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 №8

Расчёт статистических параметров

выполнил студент группы 3721

21.12.2020 _____ А. А. Булыгин

Санкт-Петербург, 2020

Изм.	Лист	№ докум.	Подп.	Дата	ГУАП.3721.0С-008		
Разраб.		Булыгин			Оптимальные системы Практическое задание №8	Лит.	Лист 1 / Листов 37
Проверил		Уздяев				Р	
Н. контр.						ГУАП	
Утв.		Савельев					

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 №8. Расчёт статистических параметров. Имеется массив знаковых 16-битных чисел. Необходимо рассчитать по ним следующие значения: минимум, максимум, среднее значение, дисперсию.

```
struct int_statistics {
   int min;
   int max;
   int average;
   int variance;
};

void calc_statistics(const short *data, int n, struct int_statistics *result);
```

Дополнительные вопросы: 1) Что эффективнее, считать все параметры статистики в одном цикле, или разбивать на несколько?

1.2. Общие принципы оформления задания

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

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

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

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

Файл с оптимизированными функциями компилируется аналогично эталонному, с получением обратной связи от компилятора, но для каждого шага оптимизации. Для удобства исследования и повторения результатов каждый шаг оптимизации оформлен в виде отдельной функции с именем, заканчивающимся суффиксом _opt_N, где N обозначает номер шага оптимизации, например, calc_statistics_opt_2 для шага оптимизации 2.

2. Базовый алгоритм выполнения задания

2.1. Эталонный алгоритм

При объявлении переменной для вычисления суммы квадратов использован тип long, так как при возведении в квадрат элемента массива, значение которого близко к 32768 происходит переполнение ячейки памяти, если переменная объявления с типом int. Хотя это и приводит к снижению производительности, но позволяет производить все расчёты 16-битных данных правильно.

Листинг 1: Эталонный алгоритм расчёта статистики

```
// Эталонная программа вычистения статистических параметров
struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};

void calc_statistics_no_opt(short *data, int n, struct int_statistics *result)
{
         int i;
         short min=32767;
         short max=-32768;
         int summ=0;
         long summ2=0;
         int x;
         for (i=0; i<n; i++)
         {
            x=data[i];
            if(x<min) min=x;
            if(x>max) max=x;
            summ+=x;
            summ2+=x*x;

         }
         // дисперисия расчитывается как разность суммы квадратов и квадрата суммы
         // всех элементов массива, деленное на количество элементов.
         // полученная разность делится на количество элементов массива
         result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
         result->average=summ/n;
         result->min=min;
         result->max=max;
}
```

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="calc_static_no_opt.pp"  "../calc_static_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.

Листинг 2: Фидбэк эталонного алгоритма, собранного для C62x

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../cacl_static_no_opt.c
;*      Loop source line                 : 16
;*      Loop opening brace source line   : 17
;*      Loop closing brace source line   : 24
;*      Known Minimum Trip Count         : 1
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound     : 2
;*      Partitioned Resource Bound(*)    : 2
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     2*       1
;*      .S units                     0        1
;*      .D units                     1        0
;*      .M units                     1        0
;*      .X cross paths               0        2*
;*      .T address paths             1        0
;*      Long read paths              1        0
;*      Long write paths             1        0
;*      Logical  ops (.LS)           0        1     (.L or .S unit)
;*      Addition ops (.LSD)          2        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        2*
;*      Bound(.L .S .D .LS .LSD)     2*       2*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 5 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      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 -mh8
```

```
;*
;*      Minimum safe trip count       : 4
;*----------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP PROLOG
           LDH     .D1T1   *A9++,A8          ; |19| (P) <1,0>  ^
           NOP             1

           MV      .L1     A7,A4
||         B       .S2     $C$L4             ; |16| (P) <0,4>
||         LDH     .D1T1   *A9++,A8          ; |19| (P) <2,0>  ^

           MV      .S1     A10,A4
||         ADD     .D1     A8,A4,A5          ; |21| (P) <0,5>
||         CMPLT   .L2X    B4,A8,B0          ; |19| (P) <0,5>  ^
||         CMPGT   .L1     A3,A8,A1          ; |20| (P) <0,5>  ^

           SUB     .L1X    B1,4,A2
||         SUB     .D2     B1,4,B1
|| [!A1]   MV      .S1     A8,A3             ; |20| (P) <0,6>  ^
|| [!B0]   MV      .L2X    A8,B4             ; |19| (P) <0,6>  ^
||         MPY     .M1     A8,A8,A7          ; |22| (P) <0,6>
||         B       .S2     $C$L4             ; |16| (P) <1,4>
||         LDH     .D1T1   *A9++,A8          ; |19| (P) <3,0>  ^

           MV      .S1     A11,A5
||         ADD     .S2     2,B1,B2
||         ADD     .D1     A8,A5,A6          ; |21| (P) <1,5>
||         CMPLT   .L2X    B4,A8,B0          ; |19| (P) <1,5>  ^
||         CMPGT   .L1     A3,A8,A1          ; |20| (P) <1,5>  ^

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

           ADD     .L1     A7,A5:A4,A5:A4    ; |22| <0,8>
|| [!A1]   MV      .S1     A8,A3             ; |20| <1,6>  ^
|| [!B0]   MV      .L2X    A8,B4             ; |19| <1,6>  ^
||         MPY     .M1     A8,A8,A7          ; |22| <1,6>
|| [ B1]   B       .S2     $C$L4             ; |16| <2,4>
|| [ A2]   LDH     .D1T1   *A9++,A8          ; |19| <4,0>  ^

   [ B2]   SUB     .S2     B2,1,B2           ; <0,9>
|| [ A2]   SUB     .S1     A2,1,A2           ; <0,9>
|| [ B2]   ADD     .D1     A8,A6,A6          ; |21| <2,5>
||         CMPLT   .L2X    B4,A8,B0          ; |19| <2,5>  ^
||         CMPGT   .L1     A3,A8,A1          ; |20| <2,5>  ^
|| [ B1]   SUB     .D2     B1,1,B1           ; |16| <3,3>

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

Предварительный анализ функции с эталонным алгоритмом показывает, что:

- Цикл имеет небольшой пролог и не имеет эпилога.
- Ядро цикла умещается в 2 пакета выборки инструкций (fetch packet). Использовано

за 2 такта 75% вычислительных ресурсов ядра, это говорит об изначальной высокой производительности алгоритма, так как ядро цикла в основном занято выполнением инструкций для получения статистических параметров

- Для того, чтобы сравнить 2 числа используется 2 юнита в ядре цикла за 2 такта, при таком алгоритме большего количества сравнений получить не получается, так как юниты выполняют команды сложения, вычитания, загрузки слов.
- Ядро использует 1 умножитель для расчёта статистики. За ii=2 доступно 4 умножителя, но они не используются, как как за итерацию удаётся загрузить только 1 слово, что говорит о том, что умножители слабо загружены в данном алгоритме
- Сторона A более загружена, чем сторона B, на корой используется 1 раз за ядро цикла .L юнит, и не используются .M и .D юниты, что говорит о том, что в оптимизированной программе нужно произвести балансировку ресурсов по сторонам процессора.

