapse predicate
||         AND     .D1X    -4,B4,A16         ; |219|
|| [ B0]   BDEC    .S2     $C$L31,B0         ; |212| (P) <0,9>

;** --------------------------------------------------------------------------*
$C$L31:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_4_opt_8$4$B:

           MAXU4   .L2     B4,B5,B8          ; |218| <0,10>
||         SHRU    .S1     A5,8,A3           ; |218| <0,10>
||         PACKH4  .L1     A7,A6,A4          ; |218| <1,5>
||         LDDW    .D1T1   *A8++(16),A7:A6   ; |218| <2,0>

           SHRU    .S2     B8,8,B4           ; |218| <0,11>
||         MAXU4   .L1     A3,A5,A3          ; |218| <0,11>
||         PACKH4  .L2     B7,B6,B5          ; |218| <1,6>
||         LDDW    .D1T2   *-A8(8),B7:B6     ; |218| <2,1>

           MAXU4   .L2     B4,B8,B4          ; |218| <0,12>
||         PACKL4  .L1     A3,A3,A3          ; |218| <0,12>

           PACKL4  .L2     B4,B4,B4          ; |218| <0,13>
|| [!A0]   STH     .D1T1   A3,*++A9(4)       ; |218| <0,13>
||         PACKL4  .L1     A7,A6,A3          ; |218| <1,8>

   [ A0]   SUB     .S1     A0,1,A0           ; <0,14>
|| [!A0]   STH     .D1T2   B4,*+A9(2)        ; |218| <0,14>
|| [ B0]   BDEC    .S2     $C$L31,B0         ; |212| <1,9>
||         PACKL4  .L2     B7,B6,B4          ; |218| <1,9>
||         MAXU4   .L1     A3,A4,A5          ; |218| <1,9>

$C$DW$L$_c64_pack_scale_4_opt_8$4$E:
;** --------------------------------------------------------------------------*
$C$L32:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Name	Calls	Incl Count Average
c64_pack_scale_4_opt_8(unsigned char *, int, unsigned char *)	100	81.31
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	1362.32

Рисунок 17: Сравнение эталона и программы, производящей уплотнение по 4 байта (n = 128, m = 4)

Результаты профилирования для массива из 128 элементов и m=4 представлены на рисунке 17. Прирост скорости составляет 1675,5%. За 5 тактов обрабатывается 16 элементов входного массива.

4.9. Программа для нечётного количества соседних байт

Для увеличения скорости расчёта при нечётных m создадим программу, не имеющую вложенного цикла. Адрес входного массива в данной программе должен быть кратен 2m.

Листинг 38: Программа для нечётных m

```
#include "pack_type.h"

int c64_pack_scale_x1_opt_9(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   u32 max = 0;
   u32 max1 = 0;
   int count = 0;
   int countb = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 2) {
      u32 y = *(in + m);
      u32 x = *in++;
      max = _maxu4(max, x);
      max1 = _maxu4(max1, y);
      count++;
      if (count == m) {
         *result++ = max;
         *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb += 2;
         in += m;
      }
   }
   return countb;
}
```

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="c64_pack_scale_opt.pp"  "../c64_pack_scale_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 39: Фидбэк программы для нечётных m

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../c64_pack_scale_opt.c
;*      Loop source line                : 200
;*      Loop opening brace source line  : 200
;*      Loop closing brace source line  : 215
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 3
```

```
;*      Unpartitioned Resource Bound     : 3
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                  A-side   B-side
;*      .L units                      2        1
;*      .S units                      0        1
;*      .D units                      3*       1
;*      .M units                      0        0
;*      .X cross paths                1        1
;*      .T address paths              2        2
;*      Long read paths               0        0
;*      Long write paths              0        0
;*      Logical  ops (.LS)            0        0     (.L or .S unit)
;*      Addition ops (.LSD)           3        4     (.L or .S or .D unit)
;*      Bound(.L .S .LS)              1        1
;*      Bound(.L .S .D .LS .LSD)      3*       3*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 3  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Epilog not removed
;*      Collapsed epilog stages       : 0
;*
;*      Prolog not removed
;*      Collapsed prolog stages       : 0
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*      Minimum safe trip count       : 3
;*----------------------------------------------------------------------------*
$C$L20:    ; PIPED LOOP PROLOG

   [ A1]   ADD     .L2X    A5,B4,B4          ; |213| (P) <0,3>
|| [ B0]   BDEC    .S2     $C$L21,B0         ; |200| (P) <0,3>
||         LDBU    .D1T1   *A6++,A3          ; |202| (P) <1,0>

           LDBU    .D2T2   *B4++,B7          ; |201| (P) <1,1>
||         CMPEQ   .L1X    B6,A5,A1          ; |206| (P) <1,1>  ^

           MVD     .M1     A1,A0             ; |206| (P) <1,2> Split a long life
|| [!A1]   ADD     .L2     1,B6,B6           ; |205| (P) <1,2>  ^
|| [ A1]   ADD     .L1     A5,A6,A6          ; |213| (P) <1,2>
|| [ A1]   MVK     .S2     0x1,B6            ; |211| (P) <1,2>  ^

;** ----------------------------------------------------------------------------*
$C$L21:    ; PIPED LOOP KERNEL
$C$DW$L$_c64_pack_scale_x1_opt_9$7$B:

           MAXU4   .L2     B5,B7,B5          ; |204| <0,6>
||         MAXU4   .L1     A8,A3,A8          ; |203| <0,6>
|| [ A1]   ADD     .D2X    A5,B4,B4          ; |213| <1,3>
|| [ B0]   BDEC    .S2     $C$L21,B0         ; |200| <1,3>
||         LDBU    .D1T1   *A6++,A3          ; |202| <2,0>

   [ A0]   ZERO    .S1     A8                ; |209| <0,7>
|| [ A0]   STB     .D1T1   A8,*A7++          ; |207| <0,7>
```

```
||         LDBU    .D2T2   *B4++,B7          ; |201| <2,1>
||         CMPEQ   .L1X    B6,A5,A1          ; |206| <2,1>  ^

   [ A0]   ADD     .L1     2,A4,A4           ; |212| <0,8>
|| [ A0]   ZERO    .D2     B5                ; |210| <0,8>
|| [ A0]   STB     .D1T2   B5,*A7++          ; |208| <0,8>
||         MVD     .M1     A1,A0             ; |206| <2,2> Split a long life
|| [!A1]   ADD     .L2     1,B6,B6           ; |205| <2,2>  ^
|| [ A1]   ADD     .S1     A5,A6,A6          ; |213| <2,2>
|| [ A1]   MVK     .S2     0x1,B6            ; |211| <2,2>  ^

$C$DW$L$_c64_pack_scale_x1_opt_9$7$E:
;** --------------------------------------------------------------------------*
$C$L22:    ; PIPED LOOP EPILOG

           MAXU4   .L2     B5,B7,B5          ; |204| (E) <1,6>
||         MAXU4   .L1     A8,A3,A8          ; |203| (E) <1,6>
|| [ A1]   ADD     .S2X    A5,B4,B4          ; |213| (E) <2,3>

   [ A0]   ZERO    .L1     A8                ; |209| (E) <1,7>
|| [ A0]   STB     .D1T1   A8,*A7++          ; |207| (E) <1,7>

           MVC     .S2     B8,CSR            ; interrupts on
||         MAXU4   .L1     A8,A3,A8          ; |203| (E) <2,6>
|| [ A0]   ADD     .S1     2,A4,A4           ; |212| (E) <1,8>
|| [ A0]   ZERO    .L2     B5                ; |210| (E) <1,8>
|| [ A0]   STB     .D1T2   B5,*A7++          ; |208| (E) <1,8>