3. Создание оптимального алгоритма (C62x)

3.1. Разворачивание цикла

Листинг 3: Добавление прагмы кратности 4

```
// Оптимизированная программа статистических параметров
struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};

void calc_statistics_opt(short * data,int n, struct int_statistics * result)
{
   int i;
   short min=32767;
   short max=-32768;
   int summ=0;
   long summ2=0;
   short x;
   #pragma MUST_ITERATE(4,  ,4)
   for (i=0; i<n; i++)
   {
      x=data[i];
      if(x<min) min=x;
      if(x>max) max=x;
      summ+=x;
      summ2+=x*x;
   }
   result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
   result->average=summ/n;
   result->min=min;
   result->max=max;
}
```

Компиляция с ключами:

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

Позволяет компилятору развернуть цикл:

Листинг 4: Добавление прагмы кратности 4

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../cacl_statistics_opt.c
;*      Loop source line                : 91
;*      Loop opening brace source line  : 92
;*      Loop closing brace source line  : 98
;*      Known Minimum Trip Count        : 4
;*      Known Max Trip Count Factor     : 4
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound    : 2
;*      Partitioned Resource Bound(*)   : 2
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                    2*       1
;*      .S units                    0        1
;*      .D units                    1        0
;*      .M units                    1        0
;*      .X cross paths              0        2*
;*      .T address paths            1        0
;*      Long read paths             1        0
;*      Long write paths            1        0
;*      Logical  ops (.LS)          0        1     (.L or .S unit)
;*      Addition ops (.LSD)         2        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)            1        2*
;*      Bound(.L .S .D .LS .LSD)    2*       2*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 5 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      Collapsed epilog stages       : 3
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 6 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L10:    ; PIPED LOOP PROLOG

           B       .S2     $C$L11           ; |91| (P) <0,4>
||         LDH     .D1T1   *A9++,A8         ; |94| (P) <1,0>  ^

           ZERO    .L1     A4

           ZERO    .L1     A5
```

```
||          MV      .S1     A7,A6
||          SUB     .L2     B7,3,B1
||          B       .S2     $C$L11          ; |91| (P) <1,4>
||          LDH     .D1T1   *A9++,A8        ; |94| (P) <2,0>  ^

            MVK     .S2     0x1,B2          ; init prolog collapse predicate
||          ADD     .S1X    2,B1,A2
||          ADD     .D1     A8,A6,A6        ; |96| (P) <0,5>
||          CMPLT   .L2X    B4,A8,B0        ; |94| (P) <0,5>  ^
||          CMPGT   .L1     A3,A8,A1        ; |95| (P) <0,5>  ^

;** --------------------------------------------------------------------------*
$C$L11:    ; PIPED LOOP KERNEL
$C$DW$L$_calc_statistics_opt$3$B:

   [!B2]   ADD     .L1     A7,A5:A4,A5:A4  ; |97| <0,8>
|| [!A1]   MV      .S1     A8,A3           ; |95| <1,6>  ^
|| [!B0]   MV      .L2X    A8,B4           ; |94| <1,6>  ^
||          MPY     .M1     A8,A8,A7        ; |97| <1,6>
|| [ B1]   B       .S2     $C$L11          ; |91| <2,4>
||          LDH     .D1T1   *A9++,A8        ; |94| <4,0>  ^

   [ B2]   SUB     .S2     B2,1,B2         ; <0,9>
|| [ A2]   SUB     .S1     A2,1,A2         ; <0,9>
|| [ A2]   ADD     .D1     A8,A6,A6        ; |96| <2,5>
||          CMPLT   .L2X    B4,A8,B0        ; |94| <2,5>  ^
||          CMPGT   .L1     A3,A8,A1        ; |95| <2,5>  ^
|| [ B1]   SUB     .D2     B1,1,B1         ; |91| <3,3>

$C$DW$L$_calc_statistics_opt$3$E:
;** --------------------------------------------------------------------------*
$C$L12:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Как видим, цикл не разворачивается. При добавлении прагмы UNROLL(2) цикл разворачивается, но программа обрабатывает массив медленнее эталонной программы. Это происходит из-за того, что ядро цикла становиться менее загруженным. При добавлении прагмы UNROLL(2) также наблюдается снижение скорости обработки массива, по сравнению с полученной программой без прагмы UNROLL(4). Прирост производительности составляет 4,7% для массива со 100 элементами:

Name	Calls	Excl Count Average
calc_statistics_no_opt(short *, int, struct int_statistics *)	100	278.88
calc_statistics_opt(short *, int, struct int_statistics *)	100	266.43

Рисунок 1: Сравнение эталона с простым разворачиванием цикла

Это можно обосновать тем, что в оптимизированном варианте использовалась опция -mh12, которая позволила избавиться от длинного эпилога, а также тем, что теперь ядро цикла загружено на 75%. Таким образом, главным сдерживающим фактором остается сложный расчёт минимума и максимума.

3.2. Использование альтернативных команд

Заменим условные операторы на операторы ? : для упрощения условий.

Листинг 5: Изменение команд сравнения

```
struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};

void calc_statistics_opt_2(short *data, int n, struct int_statistics *result)
{
        int i;
        short min=32767;
        short max=-32768;
        int summ=0;
        long summ2=0;
        short x;
        #pragma MUST_ITERATE(4,  ,4)
        for (i=0; i<n; i++)
        {
           x =   data[i];
           min=x<min ? x:min;
           max=x>max ? x:max;
           summ+=x;
           summ2+=x*x;
        }
        result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
        result->average=summ/n;
        result->min=min;
        result->max=max;
}
```

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="calc_static_opt_2.pp"
```

Конвейер стал выглядеть следующим образом:

Листинг 6: Фидбэк от компилятора для альтернативных команд

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../cacl_statistics_opt.c
;*      Loop source line                : 116
;*      Loop opening brace source line  : 117
;*      Loop closing brace source line  : 123
;*      Known Minimum Trip Count        : 4
;*      Known Max Trip Count Factor     : 4
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound    : 2
;*      Partitioned Resource Bound(*)   : 2
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     2*       1
```

```
;*      .S units                     0        1
;*      .D units                     1        0
;*      .M units                     1        0
;*      .X cross paths               0        2*
;*      .T address paths             1        0
;*      Long read paths              1        0
;*      Long write paths             1        0
;*      Logical  ops (.LS)           0        1     (.L or .S unit)
;*      Addition ops (.LSD)          2        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             1        2*
;*      Bound(.L .S .D .LS .LSD)     2*       2*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Schedule found with 5 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      Collapsed epilog stages       : 3
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 6 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L7:    ; PIPED LOOP PROLOG

           B       .S2     $C$L8             ; |116| (P) <0,4>
||         LDH     .D1T1   *A9++,A8          ; |119| (P) <1,0>  ^

           ZERO    .L1     A4

           ZERO    .L1     A5
||         MV      .S1     A7,A6
||         SUB     .L2     B7,3,B1
||         B       .S2     $C$L8             ; |116| (P) <1,4>
||         LDH     .D1T1   *A9++,A8          ; |119| (P) <2,0>  ^

           MVK     .S2     0x1,B2            ; init prolog collapse predicate
||         ADD     .S1X    2,B1,A2
||         ADD     .D1     A8,A6,A6          ; |121| (P) <0,5>
||         CMPLT   .L2X    B4,A8,B0          ; |119| (P) <0,5>  ^
||         CMPGT   .L1     A3,A8,A1          ; |120| (P) <0,5>  ^

;** ---------------------------------------------------------------------------*
$C$L8:    ; PIPED LOOP KERNEL
$C$DW$L$_calc_statistics_opt_2$3$B:

   [!B2]   ADD     .L1     A7,A5:A4,A5:A4    ; |122| <0,8>
|| [!A1]   MV      .S1     A8,A3             ; |120| <1,6>  ^
|| [!B0]   MV      .L2X    A8,B4             ; |119| <1,6>  ^
||         MPY     .M1     A8,A8,A7          ; |122| <1,6>
|| [ B1]   B       .S2     $C$L8             ; |116| <2,4>
||         LDH     .D1T1   *A9++,A8          ; |119| <4,0>  ^

   [ B2]   SUB     .S2     B2,1,B2           ; <0,9>
|| [ A2]   SUB     .S1     A2,1,A2           ; <0,9>
```

```
|| [ A2]   ADD     .D1     A8,A6,A6          ; |121| <2,5>
||         CMPLT   .L2X    B4,A8,B0          ; |119| <2,5>  ^
||         CMPGT   .L1     A3,A8,A1          ; |120| <2,5>  ^
|| [ B1]   SUB     .D2     B1,1,B1           ; |116| <3,3>
$C$DW$L$_calc_statistics_opt_2$3$E:
;** --------------------------------------------------------------------------*
$C$L9:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Name	Calls	Excl Count Average
calc_statistics_no_opt(short *, int, struct int_statistics *)	100	278.88
calc_statistics_opt(short *, int, struct int_statistics *)	100	266.43
calc_statistics_opt_2(short *, int, struct int_statistics *)	100	266.45

Рисунок 2: Сравнение эталона с выровненным развернутым циклом

Видим прирост скорости по сравнению с эталоном такой же, как и для программы из пункта 3.1, так как компилятор использовал аналогичные функции.

3.3. Балансировка ресурсов по сторонам конвейера

Увеличения производительности можно добиться, развернув в ручную цикл.

Листинг 7: Ручное разворачивание еще в два раза