;** --------------------------------------------------------------------------*
```

Из фидбэка следует, что за 3 такта обрабатывается 2 элемента. Профилирование для массива из 120 элементов и m=5 даёт следующие результаты:

Name	Calls	Incl Count Average
c64_pack_scale_no_opt(unsigned char *, int, int, unsigned char *)	100	1073.34
c64_pack_scale_x1_opt_9(unsigned char *, int, int, unsigned char *)	100	209.63

Рисунок 18: Сравнение эталона и программы для нечётных m (m = 5, n = 120)

Прирост производительности составляет 512%.

5. Заключение

На разработанные алгоритмы накладываются ограничения кратности для входного массива и количества соседних байт от 2 до 16. Для получения единого интерфейса можно сделать диспетчер, из которого будет вызываться определённая программа для максимально быстрой обработки входного массива, исходя из наибольшей кратности входных параметров. Так как в программе не происходит суммирования элементов входного массива, ограничений по длине массива нет.

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.

Основные факторы, ограничивающие скорость выполнения алгоритма:

- блоки .L, которые выполняют сравнение и инструкции упаковки packl4 и packh4.
- оверхэд внутреннего цикла, который всегда присутствует. При этом он многократно повторяется в теле внешнего цикла.

Оптимизация алгоритма свелась к устранению двойного цикла, в котором конвейер разворачивался только для внутреннего цикла. Также удалось использовать встроенные инструкции процессора для распараллеливания расчёта. В итоге удалось равномерно загрузить все основные блоки процессора: .S и .L для ядра C62, .L для ядра C64; добиться обработки 16 входных элементов в каждом третьем такте для ядра C64 и восьми входных элементов в каждом пятом такте для ядра C62.

Сравнительный анализ шагов оптимизации для ядер C62 и C64 представлен в таблице ниже. Каждому значению m соответствуют только те программы, для которых выполняется условие кратности m.

Таблица 1: Сравнение производительности алгоритмов уплотнения РЛС сигнала для C62 и C64

Имя функции	m	Тактов (n=960)	Тактов (n=1920)	Прирост на n=960	Тактов на элемент	Произв-сть (n=960)	Произв-сть на элемент
c62_pack_scale_no_opt	1	**33634,91**	**67235,86**	**33600,95**	**35**	**100,00%**	**100,00%**
c62_pack_scale_x1_opt_1		2900,18	5780,18	2880	3	1159,75%	1166,70%
c62_pack_scale_1_opt_4		259,21	499,21	240	0,25	12975,93%	14000,40%
c64_pack_scale_no_opt		**33634**	**67234**	**33600**	**35**	**100,00%**	**100,00%**
c64_pack_scale_x1_opt_9		1469,56	2909,56	1440	1,5	2288,71%	2333,33%
c64_pack_scale_1_opt_5		136,32	256,32	120	0,13	24672,83%	28000,00%
c62_pack_scale_no_opt	2	**21159,91**	**42280,86**	**21120,95**	**22**	**100,00%**	**100,00%**
c62_pack_scale_x1_opt_1		2900,18	5780,18	2880	3	729,61%	733,37%
c62_pack_scale_x2_opt_2		2421,23	4821,23	2400	2,5	873,93%	880,04%
c62_pack_scale_2_opt_5		742,64	1462,65	720,01	0,75	2849,28%	2933,42%
c64_pack_scale_no_opt		**21156**	**42276**	**21120**	**22**	**100,00%**	**100,00%**
c64_pack_scale_x2_opt_4		863,41	1703,41	840	0,88	2450,28%	2514,29%
c64_pack_scale_2_opt_6		229,17	409,17	180	0,19	9231,57%	11733,33%
c64_pack_scale_x1_opt_9		1469,56	2909,56	1440	1,5	1439,61%	1466,67%
c62_pack_scale_no_opt	3	**12515,94**	**24996,9**	**12480,96**	**13**	**100,00%**	**100,00%**
c62_pack_scale_x1_opt_1		2900,18	5780,18	2880	3	431,56%	433,37%
c62_pack_scale_3_opt_6		728,64	1369,64	641	0,67	1717,71%	1947,11%
c64_pack_scale_no_opt		**12833**	**25633**	**12800**	**13,33**	**100,00%**	**100,00%**
c64_pack_scale_3_opt_7		458,38	858,37	399,99	0,42	2799,64%	3200,08%
c64_pack_scale_x1_opt_9		1469,56	2909,56	1440	1,5	873,25%	888,89%

Продолжение таблицы 1

Имя функции	m	Тактов (n=960)	Тактов (n=1920)	Прирост на n=960	Тактов на элемент	Произв-сть (n=960)	Произв-сть на элемент
c62_pack_scale_no_opt	4	**9875,94**	**19716,9**	**9840,96**	**10,25**	**100,00%**	**100,00%**
c62_pack_scale_x1_opt_1		2900,18	5780,18	2880	3	340,53%	341,70%
c62_pack_scale_x2_opt_2		2421,23	4821,23	2400	2,5	407,89%	410,04%
c62_pack_scale_x4_opt_3		1226,35	2426,35	1200	1,25	805,31%	820,08%
c62_pack_scale_4_opt_7		655,55	1255,55	600	0,63	1506,51%	1640,16%
c64_pack_scale_no_opt		**10118**	**20198**	**10080**	**10,5**	**100,00%**	**100,00%**
c64_pack_scale_x4_opt_1		1226,41	2426,41	1200	1,25	825,01%	840,00%
c64_pack_scale_x4_opt_2		462,53	882,53	420	0,44	2187,53%	2400,00%
c64_pack_scale_x2_opt_4		863,41	1703,4	839,99	0,87	1171,87%	1200,01%
c64_pack_scale_4_opt_8		354,08	654,08	300	0,31	2857,55%	3360,00%
c64_pack_scale_x1_opt_9		1469,56	2909,56	1440	1,5	688,51%	700,00%
c62_pack_scale_no_opt	8	**5915,94**	**11796,9**	**5880,96**	**6,13**	**100,00%**	**100,00%**
c62_pack_scale_x1_opt_1		2900,18	5780,18	2880	3	203,99%	204,20%
c62_pack_scale_x2_opt_2		2421,23	4821,23	2400	2,5	244,34%	245,04%
c62_pack_scale_x4_opt_3		1226,35	2426,35	1200	1,25	482,40%	490,08%
c64_pack_scale_no_opt		**6038**	**12038**	**6000**	**6,25**	**100,00%**	**100,00%**
c64_pack_scale_x4_opt_1		1226,41	2426,41	1200	1,25	492,33%	500,00%
c64_pack_scale_x4_opt_2		462,53	882,53	420	0,44	1305,43%	1428,57%
c64_pack_scale_x8_opt_3		420,41	780,41	360	0,38	1436,22%	1666,67%
c64_pack_scale_x2_opt_4		863,4	1703,4	840	0,88	699,33%	714,29%
c64_pack_scale_x1_opt_9		1469,56	2909,56	1440	1,5	410,87%	416,67%
c62_pack_scale_no_opt	32	**2945,94**	**5856,9**	**2910,96**	**3,03**	**100,00%**	**100,00%**
c62_pack_scale_x1_opt_1		2900,18	5780,18	2880	3	101,58%	101,08%
c62_pack_scale_x2_opt_2		2421,23	4821,23	2400	2,5	121,67%	121,29%
c62_pack_scale_x4_opt_3		1226,35	2426,35	1200	1,25	240,22%	242,58%
c64_pack_scale_no_opt		**2978**	**5918**	**2940**	**3,06**	**100,00%**	**100,00%**
c64_pack_scale_x4_opt_1		1226,41	2426,41	1200	1,25	242,82%	245,00%
c64_pack_scale_x4_opt_2		462,53	882,53	420	0,44	643,85%	700,00%
c64_pack_scale_x8_opt_3		420,47	780,47	360	0,38	708,26%	816,67%
c64_pack_scale_x2_opt_4		863,42	1703,42	840	0,88	344,91%	350,00%
c64_pack_scale_x1_opt_9		1470,55	2910,55	1440	1,5	202,51%	204,17%

Из таблицы следует, что прирост производительности от оптимизации сильнее всего виден для малых m, при этом общая тенденция на падение ускорения наблюдается при росте m. В таблице цветом фона выделены строки с максимальной производительностью. Для ядра C62 — зелёным; для ядра C64 — голубым. По результатам, полученным в таблице, а также из анализа кода, можно составить следующий диспетчер, который выбирает самый оптимальный вариант программы для введённого m:

Листинг 40: Диспетчер вызова функций

```c
#include "pack_type.h"

int c62_pack_scale_no_opt(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x1_opt_1(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x2_opt_2(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x4_opt_3(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_1_opt_4(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_2_opt_5(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_3_opt_6(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_4_opt_7(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_no_opt(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x4_opt_1(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x4_opt_2(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x8_opt_3(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x2_opt_4(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_1_opt_5(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_2_opt_6(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_3_opt_7(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_4_opt_8(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_x1_opt_9(const u8 *in, int n, int m, u8*restrict result);

int c62_pack_scale(const u8 *in, int n, int m, u8* result)
{
   switch (m) {
   case 1:
      return c62_pack_scale_1_opt_4(in, n, result);
   case 2:
      return c62_pack_scale_2_opt_5(in, n, result);
   case 3:
      return c62_pack_scale_3_opt_6(in, n, result);
   case 4:
      return c62_pack_scale_4_opt_7(in, n, result);
   default:
      if ((m % 4) == 0)
         return c62_pack_scale_x4_opt_3(in, n, m, result);
      else if ((m % 2) == 0)
         return c62_pack_scale_x2_opt_2(in, n, m, result);
      else
         return c62_pack_scale_x1_opt_1(in, n, m, result);
   }
}

int c64_pack_scale(u8 *in, int n, int m, u8* result)
{
   switch (m) {
   case 1:
      return c64_pack_scale_1_opt_5(in, n, result);
   case 2:
      return c64_pack_scale_2_opt_6(in, n, result);
   case 3:
      return c64_pack_scale_3_opt_7(in, n, result);
   case 4:
      return c64_pack_scale_4_opt_8(in, n, result);
   default:
      if ((m % 8) == 0)
```

```
            return c64_pack_scale_x8_opt_3(in, n, m, result);
         else if ((m % 4) == 0)
            return c64_pack_scale_x4_opt_2(in, n, m, result);
         else if ((m % 2) == 0)
            return c64_pack_scale_x2_opt_4(in, n, m, result);
         else
            return c64_pack_scale_x1_opt_9(in, n, m, result);
      }
}
```

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 15 раз по сравнению с эталоном для ядра C62 при m = 4 и в 92,3 раза на ядре C64 при m = 2, что соответствует удельному росту производительности в 16,4 раза на один элемент входных данных (длину вектора) для ядра C62 и 117,3 раза для ядра C64. Копирование памяти осуществляется с удельной производительностью 0,25 такта на элемент для ядра C62 и 0,125 такта на элемент для ядра C64, что составляет удельный рост производительности в 133,4 раза для C62 и в 277,8 раз для C64.

5.1.2. Зависит ли ускорение от архитектуры ядра?

Да, за счёт встроенных инструкций, присутствующих на ядре C64 удалось получить дополнительное ускорение, которого не было при использовании ядра C62. Также в ядре C64 ширина шины данных составляет 64 бита, в то время как в ядре C62 — 32 бит, что позволяет читать данные из памяти в 2 раза быстрее, это пригодилось для программы, в которой происходит уплотнение массива при m, кратном 8, так в данной программе происходит загрузка 8 байт, что соответствует 64 битам. Также компилятор составил расписание, в котором присутствует инструкция загрузки 8 байт из памяти, для программы с m, равным 3, для ядра C64. Копирование памяти на ядре C64 также происходит по 64 бита. В целом, увеличенная ширина данных на ядре C64 пригодилась в 3 программах из 9. Можно сказать, что увеличенная ширина данных позволила получить дополнительное ускорение для 6 шкал дальности из 32, то есть, меньше, чем в 20% случаев.

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

Для приведенного алгоритма необходимо наложить следующие ограничения. Так как 1 элемент входного массива занимает в памяти 1 байт, то можно сказать, что для использования инструкции ldb отсутствует условие кратности, для использования инструкции ldh адрес массива должен быть кратен 2, для использования инструкции ldw адрес массива должен быть кратен 4, для использования инструкции lddw адрес массива должен быть кратен 8. Все перечисленные инструкции используются в оптимизированной программе. При организации диспетчера, вызывающего функцию, в которой инструкция

загрузки из памяти данных соответствует кратности количества соседних байт и кратности адреса входного массива, можно добиться максимальной скорости для каждого частного случая количества соседних байт, при этом не будут накладываться ограничения на данный входной параметр. Для максимальной скорости требуется массив с числом элементов, кратным 8, и количество соседних байт, которые необходимо объединить, кратное 8. Это следует из анализа оптимизированного кода.

5.2. Дополнительные вопросы по заданию

5.2.1. Зависит ли оптимальный алгоритм от параметра m? Для каких значений m можно обойтись общей реализацией упаковки? С чем это может быть связано?

Да, так как примитивность алгоритма поиска максимума приводит к тому, что приходится либо разворачивать цикл, либо использовать инструкции, выполняющие параллельный расчёт нескольких максимумов, что соответствует введению условия кратности m двум, четырём или восьми. Также наблюдается снижение прироста производительности с увеличением m. Оптимизация для m = 3 показывает, что для нечётных m возможна оптимизация лучше, чем самый общий, которая, правда, быстро ухудшается при росте m. Также, ввиду того, что нечётные m взаимно просты со степенями двойки, не удаётся эффективно использовать систему команд в общем случае. По этим причинам оптимизацию для произвольных нечётных m решено не делать.

5.2.2. Как вы можете объяснить разницу в скорости для четных и нечетных m?

Разница заключается в параллельном расчёте нескольких максимумов. Для чётных m можно создать программу, в которой можно организовать общую реализацию упаковки, использующую параллельную обработку нескольких элементов. А для нечётных m приходится производить последовательную обработку элементов массива, используя общую реализацию упаковки.

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

Приложение. Исходные тексты программ

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

Листинг 41: pack_type.h