```
struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};

void calc_statistics_opt_3(const short *data, int n, struct int_statistics *result)
{
      int i;
      short min=32767;
      short max=-32768;
      int summ=0;
      short max1=-32768;
      short min1=32767;
      long summ2=0;
      short x;
      short y;
      #pragma MUST_ITERATE(2,  ,2)
      for (i=0; i<n; i+=2)
      {
              x =   data[i];
              y=data[i+1];
              min=x<min ? x:min;
              max=x>max ? x:max;
              min1=y<min1 ? y:min1;
              max1=y>max1 ? y:max1;
              summ+=x+y;
              summ2+=x*x+y*y;
      }
      min=min<min1 ? min:min1;
```

```
    max=max>max1 ? max:max1;
    result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
    result->average=summ/n;
    result->min=min;
    result->max=max;
}
```

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=16 -k
--preproc_with_compile --preproc_dependency="calc_static_opt_3.pp"
"../calc_static_opt_3.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 8: Фидбэк от компилятора с разворачиванием цикла в ручную

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../cacl_statistics_opt.c
;*      Loop source line                 : 144
;*      Loop opening brace source line   : 145
;*      Loop closing brace source line   : 154
;*      Known Minimum Trip Count         : 2
;*      Known Max Trip Count Factor      : 2
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound     : 3
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3*       2
;*      .S units                     0        1
;*      .D units                     2        0
;*      .M units                     1        1
;*      .X cross paths               1        0
;*      .T address paths             1        1
;*      Long read paths              1        0
;*      Long write paths             1        0
;*      Logical  ops (.LS)           1        0     (.L or .S unit)
;*      Addition ops (.LSD)          3        4     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             2        2
;*      Bound(.L .S .D .LS .LSD)     3*       3*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 3  Schedule found with 4 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      Collapsed epilog stages       : 2
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 1
;*
;*      Minimum required memory pad   : 8 bytes
;*
```

```
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP PROLOG

           B       .S2     $C$L5             ; |144| (P) <0,6>
||         LDH     .D1T2   *++A7(4),B8       ; |148| (P) <1,0>  ^

           MV      .L1     A12,A8
||         SUB     .L2     B7,2,B1
||         ZERO    .S1     A2
||         ZERO    .S2     B4                ; |146|
||         LDH     .D1T1   *+A7(2),A5        ; |150| (P) <1,1>  ^

           MV      .L1     A13,A9
||         ZERO    .D1     A6                ; |146|
||         ADD     .S2     1,B1,B2
||         MVKH    .S1     0x10000,A2        ; init prolog collapse predicate
||         ADD     .D2     B8,B4,B7          ; |152| (P) <0,5>
||         MPY     .M2     B8,B8,B4          ; |153| (P) <0,5>
||         CMPGT   .L2     B6,B8,B0          ; |149| (P) <0,5>  ^

;** ----------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP KERNEL
$C$DW$L$_calc_statistics_opt_3$3$B:

   [ B1]   B       .S2     $C$L5             ; |144| <1,6>
||         ADD     .S1     A5,A6,A6          ; |152| <1,6>
||         MPY     .M1     A5,A5,A4          ; |153| <1,6>
||         CMPLT   .L1     A10,A5,A1         ; |150| <1,6>  ^
|| [!B0]   MV      .D2     B8,B6             ; |149| <1,6>  ^
||         CMPLT   .L2     B5,B8,B0          ; |148| <1,6>  ^
||         LDH     .D1T2   *++A7(4),B8       ; |148| <3,0>  ^

   [!A1]   MV      .S1     A5,A10            ; |150| <1,7>  ^
||         CMPGT   .L1     A0,A5,A1          ; |151| <1,7>  ^
|| [!B0]   MV      .S2     B8,B5             ; |148| <1,7>  ^
|| [ B1]   SUB     .L2     B1,1,B1           ; |144| <2,4>
||         LDH     .D1T1   *+A7(2),A5        ; |150| <3,1>  ^

   [ A2]   MPYSU   .M1     2,A2,A2           ; <0,11>
|| [ B2]   SUB     .S2     B2,1,B2           ; <0,11>
|| [!A2]   ADD     .L1     A3,A9:A8,A9:A8    ; |153| <0,11>
||         ADD     .S1X    A4,B4,A3          ; |153| <1,8>
|| [!A1]   MV      .D1     A5,A0             ; |151| <1,8>  ^
|| [ B2]   ADD     .D2     B8,B7,B7          ; |152| <2,5>
||         MPY     .M2     B8,B8,B4          ; |153| <2,5>
||         CMPGT   .L2     B6,B8,B0          ; |149| <2,5>  ^

$C$DW$L$_calc_statistics_opt_3$3$E:
;** ----------------------------------------------------------------------------*
$C$L6:    ; PIPED LOOP EPILOG
;** ----------------------------------------------------------------------------*
```

Теперь за 3 такта вычисляется 2 умножения и 2 вычисления минимума и максимума, в отличие от 1 умножения и 1 вычисления минимума и максимума за 2 такта в предыдущем пункте. Это позволило получить прирост еще на 21%.

Name	Calls	Excl Count Average
calc_statistics_no_opt(short *, int, struct int_statistics *)	100	278.88
calc_statistics_opt(short *, int, struct int_statistics *)	100	266.43
calc_statistics_opt_2(short *, int, struct int_statistics *)	100	266.45
calc_statistics_opt_3(short *, int, struct int_statistics *)	100	221.47

Рисунок 3: Прирост производительности после балансировки нагрузки

В итоге имеем рост производительности в 1,26 раза. При обработке массива из 200 элементов прирост составляет 1,29 раз. Прирост небольшой так как эталонная программа имеет высокую загруженность цикла.