```
#include <stdio.h>
#include <string.h>

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;
```

Листинг 42: main.c

```
#include "pack_type.h"

#define N 480
#pragma DATA_ALIGN(A, 1)
u8 A[N], p1[N], p2[N], p3[N], p4[N], p5[N], p6[N], p7[N], p8[N], p9[N], p10[N],
      p11[N], p12[N], p13[N], p14[N], p15[N], p16[N], p17[N], p18[N], p19[N];

int c62_pack_scale_no_opt(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x1_opt_1(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x2_opt_2(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x4_opt_3(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_1_opt_4(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_2_opt_5(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_3_opt_6(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_4_opt_7(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_no_opt(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x4_opt_1(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x4_opt_2(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x8_opt_3(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x2_opt_4(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_1_opt_5(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_2_opt_6(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_3_opt_7(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_4_opt_8(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_x1_opt_9(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale(const u8 *in, int n, int m, u8* result);
int c64_pack_scale(u8 *in, int n, int m, u8* result);

void print_scale(const u8 *array, int n, char *prefix, u8 *in, int m)
{
   printf("-------------------------------------------------\n");
   printf("%s\nquantity of bytes: %d\n", prefix, n);
   printf("begin of packed array: %d, %d, %d, %d;\n", array[0], array[1],
         array[2], array[3]);
   c62_pack_scale_no_opt(in, N, m, p1);
   if (memcmp(array, p1, n) == 0)
```

```
      printf("array equal to array of reference function.\n\n");
   else
      printf("array don't equal to array of reference function.\n\n");
}

int main(void) {
   int i, m, nd, c1, c2, c3, c4, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15,
         c16, c17, c18;
   char k;
   nd = N / 2;
   for (i = 0; i < N; i++) {
      if (i < nd)
         k = i;
      else
         k = N - i;
      A[i] = k;
   }
   m = 8;
   for (i = 0; i < 100; i++) {
      c62_pack_scale_no_opt(A, N, m, p1);
      c2 = c62_pack_scale_x1_opt_1(A, N, m, p2);
      c3 = c62_pack_scale_x2_opt_2(A, N, m, p3);
      c4 = c62_pack_scale_x4_opt_3(A, N, m, p4);
      c64_pack_scale_no_opt(A, N, m, p5);
      c6 = c64_pack_scale_x4_opt_1(A, N, m, p6);
      c7 = c64_pack_scale_x4_opt_2(A, N, m, p7);
      c8 = c64_pack_scale_x8_opt_3(A, N, m, p9);
      c9 = c64_pack_scale_x2_opt_4(A, N, m, p8);
      c10 = c64_pack_scale_1_opt_5(A, N, p10);
      c11 = c64_pack_scale_2_opt_6(A, N, p11);
      c12 = c62_pack_scale_1_opt_4(A, N, p12);
      c13 = c64_pack_scale_3_opt_7(A, N, p13);
      c14 = c62_pack_scale_2_opt_5(A, N, p14);
      c15 = c62_pack_scale_3_opt_6(A, N, p15);
      c16 = c62_pack_scale_4_opt_7(A, N, p16);
      c17 = c64_pack_scale_4_opt_8(A, N, p17);
      c18 = c64_pack_scale_x1_opt_9(A, N, m, p18);

   }
   printf("n: %d, m: %d\n", N, m);
   print_scale(p2, c2, "c62_pack_scale_x1_opt_1:", A, m);
   print_scale(p3, c3, "c62_pack_scale_x2_opt_2:", A, m);
   print_scale(p4, c4, "c62_pack_scale_x4_opt_3:", A, m);
   print_scale(p12, c12, "c62_pack_scale_1_opt_4:", A, 1);
   print_scale(p14, c14, "c62_pack_scale_2_opt_5:", A, 2);
   print_scale(p15, c15, "c62_pack_scale_3_opt_6:", A, 3);
   print_scale(p16, c16, "c62_pack_scale_4_opt_7:", A, 4);
   print_scale(p6, c6, "c64_pack_scale_x4_opt_1:", A, m);
   print_scale(p7, c7, "c64_pack_scale_x4_opt_2:", A, m);
   print_scale(p9, c8, "c64_pack_scale_x8_opt_3:", A, m);
   print_scale(p8, c9, "c64_pack_scale_x2_opt_4:", A, m);
   print_scale(p10, c10, "c64_pack_scale_1_opt_5:", A, 1);
   print_scale(p11, c11, "c62_pack_scale_2_opt_6:", A, 2);
   print_scale(p13, c13, "c62_pack_scale_3_opt_7:", A, 3);
   print_scale(p17, c17, "c62_pack_scale_4_opt_8:", A, 4);
   print_scale(p18, c18, "c62_pack_scale_x1_opt_9:", A, m);
   c1 = c62_pack_scale(A, N, m, p1);
```