3.4. Написание функции на линейном ассемблере

Для улучшения расчётов попробуем разбить цикл на 2, записав вычисление дисперсии на линейном ассемблере, что может обеспечить более оптимальное распределение ресурсов процессора. Функция на ассемблере выглядит следующим образом:

Листинг 9: программа для вычисления суммы квадратов на линейном ассемблере

```
                .global     _summ2sa

_summ2sa:       .cproc          data, count
                .reg            x, mult, result:result1
                zero            result:result1
loop:
                ldh             *data++, x
                mpy             x, x, mult
                add             mult, result:result1, result:result1
                sub             count, 1, count
[count]         b               loop
                .return         result:result1
                .endproc
```

тогда программа выглядит следующим образом:

Листинг 10: программа для вычисления элементов статистики за 2 цикла

```
struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};

long long  summ2sa(short*,int);

void calc_statistics_opt_4(short * data,int n, struct int_statistics *result)
{
   int i;
   short min=32767;
   short max=-32768;
   int summ=0;
   short max1=32767;
   short min1=-32768;
   long summ2=0;
   int x;
   int y;
```

```
  #pragma MUST_ITERATE(4,  ,4)
  for (i=0; i<n; i+=2)
  {
     x=data[i];
     y=data[i+1];
     min=x<min ? x:min;
     max=x>max ? x:max;
     min1=y<min1 ? y:min1;
     max1=y>max1 ? y:max1;
     summ+=x+y;
  }
  summ2=summ2sa(data, n);
  min=min<min1 ? min:min1;
  max=max>max1 ? max:max1;
  result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
  result->average=summ/n;
  result->min=min;
 result->max=max;
}
```

Компиляция с ключами:

```
"C:/ti/ccsv5/tools/compiler/c6000_7.4.24/bin/cl6x" -mv6200 --abi=coffabi -O2 -g
--include_path="C:/ti/ccsv5/tools/compiler/c6000_7.4.24/include"
--display_error_number --diag_warning=225 -k --preproc_with_compile
--preproc_dependency="calc_static_opt_4.pp"  "../calc_static_opt_4.c"

"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=28 -k
--preproc_with_compile --preproc_dependency="summ2sa.pp"  "../summ2sa.sa"
```

Получаем фидбэк от компилятора для вычисления дисперсии:

Листинг 11: Фидбэк от компилятора для вычисления дисперсии

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop label : loop
;*      Loop found in file              : ../summ2sa.sa
;*      Loop source line                : 9
;*      Loop closing brace source line  : 13
;*      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                    1*       0
;*      .S units                    0        1*
;*      .D units                    1*       0
;*      .M units                    1*       0
;*      .X cross paths              0        0
;*      .T address paths            1*       0
;*      Long read paths             1*       0
;*      Long write paths            1*       0
```

```
;*      Logical  ops (.LS)           0        0     (.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 8 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 7
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 6
;*
;*      Minimum required memory pad   : 0 bytes
;*
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG
           B       .S2     $C$L2             ; |13| (P) <0,2>
           B       .S2     $C$L2             ; |13| (P) <1,2>
           B       .S2     $C$L2             ; |13| (P) <2,2>

           MV      .L1X    count,count'
||         B       .S2     $C$L2             ; |13| (P) <3,2>

           SUB     .L1     count',1,A1       ; init epilog collapse predicate
||         LDH     .D1T1   *data++,x         ; |9| (P) <0,0>
||         B       .S2     $C$L2             ; |13| (P) <4,2>
;** --------------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_summ2sa$3$B:

   [ B1]   SUB     .L2     B1,1,B1           ; <0,7>
|| [ A1]   SUB     .S1     A1,1,A1           ; <0,7>
|| [!B1]   ADD     .L1     mult,result:result1,result:result1 ; |11| <0,7>
||         MPY     .M1     x,x,mult          ; |10| <2,5>
|| [ count] B      .S2     $C$L2             ; |13| <5,2>
|| [ count] ADD    .D2     0xffffffff,count,count ; |12| <6,1>
|| [ A1]   LDH     .D1T1   *data++,x         ; |9| <7,0>
$C$DW$L$_summ2sa$3$E:
;** --------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Листинг 12: Фидбэк от компилятора для цикла без вычисления дисперсии

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../cacl_statistics_opt.c
;*      Loop source line                : 176
;*      Loop opening brace source line  : 177
;*      Loop closing brace source line  : 185
;*      Known Minimum Trip Count        : 4
```

```
;*      Known Max Trip Count Factor      : 4
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound     : 3
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                    3*       1
;*      .S units                    0        1
;*      .D units                    2        0
;*      .M units                    0        0
;*      .X cross paths              1        3*
;*      .T address paths            2        0
;*      Long read paths             0        0
;*      Long write paths            0        0
;*      Logical  ops (.LS)          0        2     (.L or .S unit)
;*      Addition ops (.LSD)         4        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)            2        2
;*      Bound(.L .S .D .LS .LSD)    3*       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   : 8 bytes
;*
;*      Minimum safe trip count       : 1
;*----------------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG

           MV      .L1     A11,A8
||         B       .S2     $C$L2             ; |176| (P) <0,3>
||         LDH     .D1T1   *++A9(4),A3       ; |180| (P) <1,0>

           MV      .L1     A10,A7
||         SUB     .L2     B4,2,B1
||         MV      .S2     B11,B5
||         ZERO    .S1     A5                ; |178|
||         LDH     .D1T1   *+A9(2),A4        ; |182| (P) <1,1>

           MV      .L2     B12,B4
||         ZERO    .S1     A6                ; |178|
||         ADD     .S2     1,B1,B2
||         ADD     .D1     A3,A5,A5          ; |184| (P) <0,5>
||         CMPLT   .L1X    B5,A3,A1          ; |180| (P) <0,5>  ^

;** ----------------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_calc_statistics_opt_4$3$B:

           ADD     .S1     A4,A6,A6          ; |184| <0,6>
|| [!A1]   MV      .L2X    A3,B5             ; |180| <0,6>  ^
||         CMPGT   .L1     A7,A3,A1          ; |181| <0,6>  ^
|| [ B1]   B       .S2     $C$L2             ; |176| <1,3>
||         LDH     .D1T1   *++A9(4),A3       ; |180| <2,0>
```

```
   [!A1]   MV      .S1     A3,A7           ; |181| <0,7>  ^
||         CMPLT   .L2X    B4,A4,B0        ; |182| <0,7>  ^
||         CMPGT   .L1     A8,A4,A1        ; |183| <0,7>  ^
||         LDH     .D1T1   *+A9(2),A4      ; |182| <2,1>

   [ B2]   SUB     .S2     B2,1,B2         ; <0,8>
|| [!B0]   MV      .L2X    A4,B4           ; |182| <0,8>  ^
|| [!A1]   MV      .D1     A4,A8           ; |183| <0,8>  ^
|| [ B2]   ADD     .S1     A3,A5,A5        ; |184| <1,5>
||         CMPLT   .L1X    B5,A3,A1        ; |180| <1,5>  ^
|| [ B1]   SUB     .D2     B1,1,B1         ; |176| <2,2>

$C$DW$L$_calc_statistics_opt_4$3$E:
;** --------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG
;** --------------------------------------------------------------------------*
```

Из-за разгрузки цикла, ядро стало менее нагруженное, так как операция умножения вынесена в отдельный цикл.

Профилирование кода дает следующие результаты:

Name	Calls	Excl Count Average
calc_statistics_no_opt(short *, int, struct int_statistics *)	100	278.88
calc_statistics_opt(short *, int, struct int_statistics *)	100	266.43
calc_statistics_opt_2(short *, int, struct int_statistics *)	100	266.45
calc_statistics_opt_3(short *, int, struct int_statistics *)	100	221.47
calc_statistics_opt_4(short *, int, struct int_statistics *)	100	289.05
main()	1	6674.00
summ2sa()	100	128.09

Рисунок 4: Результаты после разбиения цикла на 2

Как видим, скорость упала ниже варианта, рассмотренного в пункте 3.1 (так как цикл разбит на 2 для вычисления тактов, потраченных на вычисление 4 программы, необходимо суммировать количество тактов 4 программы и функции вычисления суммы квадратов, что в результате получим 417,14 такта. Это говорит о невыгодности разбивать цикл для данной задачи на процессоре TMS320C6200, также можно сказать, что дальнейшее ускорение ограничено загруженностью .L юнитов, выполняющих сравнение, и .S юнитов.

4. Создание оптимального алгоритма (C64x)

4.1. Запуск программы, занявшей меньше всего тактов, на ядре C64

Запустим Программу из пункта 3.3 и посмотрим, насколько быстро программа работает на ядре C64.

Листинг 13: Программа с ручным разворачиванием цикла