```
   print_scale(p1, c1, "C62_dispatcher:", A, m);
   c1 = c64_pack_scale(A, N, m, p1);
   print_scale(p1, c1, "C64_dispatcher:", A, m);
   return 0;
```

Листинг 43: pack_scale_no_opt_c62.c

```
// Эталонная программа для уплотнения РЛС сигнала (ядро C62)
#include "pack_type.h"

int c62_pack_scale_no_opt(const u8 *in, int n, int m, u8* result)
{
   int k;
   u8 max;
   int count = 0;
   while (n >= m) {
      max = 0;
      for (k = 0; k < m; k++) {
         u8 x = *in++;
         max = (x > max) ? x : max;
      }
      *result++ = max;
      n -= m;
      count++;
   }
   return count;
}
```

Листинг 44: pack_scale_no_opt_c64.c

```
// Эталонная программа для уплотнения РЛС сигнала (ядро C64)
#include "pack_type.h"

int c64_pack_scale_no_opt(const u8 *in, int n, int m, u8* result)
{
   int k;
   u8 max;
   int count = 0;
   while (n >= m) {
      max = 0;
      for (k = 0; k < m; k++) {
         u8 x = *in++;
         max = (x > max) ? x : max;
      }
      *result++ = max;
      n -= m;
      count++;
   }
   return count;
}
```

Листинг 45: pack_csale_opt_c62.c

```c
// Оптимизированная программа для уплотнения РЛС сигнала (ядро C62)
#include "pack_type.h"

int c62_pack_scale_x1_opt_1(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   int count = 0;
   int countb = 0;
   u8 max = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k++) {
      u8 x = *in++;
      max = (x > max) ? x : max;
      count++;
      if (count == m) {
         *result++ = max;
         max = 0;
         count = 0;
         countb++;
      }
   }
   return countb;
}

int c62_pack_scale_x2_opt_2(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   int count = 0;
   int countb = 0;
   u8 max = 0;
   u8 max1 = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 2) {
      u8 x = *in++;
      u8 y = *in++;
      max = (x > max) ? x : max;
      max1 = (y > max1) ? y : max1;
      count += 2;
      if (count == m) {
         // конструкция для уменьшения завязки по данным
         if (max > max1)
            *result++ = max;
         if (max1 > max)
            *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb++;
      }
   }
   return countb;
}
```

```c
int c62_pack_scale_x4_opt_3(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   int count = 0;
   int countb = 0;
   u8 max = 0;
   u8 max1 = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 4) {
      u8 x = *in++;
      u8 y = *in++;
      u8 z = *in++;
      u8 a = *in++;
      x = (x > z) ? x : z;
      y = (y > a) ? y : a;
      max = (x > max) ? x : max;
      max1 = (y > max1) ? y : max1;
      count += 4;
      if (count == m) {
         // конструкция для уменьшения завязки по данным
         if (max > max1)
            *result++ = max;
         if (max1 > max)
            *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb++;

      }
   }
   return countb;
}

int c62_pack_scale_1_opt_4(const u8 *in, int n, u8*restrict result)
{
   int i;
   _nassert(((int) (in) & 0x3) == 0);
   u32 *din = (u32 *) in;
   u32 *restrict dres = (u32 *) result;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 4) {
      u32 x = *din++;
      *dres++ = x;
   }
   return n;
}

int c62_pack_scale_2_opt_5(const u8 *in, int n, u8*restrict result)
{
   int i;
   int count = 0;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 4) {
      u8 x = *in++;
      u8 y = *in++;
      *result++ = (x > y) ? x : y;
```

```
        u8 a = *in++;
        u8 b = *in++;
        *result++ = (a > b) ? a : b;
        count += 2;
    }
    return count;
}

int c62_pack_scale_3_opt_6(const u8 *in, int n, u8*restrict result)
{
    int i;
    int count = 0;
    u8 max;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
    for (i = 0; i < n; i += 3) {
        u8 x = *in++;
        u8 y = *in++;
        u8 z = *in++;
        max = (z > y) ? z : y;
        *result++ = (x > max) ? x : max;
        count++;
    }
    return count;
}

int c62_pack_scale_4_opt_7(const u8 *in, int n, u8*restrict result)
{
    int i;
    int count = 0;
    u8 max, max1;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
    for (i = 0; i < n; i += 4) {
        u8 x = *in++;
        u8 y = *in++;
        u8 z = *in++;
        u8 a = *in++;
        max = (z > a) ? z : a;
        max1 = (x > y) ? x : y;
        *result++ = (max > max1) ? max : max1;
        count++;
    }
    return count;
}
```

Листинг 46: pack_scale_opt_c64.c

```
// Оптимизированная программа для уплотнения РЛС сигнала (ядро C64)
#include "pack_type.h"

int c64_pack_scale_x4_opt_1(const u8 *in, int n, int m, u8*restrict result)
{
    int k;
    int count = 0;
```

```
   int countb = 0;
   u8 max = 0;
   u8 max1 = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 4) {
      u8 x = *in++;
      u8 y = *in++;
      u8 z = *in++;
      u8 a = *in++;
      x = (x > z) ? x : z;
      y = (y > a) ? y : a;
      max = (x > max) ? x : max;
      max1 = (y > max1) ? y : max1;
      count += 4;
      if (count == m) {
         // структура для уменьшения завязки по данным
         if (max > max1)
            *result++ = max;
         if (max1 > max)
            *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb++;

      }
   }
   return countb;
}

int c64_pack_scale_x4_opt_2(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   u32 max = 0;
   u32 max1 = 0;
   u32 max2 = 0;
   u32 max3 = 0;
   int count = 0;
   int countb = 0;
   int m1 = m >> 2;
   u32 *din = (u32 *) in;
   u32 *dres = (u32 *) result;
   int m12 = _mpy(m1, 2);
   int m13 = _mpy(m1, 3);
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 16) {
      u32 v = *(din + m13);
      u32 z = *(din + m12);
      u32 y = *(din + m1);
      u32 x = *din++;
      max = _maxu4(max, v);
      max1 = _maxu4(max1, z);
      max2 = _maxu4(max2, y);
      max3 = _maxu4(max3, x);
      count += 4;
      if (count == m) {
         u32 l = _packl4(max, max1);
```

```
            u32 h = _packh4(max, max1);
            u32 maxlh = _maxu4(l, h);
            u32 l1 = _packl4(max2, max3);
            u32 h1 = _packh4(max2, max3);
            u32 maxlh1 = _maxu4(l1, h1);
            u32 pack16l = _packl4(maxlh, maxlh1);
            u32 pack16h = _packh4(maxlh, maxlh1);
            *dres++ = _maxu4(pack16l, pack16h);
            max = 0;
            max1 = 0;
            max2 = 0;
            max3 = 0;
            count = 0;
            countb += 4;
            din += m13;
         }
      }
      return countb;
}

int c64_pack_scale_x8_opt_3(const u8 *in, int n, int m, u8*restrict result)
{
      int k;
      u32 max = 0;
      u32 maxs32 = 0;
      int count = 0;
      int countb = 0;
      u64 *din = (u64 *) in;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
      for (k = 0; k < n; k += 8) {
         u64 x = *din++;
         maxs32 = _maxu4(maxs32, x >> 32);
         max = _maxu4(max, x);
         count += 8;
         if (count == m) {
            max = _maxu4(max, maxs32);
            max = _maxu4(max, max >> 16);
            *result++ = _maxu4(max, max >> 8);
            max = 0;
            maxs32 = 0;
            count = 0;
            countb++;
         }
      }
      return countb;
}

int c64_pack_scale_x2_opt_4(const u8 *in, int n, int m, u8*restrict result)
{
      int k;
      u32 max = 0;
      u32 max1 = 0;
      int count = 0;
      int countb = 0;
      int m1 = m >> 1;
      u16 *din = (u16 *) in;
```

```
#pragma UNROLL(2)
#pragma MUST_ITERATE(2, ,2)
   for (k = 0; k < n; k += 4) {
      u32 y = *(din + m1);
      u32 x = *din++;
      max = _maxu4(max, y);
      max1 = _maxu4(max1, x);
      count += 2;
      if (count == m) {
         *result++ = _maxu4(max1, max1 >> 8);
         *result++ = _maxu4(max, max >> 8);
         count = 0;
         countb += 2;
         din += m1;
         max = 0;
         max1 = 0;
      }
   }
   return countb;
}

int c64_pack_scale_1_opt_5(const u8 *in, int n, u8*restrict result)
{
   _nassert(((int) (in) & 0x7) == 0);
   int i;
   u64 *din = (u64 *) in;
   u64 *dres = (u64 *) result;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 16) {
      *dres++ = *din++;
      *dres++ = *din++;
   }
   return n;
}

int c64_pack_scale_2_opt_6(const u8 *in, int n, u8*restrict result)
{
   int i;
   int count = 0;
   u64 *din = (u64 *) in;
   u32 *dres = (u32 *) result;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 8) {
      u64 x = *din++;
      u32 l = _packl4(x >> 32, x);
      u32 h = _packh4(x >> 32, x);
      u32 max = _maxu4(h, l);
      *dres++ = max;
      count += 4;
   }
   return count;
}

int c64_pack_scale_3_opt_7(const u8 *in, int n, u8*restrict result)
{
   int i;
```

```
   int count = 0;
   u32 m8 = 256;
   u32 *din = (u32 *) in;
   u32 *dres = (u32 *) result;
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 12) {
      u32 x = *din++;
      u32 y = *din++;
      u32 z = *din++;
      u8 x0 = x;
      u8 x1 = x >> 8;
      u8 x2 = x >> 16;
      u8 y2 = y >> 16;
      u8 y3 = y >> 24;
      u8 z0 = z;
      u32 a = _mpy(x >> 24, m8) + x0;
      u32 b = _mpy(y, m8) + x1;
      u32 c = _mpy(y >> 8, m8) + x2;
      u32 d = _mpy(z >> 8, m8) + y2;
      u32 e = _mpy(z >> 16, m8) + y3;
      u32 f = _mpy(z >> 24, m8) + z0;
      a = _pack2(d, a);
      b = _pack2(e, b);
      c = _pack2(f, c);
      u32 max = _maxu4(a, b);
      max = _maxu4(max, c);
      *dres++ = max;
      count += 4;
   }
   return count;
}

int c64_pack_scale_4_opt_8(const u8 *in, int n, u8*restrict result)
{
   int i;
   u32 max;
   int count = 0;
   u64 *din = (u64 *) in;
   u16 *dres = (u16 *) result;
#pragma UNROLL(2)
#pragma MUST_ITERATE(1)
   for (i = 0; i < n; i += 8) {
      u64 x = *din++;
      u32 l = _packl4(x >> 32, x);
      u32 h = _packh4(x >> 32, x);
      max = _maxu4(l, h);
      max = _maxu4(max >> 8, max);
      *dres++ = _packl4(max, max);
      count += 2;
   }
   return count;
}

int c64_pack_scale_x1_opt_9(const u8 *in, int n, int m, u8*restrict result)
{
   int k;
   u32 max = 0;
```