```
struct int_statistics {
   short min;
```

```
   short max;
   int average;
   int variance;
};

void calc_statistics_opt(short * data,int n, struct int_statistics *result)
{
        int i;
        short min=32767;
        short max=-32768;
        int summ=0;
        short max1=-32768;
        short min1=32767;
        long summ2=0;
        short x, y;
        #pragma MUST_ITERATE(4,  ,4)
        for (i=0; i<n; i+=2)
        {
                x=data[i];
                y=data[i+1];
                min=x<min ? x:min;
                max=x>max ? x:max;
                min1=y<min1 ? y:min1;
                max1=y>max1 ? y:max1;
                summ+=x+y;
                summ2+=x*x+y*y;
        }
        min=min<min1 ? min:min1;
        max=max>max1 ? max:max1;
        result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
        result->average=summ/n;
        result->min=min;
        result->max=max;
}
```

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=16 -k
--preproc_with_compile --preproc_dependency="calc_static_opt.pp"
"../calc_static_opt.c"
```

В итоге, получаем такой фидбэк от компилятора.

Листинг 14: Фидбэк от компилятора после дополнительного разворачивания

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../cacl_statistics_opt.c
;*      Loop source line                : 22
;*      Loop opening brace source line  : 23
;*      Loop closing brace source line  : 32
;*      Loop Unroll Multiple            : 2x
;*      Known Minimum Trip Count        : 2
;*      Known Max Trip Count Factor     : 2
;*      Loop Carried Dependency Bound(^) : 4
```

```
;*      Unpartitioned Resource Bound     : 6
;*      Partitioned Resource Bound(*)    : 6
;*      Resource Partition:
;*                                  A-side   B-side
;*      .L units                      6*       4
;*      .S units                      4        5
;*      .D units                      4        0
;*      .M units                      2        0
;*      .X cross paths                0        6*
;*      .T address paths              4        0
;*      Long read paths               0        0
;*      Long write paths              0        0
;*      Logical  ops (.LS)            2        0     (.L or .S unit)
;*      Addition ops (.LSD)           2        2     (.L or .S or .D unit)
;*      Bound(.L .S .LS)              6*       5
;*      Bound(.L .S .D .LS .LSD)      6*       4
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 6  Register is live too long
;*         ii = 6  Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Collapsed epilog stages       : 2
;*      Collapsed prolog stages       : 2
;*      Minimum required memory pad   : 16 bytes
;*
;*      Minimum safe trip count       : 1 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L4:    ; PIPED LOOP PROLOG
;** ---------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP KERNEL
$C$DW$L$_calc_statistics_opt$3$B:

   [ B0]   BDEC    .S2     $C$L5,B0          ; |22| <0,8>
|| [!B1]   ADD     .L2X    A16,B7,B7         ; |30| <0,8>
||         MIN2    .L1     A6,A16,A3         ; |26| <0,8>  ^
||         PACK2   .S1     A17,A16,A4        ; |31| <0,8>
||         DOTP2   .M1     A3,A3,A4          ; |31| <0,8>
|| [!A0]   LDH     .D1T1   *+A7(4),A18       ; |26| <1,2>

           MAX2    .L2X    B9,A16,B4         ; |27| <0,9>  ^
||         EXT     .S1     A3,16,16,A16      ; |26| <0,9>  ^
||         MAX2    .L1     A5,A17,A4         ; |29| <0,9>  ^
||         DOTP2   .M1     A4,A4,A3          ; |31| <0,9>
|| [!A0]   LDH     .D1T1   *+A7(2),A17       ; |30| <1,3>

   [!B1]   ADD     .D1     A18,A9,A9         ; |30| <0,10>
||         MIN2    .L2X    B8,A17,B4         ; |28| <0,10>  ^
||         EXT     .S2     B4,16,16,B5       ; |27| <0,10>  ^
||         MIN2    .L1     A16,A18,A19       ; |26| <0,10>  ^
||         EXT     .S1     A4,16,16,A4       ; |29| <0,10>  ^

   [ A0]   MPYSU   .M1     2,A0,A0           ; <0,11>
|| [!B1]   ADD     .D1     A3,A8,A8          ; |30| <0,11>
|| [!B1]   EXT     .S1     A19,16,16,A6      ; |26| <0,11>  ^
||         EXT     .S2     B4,16,16,B4       ; |28| <0,11>  ^
||         MAX2    .L2X    B5,A18,B5         ; |27| <0,11>  ^
||         MAX2    .L1     A4,A3,A18         ; |29| <0,11>  ^
```

```
   [!B1]   EXT     .S1     A18,16,16,A5      ; |29| <0,12>  ^
|| [!B1]   EXT     .S2     B5,16,16,B9       ; |27| <0,12>  ^
||         MIN2    .L2X    B4,A3,B4          ; |28| <0,12>  ^
|| [!B1]   ADD     .L1     A4,A23:A22,A23:A22 ; |31| <0,12>
||         MVD     .M1     A19,A3            ; |28| <1,6> Split a long life
||         LDH     .D1T1   *++A7(8),A16      ; |30| <2,0>

   [ B1]   SUB     .L2     B1,1,B1           ; <0,13>
|| [!B1]   ADD     .D2X    A17,B6,B6         ; |30| <0,13>
|| [!B1]   ADD     .L1     A3,A21:A20,A21:A20 ; |31| <0,13>
|| [!B1]   EXT     .S2     B4,16,16,B8       ; |28| <0,13>  ^
||         PACK2   .S1     A19,A18,A3        ; |31| <1,7>
||         LDH     .D1T1   *+A7(6),A19       ; |28| <2,1>

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

Можно отметить большее количество тактов в ядре цикла по сравнению с процессором 62 серии. Но в тоже время ядро цикла загружено на 34/48=70,8% для ядра C64, а для ядра C62 эта же программа загружает ядро цикла на 20/24=83,3%.

Name	Calls	Excl Count Average
calc_statistics_no_opt(short *, int, struct int_statistics *)	100	276.83
calc_statistics_opt(short *, int, struct int_statistics *)	100	269.15

Рисунок 5: Запуск оптимизированной программы на процессоре 64 серии

Из рисунка следует, что эталонная программа на 64 процессоре работает быстрее, чем эталонная программа на ядре C62. Это связано с использованием 64 процессором ресурсов более оптимальным способом, и использовании интринсиков min2, max2 при использовании данных типа short автоматически что видно из листинга фидбэка эталонной программы:

Листинг 15: Фидбэк от компилятора для эталонной программы на ядре 64

```
;*--------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file              : ../cacl_static_no_opt.c
;*      Loop source line                : 16
;*      Loop opening brace source line  : 17
;*      Loop closing brace source line  : 24
;*      Known Minimum Trip Count        : 1
;*      Known Max Trip Count Factor     : 1
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound    : 2
;*      Partitioned Resource Bound(*)   : 2
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                    2*       1
;*      .S units                    1        2*
;*      .D units                    1        0
;*      .M units                    1        0
;*      .X cross paths              0        1
```

```
;*      .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        0     (.L or .S or .D unit)
;*      Bound(.L .S .LS)              2*       2*
;*      Bound(.L .S .D .LS .LSD)      2*       1
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 2  Register is live too long
;*         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 -mh6
;*
;*      Minimum safe trip count       : 3
;*----------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP PROLOG

           LDH     .D1T1   *A9++,A8          ; |19| (P) <0,0>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |16| (P) <0,2>

           NOP             1

           ZERO    .L1     A0
||         LDH     .D1T1   *A9++,A8          ; |19| (P) <1,0>
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |16| (P) <1,2>

           MV      .L1     A10,A4
||         MV      .D1     A11,A5
||         MVKH    .S1     0x10000,A0        ; init prolog collapse predicate

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

   [ A0]   MPYSU   .M1     2,A0,A0           ; <0,6>
||         MV      .S1     A8,A3             ; |19| <0,6> Split a long life
||         MAX2    .L1     A16,A8,A7         ; |20| <0,6>  ^
||         MIN2    .L2X    B5,A8,B4          ; |19| <0,6>  ^
|| [ B0]   BDEC    .S2     $C$L4,B0          ; |16| <2,2>
||         LDH     .D1T1   *A9++,A8          ; |19| <3,0>

   [!A0]   ADD     .D1     A3,A17,A17        ; |21| <0,7>
|| [!A0]   EXT     .S2     B4,16,16,B5       ; |19| <0,7>  ^
|| [!A0]   EXT     .S1     A7,16,16,A16      ; |20| <0,7>  ^
|| [!A0]   ADD     .L1     A6,A5:A4,A5:A4    ; |22| <0,7>
||         MPY     .M1     A8,A8,A6          ; |22| <1,5>

$C$DW$L$_calc_statistics_no_opt$8$E:
```

```
;** --------------------------------------------------------------------------*
$C$L5:    ; PIPED LOOP EPILOG

          MV      .S1     A8,A3             ; |19| (E) <1,6> Split a long life
||        MAX2    .L1     A16,A8,A7         ; |20| (E) <1,6>  ^
||        MIN2    .L2X    B5,A8,B4          ; |19| (E) <1,6>  ^

          ADD     .D1     A3,A17,A6         ; |21| (E) <1,7>
||        EXT     .S2     B4,16,16,B4       ; |19| (E) <1,7>  ^
||        EXT     .S1     A7,16,16,A3       ; |20| (E) <1,7>  ^
||        ADD     .L1     A6,A5:A4,A5:A4    ; |22| (E) <1,7>
||        MPY     .M1     A8,A8,A3          ; |22| (E) <2,5>

          MVC     .S2     B6,CSR            ; interrupts on
||        MV      .S1     A8,A7             ; |19| (E) <2,6> Split a long life
||        MAX2    .L1     A3,A8,A9          ; |20| (E) <2,6>  ^
||        MIN2    .L2X    B4,A8,B4          ; |19| (E) <2,6>  ^

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

Можно отметить, что ядро C64 для данных типа short использует встроенные функции для сравнения значений, из-за чего вычисление минимума и максимума уменьшается с 2 до 1 такта.

При этом оптимизированная программа работает медленнее на 37 тактов для обработки массива из 100 элементов, по сравнению с этой же программой на ядре C62.

4.2. Использование альтернативных функций

Используем интринсики min2, max2, pack2 и dotp2, доступные на процессоре 64 и позволяющие осуществить сравнение и запись в переменную нужного значения 1 инструкцией, а также позволяющие упаковать массив. Листинг программы выглядит следующим образом:

Листинг 16: Программа, использующая интринсики

```
struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};

void calc_statistics_opt_2(short *data, int n, struct int_statistics *result)
{
      int i;
      int summ=0;
      int minp=0x7fff7fff;
      int maxp=0x80008000;
      long summ2=0;
      int dotp;
      int one=0x00010001;
      short min, min1, max, max1;
      #pragma MUST_ITERATE(4,  ,4)
      for (i=0; i<n; i+=2)
      {
```

```
            dotp=_pack2(data[i], data[i+1]);
            minp=_min2(minp, dotp);
            maxp=_max2(maxp, dotp);
            summ+=_dotp2(dotp, one);
            summ2+=_dotp2(dotp, dotp);
      }
      min=minp;
      min1=minp>>16;
      min=min<min1 ? min:min1;
      max=maxp;
      max1=maxp>>16;
      max=max>max1 ? max:max1;
      result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
      result->average=summ/n;
      result->min=min;
      result->max=max;;
}
```

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="calc_static_opt_2.pp"
"../calc_static_opt_2.c"
```

В итоге, получаем такой фидбэк от компилятора.