```
   u32 max1 = 0;
   int count = 0;
   int countb = 0;
#pragma MUST_ITERATE(1)
   for (k = 0; k < n; k += 2) {
      u32 y = *(in + m);
      u32 x = *in++;
      max = _maxu4(max, x);
      max1 = _maxu4(max1, y);
      count++;
      if (count == m) {
         *result++ = max;
         *result++ = max1;
         max = 0;
         max1 = 0;
         count = 0;
         countb += 2;
         in += m;
      }
   }
   return countb;
}
```

Листинг 47: Dispatcher.c

```
#include "pack_type.h"

int c62_pack_scale_no_opt(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x1_opt_1(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x2_opt_2(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_x4_opt_3(const u8 *in, int n, int m, u8*restrict result);
int c62_pack_scale_1_opt_4(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_2_opt_5(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_3_opt_6(const u8 *in, int n, u8*restrict result);
int c62_pack_scale_4_opt_7(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_no_opt(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x4_opt_1(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x4_opt_2(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x8_opt_3(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_x2_opt_4(const u8 *in, int n, int m, u8*restrict result);
int c64_pack_scale_1_opt_5(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_2_opt_6(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_3_opt_7(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_4_opt_8(const u8 *in, int n, u8*restrict result);
int c64_pack_scale_x1_opt_9(const u8 *in, int n, int m, u8*restrict result);

int c62_pack_scale(const u8 *in, int n, int m, u8* result)
{
   switch (m) {
   case 1:
      return c62_pack_scale_1_opt_4(in, n, result);
   case 2:
      return c62_pack_scale_2_opt_5(in, n, result);
   case 3:
      return c62_pack_scale_3_opt_6(in, n, result);
```

```
   case 4:
      return c62_pack_scale_4_opt_7(in, n, result);
   default:
      if ((m % 4) == 0)
         return c62_pack_scale_x4_opt_3(in, n, m, result);
      else if ((m % 2) == 0)
         return c62_pack_scale_x2_opt_2(in, n, m, result);
      else
         return c62_pack_scale_x1_opt_1(in, n, m, result);
   }
}

int c64_pack_scale(u8 *in, int n, int m, u8* result)
{
   switch (m) {
   case 1:
      return c64_pack_scale_1_opt_5(in, n, result);
   case 2:
      return c64_pack_scale_2_opt_6(in, n, result);
   case 3:
      return c64_pack_scale_3_opt_7(in, n, result);
   case 4:
      return c64_pack_scale_4_opt_8(in, n, result);
   default:
      if ((m % 8) == 0)
         return c64_pack_scale_x8_opt_3(in, n, m, result);
      else if ((m % 4) == 0)
         return c64_pack_scale_x4_opt_2(in, n, m, result);
      else if ((m % 2) == 0)
         return c64_pack_scale_x2_opt_4(in, n, m, result);
      else
         return c64_pack_scale_x1_opt_9(in, n, m, result);
   }
}
```

Вывод программы при запуске:

```
n: 480, m: 8
--------------------------------------------------
c62_pack_scale_x1_opt_1:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_x2_opt_2:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_x4_opt_3:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.
```

```
--------------------------------------------------
c62_pack_scale_1_opt_4:
quantity of bytes: 480
begin of packed array: 0, 1, 2, 3;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_2_opt_5:
quantity of bytes: 240
begin of packed array: 1, 3, 5, 7;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_3_opt_6:
quantity of bytes: 160
begin of packed array: 2, 5, 8, 11;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_4_opt_7:
quantity of bytes: 120
begin of packed array: 3, 7, 11, 15;
array equal to array of reference function.

--------------------------------------------------
c64_pack_scale_x4_opt_1:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
c64_pack_scale_x4_opt_2:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
c64_pack_scale_x8_opt_3:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
c64_pack_scale_x2_opt_4:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
c64_pack_scale_1_opt_5:
quantity of bytes: 480
begin of packed array: 0, 1, 2, 3;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_2_opt_6:
```

```
quantity of bytes: 240
begin of packed array: 1, 3, 5, 7;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_3_opt_7:
quantity of bytes: 160
begin of packed array: 2, 5, 8, 11;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_4_opt_8:
quantity of bytes: 120
begin of packed array: 3, 7, 11, 15;
array equal to array of reference function.

--------------------------------------------------
c62_pack_scale_x1_opt_9:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
C62_dispatcher:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.

--------------------------------------------------
C64_dispatcher:
quantity of bytes: 60
begin of packed array: 7, 15, 23, 31;
array equal to array of reference function.
```

КОНЕЦ ДОКУМЕНТА