листинг 17: Фидбэк программы, использующей интринсики min2 и max2

```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop found in file               : ../cacl_statistics_opt.c
;*      Loop source line                 : 54
;*      Loop opening brace source line   : 55
;*      Loop closing brace source line   : 61
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 2
;*      Known Max Trip Count Factor      : 2
;*      Loop Carried Dependency Bound(^) : 2
;*      Unpartitioned Resource Bound     : 3
;*      Partitioned Resource Bound(*)    : 3
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     3*       3*
;*      .S units                     1        0
;*      .D units                     3*       1
;*      .M units                     2        2
;*      .X cross paths               2        1
;*      .T address paths             2        2
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           1        1     (.L or .S unit)
;*      Addition ops (.LSD)          1        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             3*       2
;*      Bound(.L .S .D .LS .LSD)     3*       2
;*
;*      Searching for software pipeline schedule at ...
```

```
;*         ii = 3  Schedule found with 5 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      Collapsed epilog stages       : 3
;*
;*      Prolog not entirely removed
;*      Collapsed prolog stages       : 3
;*
;*      Minimum required memory pad   : 24 bytes
;*
;*      Minimum safe trip count       : 1 (after unrolling)
;*----------------------------------------------------------------------------*
$C$L1:    ; PIPED LOOP PROLOG

           ADD     .L1     2,A3,A6
|| [ A0]   BDEC    .S1     $C$L2,A0          ; |54| (P) <0,8>

           MVK     .L2     0x2,B0            ; init prolog collapse predicate
||         ZERO    .L1     A17               ; |56|
||         ZERO    .S2     B18               ; |56|
||         ZERO    .S1     A4                ; |56|
||         MV      .D2X    A3,B4
||         LDH     .D1T1   *A6++(8),A8       ; |59| (P) <0,0>

           ZERO    .L2     B6                ; |56|
||         MVK     .S1     0x4000,A1         ; init prolog collapse predicate
||         MV      .L1     A11,A5
||         MV      .S2X    A11,B19           ; |56|
||         LDH     .D2T1   *B4++(8),A9       ; |59| (P) <0,1>
||         LDH     .D1T2   *-A6(4),B17       ; |57| (P) <0,1>

;** ----------------------------------------------------------------------------*
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_calc_statistics_opt_2$3$B:

   [!A1]   ADD     .L1     A7,A5:A4,A5:A4    ; |60| <0,11>
||         DOTP2   .M1X    A9,B16,A7         ; |59| <1,8>
||         MIN2    .L2X    B7,A9,B8          ; |57| <1,8>  ^
|| [ A0]   BDEC    .S1     $C$L2,A0          ; |54| <1,8>
||         DOTP2   .M2     B9,B16,B5         ; |59| <1,8>
||         LDH     .D1T2   *-A6(6),B8        ; |57| <3,2>

   [ B0]   SUB     .S2     B0,1,B0           ; <0,12>
|| [ A1]   MPYSU   .M1     2,A1,A1           ; <0,12>
|| [!A1]   ADD     .D2     B5,B6,B6          ; |59| <0,12>
|| [!B0]   MIN2    .L2     B8,B9,B7          ; |57| <1,9>  ^
|| [!B0]   MAX2    .L1X    A3,B9,A16         ; |58| <1,9>  ^
||         DOTP2   .M2     B9,B9,B5          ; |60| <1,9>
||         PACK2   .S1     A9,A8,A9          ; |59| <2,6>  ^
||         LDH     .D1T1   *A6++(8),A8       ; |59| <4,0>

   [!A1]   ADD     .S1     A7,A17,A17        ; |59| <0,13>
|| [!A1]   ADD     .L2     B5,B19:B18,B19:B18 ; |60| <0,13>
||         DOTP2   .M1     A9,A9,A7          ; |60| <2,7>
||         MAX2    .L1     A16,A9,A3         ; |58| <2,7>  ^
||         PACK2   .S2     B8,B17,B9         ; |57| <2,7>  ^
||         LDH     .D2T1   *B4++(8),A9       ; |59| <4,1>
```

```
||         LDH     .D1T2   *-A6(4),B17       ; |57| <4,1>

$C$DW$L$_calc_statistics_opt_2$3$E:
;** --------------------------------------------------------------------------*
$C$L3:    ; PIPED LOOP EPILOG
           ADD     .L1     A7,A5:A4,A5:A4    ; |60| (E) <4,11>

           MV      .L1     A4,A10
||         MV      .S1     A5,A11
||         ADD     .L2     B5,B6,B6          ; |59| (E) <4,12>

           ADD     .L1     A7,A17,A17        ; |59| (E) <4,13>
||         ADD     .L2     B5,B19:B18,B19:B18 ; |60| (E) <4,13>

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

В данной программе ядро цикла загружено на 21/24=87,5 %, а в предыдущей программе, где условия не были записаны через интринсики, загруженность ядра составила 11/16=68,75%, что говорит о том, что полученная программа оптимизирована лучше, чем предыдущая на 37,5%, так как упаковывание массива позволяет обрабатывать 4 элемента за выполнение 1 итерации ядра цикла.

Name	Calls	Excl Count Average
calc_statistics_no_opt(short *, int, struct int_statistics *)	100	276.83
calc_statistics_opt(short *, int, struct int_statistics *)	100	269.15
calc_statistics_opt_2(short *, int, struct int_statistics *)	100	157.46

Рисунок 6: Результаты после разбиения цикла на 2

Программа работает быстрее эталона для массива из 100 элементов в 1,76 раз быстрее.

5. Заключение

Разработанный в данной работе алгоритм расчёта статистических параметров рассчитан на массивы, длина которых кратна 4. Если выполнять расчёт параметров для массива, элементы которого составляют арифметическую прогрессию с единичным шагом, первый элемент которой начинается с 32000, то максимальная длина массива составляет 512 элементов, если увеличивать количество элементов прогрессии, то возникает переполнение при расчёте дисперсии и среднего значения. Это связано с тем, что в типе long 1 бит из 40 знаковый, а в типе short 1 из 16 битов знаковый, поэтому в типе long 39 бит используются для записи числа без знака, а в типе short 15 — бит. В итоге максимальная длина массива для того, чтобы расчёт суммы квадратов рассчитывался без переполнений, равна: $\frac{2^{39}}{2^{15}\cdot 2^{15}} = 2^9 = 512$ элементам.

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 инструкция для ядра C64 .

Основные факторы, ограничивающие скорость выполнения алгоритма:

- блоки .D, которые считывают данные малыми порциями
- блоки .L, которые перегружены работой по сравнению значений регистров
- блоки .S, осуществляющие поиск максимумов

Оптимизация алгоритма свелась к развертыванию цикла для обработки за один раз нескольких элементов входных массивов, загрузить ресурсы процессора. В итоге удалось равномерно и практически полностью загрузить все основные блоки процессора: .D, .S и .L, а также для 64 ядра .M юниты, и добиться обработки 4 входного элемента для каждого элемента статистики в каждом 3 такте для 64 процессора и 2 входных элементов для каждого элемента статистики в каждом 3 такте для 62 процессора.

Сравнительный анализ шагов оптимизации для ядра 62x представлен в таблице ниже.

Таблица 1: Сравнение производительности алгоритмов расчёта статистики для C62

Имя функции	Тактов (n=100)	Тактов (n=200)	Прирост на n=100	Тактов на элемент	Произв-сть (n=100)	Произв-сть на элемент
`calc_ctatistics_no_opt`	278,88	479,07	200,19	2	100,00%	100%
`calc_ctatistics_opt`	266,43	466,43	200	2	104,67%	100,09%
`calc_ctatistics_opt_2`	266,45	466,45	200	2	104,67%	100,09%
`calc_ctatistics_opt_3`	221,47	371,47	150	1,5	125,92%	133%

Как видно из таблицы, последний вариант функции дает прирост производительности, около 1,26 раз при n=100, и удельный прирост в 1,33 раза по сравнению с эталонным алгоритмом. Не удалось добиться большего ускорения программы из-за высокой загруженности (в 83%) ядра цикла конвейера в эталонной программе.

Сравнительный анализ шагов оптимизации для ядра 64x представлен в таблице ниже.

Таблица 2: Сравнение производительности алгоритмов расчёта статистики для C64

Имя функции	Тактов (n=100)	Тактов (n=200)	Прирост на n=100	Тактов на элемент	Произв-сть (n=100)	Произв-сть на элемент
`calc_ctatistics_no_opt`	276,82	477,01	200,19	2	100,00%	100%
`calc_ctatistics_opt`	269,15	419,15	150	1,5	102,85%	133,33%
`calc_ctatistics_opt_2`	157,46	232,46	75	0,75	175,8%	266,66%

Можно сказать, что на 64 ядре удалось добиться большей производительности по сравнению с полученной производительностью на 62 ядре, это связано с тем что компилятор создал более плотное расписание команд параллельного выполнения: MIN2, MAX2, DOTP, которых стало меньше и удалось их уплотнить в меньший набор FETCH пакетов.

5.1. Общие вопросы по заданию

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

Как видно из таблиц 1 и 2, удалось добиться ускорения в 1,26 раз по сравнению с эталоном для архитектуры C62x и в 1,76 раза на 64 ядре, что соответствует удельному росту производительности в 1,33 раза на один элемент входных данных (длину вектора) для 62 ядра и 2,7 раз для 64 ядра.

5.1.2. Зависит ли ускорение от архитектуры ядра?

Да, в алгоритме расчёта статистических параметров основное ограничение производительности связано с ограниченным количеством .L, .S и .D юнитов и возможностями 62 ядра для расчёта минимального и максимального значения, а также возможностями упаковки массива. В ядрах архитектуры C64x, C64x+ присутствуют встроенные функции, позволяющие ускорить расчёт. Кроме того, там вдвое больше умножителей, что позволяет проводить расчеты компонентов скалярного произведения еще вдвое быстрее.

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

Для приведенного алгоритма необходимо наложить следующие ограничения.

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

5.2. Дополнительные вопросы по заданию

5.2.1. Что эффективнее, считать все параметры статистики в одном цикле, или разбивать на несколько?

Из полученных результатов следует, что считать статистические параметры лучше в одном цикле, так как разбивание расчёта на несколько циклов уменьшает загруженность ядра цикла, при этом не удаётся уплотнить массив исходных данных, или найти алгоритм для вычисления отдельных параметров статистики, которые бы работали быстрее полученных.

Список использованной литературы

1. *TMS320C6000 Optimizing C Compiler Tutorial* // SPRU425
2. *Hand-Tuning Loops and Control Code on the TMS320C6000* // SPRA666
3. *TMS320C6000 Optimizing Compiler User's Guide* // SPRUI04C

Приложение. Исходные тексты программ

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

Листинг 18: main.c

```c
#include <stdio.h>

short A[512];

struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};


void calc_statistics_no_opt(short *data, int n, struct int_statistics *result);
void calc_statistics_opt_2(short *data, int n, struct int_statistics *result);
void calc_statistics_opt_3( short * data,int n, struct int_statistics *result);
void calc_statistics_opt_4( short * data,int n, struct int_statistics *result);
void calc_statistics_opt(short * data,int n, struct int_statistics * result);


void print_statictics (struct int_statistics disp, char *prefix)
{
   printf("%s: \n minimum: %d \n maximum: %d \n ",prefix, disp.min, disp.max);
   printf("average: %d \n variance: %d \n", disp.average, disp.variance);
}


int main(void)
{
   struct int_statistics p;

   int n=sizeof(A)/sizeof(A[0]);

   int i;
   for (i=0; i<n; i++)
   {
      A[i]=i+32000;
   }
   for (i=0; i<100; i++){
      calc_statistics_no_opt(A , n, &p);
   }
   print_statictics(p, "No opt");
   for (i=0; i<100; i++){
      calc_statistics_opt(A ,n , &p);
   }
   print_statictics(p, "Opt");
   for (i=0; i<100; i++){
      calc_statistics_opt_2(A ,n , &p);
   }
   print_statictics(p, "Opt 2");
```

```
#if defined(_TMS320C6200)
   for (i=0; i<100; i++){
      calc_statistics_opt_3(A ,n , &p);
   }
   print_statictics(p, "Opt 3");
   for (i=0; i<100; i++){
      calc_statistics_opt_4(A ,n , &p);
   }
   print_statictics(p, "Opt 4");

#endif
   return 0;
}
```

Листинг 19: calc_statistics_no_opt.c

```
// Эталонная программа вычистения статистических параметров
struct int_statistics {
   short min;
   short max;
   int average;
   int variance;
};

void calc_statistics_no_opt(short *data, int n, struct int_statistics *result)
{
         int i;
         short min=32767;
         short max=-32768;
         int summ=0;
         long summ2=0;
         int x;
         for (i=0; i<n; i++)
         {
            x=data[i];
            if(x<min) min=x;
            if(x>max) max=x;
            summ+=x;
            summ2+=x*x;

         }
         // дисперисия расчитывается как разность суммы квадратов и квадрата суммы
         // всех элементов массива, деленное на количество элементов.
         // полученная разность делится на количество элементов массива
         result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
         result->average=summ/n;
         result->min=min;
         result->max=max;
}
```

Листинг 20: calc_statictics_opt.c

```
// Оптимизированная программа статистических параметров
struct int_statistics {
   short min;
   short max;
```

```
   int average;
   int variance;
};

long long  summ2sa(short*,int);

#if defined(_TMS320C6400)


void calc_statistics_opt(short * data,int n, struct int_statistics *result)
{
        int i;
        short min=32767;
        short max=-32768;
        int summ=0;
        short max1=-32768;
        short min1=32767;
        long summ2=0;
        short x, y;
        #pragma MUST_ITERATE(4,  ,4)
        for (i=0; i<n; i+=2)
        {
           x=data[i];
           y=data[i+1];
           min=x<min ? x:min;
           max=x>max ? x:max;
           min1=y<min1 ? y:min1;
           max1=y>max1 ? y:max1;
           summ+=x+y;
           summ2+=x*x+y*y;
        }
        min=min<min1 ? min:min1;
        max=max>max1 ? max:max1;
        result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
        result->average=summ/n;
        result->min=min;
        result->max=max;
}


void calc_statistics_opt_2(short *data, int n, struct int_statistics *result)
{
        int i;
        int summ=0;
        int minp=0x7fff7fff;
        int maxp=0x80008000;
        long summ2=0;
        int dotp;
        int one=0x00010001;
        short min, min1, max, max1;
        #pragma MUST_ITERATE(4,  ,4)
        for (i=0; i<n; i+=2)
        {
           dotp=_pack2(data[i], data[i+1]);
           minp=_min2(minp, dotp);
           maxp=_max2(maxp, dotp);
           summ+=_dotp2(dotp, one);
           summ2+=_dotp2(dotp, dotp);
```

```
        }
        min=minp;
        min1=minp>>16;
        min=min<min1 ? min:min1;
        max=maxp;
        max1=maxp>>16;
        max=max>max1 ? max:max1;
        result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
        result->average=summ/n;
        result->min=min;
        result->max=max;;
}


#endif

#if defined(_TMS320C6200)


void calc_statistics_opt(short * data,int n, struct int_statistics * result)
{
        int i;
        short min=32767;
        short max=-32768;
        int summ=0;
        long summ2=0;
        short x;
        #pragma MUST_ITERATE(4,  ,4)
        for (i=0; i<n; i++)
        {
           x=data[i];
           if(x<min) min=x;
           if(x>max) max=x;
           summ+=x;
           summ2+=x*x;
        }
        result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
        result->average=summ/n;
        result->min=min;
        result->max=max;
}


void calc_statistics_opt_2(short *data, int n, struct int_statistics *result)
{
        int i;
        short min=32767;
        short max=-32768;
        int summ=0;
        long summ2=0;
        short x;
        #pragma MUST_ITERATE(4,  ,4)
        for (i=0; i<n; i++)
        {
           x =   data[i];
           min=x<min ? x:min;
           max=x>max ? x:max;
           summ+=x;
```

```
            summ2+=x*x;
         }
         result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
         result->average=summ/n;
         result->min=min;
         result->max=max;
}


void calc_statistics_opt_3(const short *data, int n, struct int_statistics *result)
{
         int i;
         short min=32767;
         short max=-32768;
         int summ=0;
         short max1=-32768;
         short min1=32767;
         long summ2=0;
         short x;
         short y;
         #pragma MUST_ITERATE(2,  ,2)
         for (i=0; i<n; i+=2)
         {
            x =   data[i];
            y=data[i+1];
            min=x<min ? x:min;
            max=x>max ? x:max;
            min1=y<min1 ? y:min1;
            max1=y>max1 ? y:max1;
            summ+=x+y;
            summ2+=x*x+y*y;
         }
         min=min<min1 ? min:min1;
         max=max>max1 ? max:max1;
         result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
         result->average=summ/n;
         result->min=min;
         result->max=max;
}



void calc_statistics_opt_4(short * data,int n, struct int_statistics *result)
{
         int i;
         short min=32767;
         short max=-32768;
         int summ=0;
         short max1=32767;
         short min1=-32768;
         long summ2=0;
         int x;
         int y;
         #pragma MUST_ITERATE(4,  ,4)
         for (i=0; i<n; i+=2)
         {
            x=data[i];
            y=data[i+1];
```

```
            min=x<min ? x:min;
            max=x>max ? x:max;
            min1=y<min1 ? y:min1;
            max1=y>max1 ? y:max1;
            summ+=x+y;
        }
        summ2=summ2sa(data, n);
        min=min<min1 ? min:min1;
        max=max>max1 ? max:max1;
        result->variance=(summ2-(long long)summ*(long long)summ/n)/(n);
        result->average=summ/n;
        result->min=min;
        result->max=max;
}

#endif
```

Листинг 21: summ2sa.sa

```
; программа для вычисления суммы квадратов
      .global    _summ2sa

_summ2sa:        .cproc          data, count
                 .reg            x, mult, result:result1

                 zero            result:result1
loop:
                 ldh             *data++, x
                 mpy             x, x, mult
                 add             mult, result:result1, result:result1
                 sub             count, 1, count
[count]          b               loop
                 .return         result:result1
                 .endproc
```

Вывод программы при запуске на 62 ядре:

```
No opt:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
Opt:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
Opt 2:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
Opt 3:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
```

```
Opt 4:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
```

Вывод программы при запуске на 64 ядре:

```
No opt:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
Opt:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
Opt 2:
 minimum: 32000
 maximum: 32511
 average: 32255
 variance: 21845
```

КОНЕЦ ДОКУМЕНТА