УДК 004.428.4

Ускорение вычислений с использованием высокопроизводительных математических и мультимедийных библиотек для архитектуры Эльбрус.

П.А. Ишин, В.Е. Логинов, П.П. Васильев.

Приведено краткое описание высокопроизводительной мультимедийной и математической библиотеки EML. Описаны некоторые способы оптимизации функций и полученные результаты производительности.

Ключевые слова: Эльбрус, архитектура, оптимизация, библиотека

Введение.

Все производители современных процессоров разрабатывают и поставляют пользователям высокопроизводительные библиотеки, обеспечивающие скорость работы, близкую к максимальной для данного типа процессоров. Примерами таких библиотек являются библиотеки IPP/MKL фирмы Intel, библиотеки mediaLib/Perflib фирмы Oracle, библиотеки ACML/APL фирмы AMD.

Для процессоров линии Эльбрус так же была разработана библиотека EML – высокопроизводительная математическая и мультимедийная библиотека, представляющая из себя набор разнообразных функций для обработки сигналов, изображений, видео, а так же широкий набор математических функций и операций, которые часто используются в вычислениях. При этом библиотека является высокопроизводительной, что означает высокую скорость её работы. В статье приведено описание библиотеки, а так же описаны методы оптимизации функций под архитектуру Эльбрус-2с+.

1. Структура библиотеки

Основные разделы библиотеки EML:

- Ядро (Core) – выделение и освобождение памяти, номер версии и статус.
- Вектор (Vector) – различные операции над векторами: арифметические, логические, преобразование типов, математические функции, статистика.
- Сигналы (Signal) – цифровая обработка сигналов: конволюция, фильтрация, усиление, генерация, быстрые преобразования Фурье и Хартли.

- Изображение (Image) – создание и заполнение изображений, арифметические операции, фильтрация, геометрические и цветовые преобразования, ДПФ.
- Линейная Алгебра (Algebra) – стандартные пакеты работы с матрицами и векторами BLAS1/2/3, LAPACK.
- Видео (Video) - обработка видео: интерполяция, усреднение, оценка движения, цветовые преобразования, ДКП, квантизация.
- Графика (Graphics) - рисование/закрашивание точек/линий/треугольников/ прямоугольников/полигонов/дуг/окружностей/эллипсов, закрашивание/перекрашивание области. Режимы: со сглаживанием, смешиванием по альфа, затенением по Гуро, с буфером глубины, исключающее ИЛИ
- Объем (Volume) - бросание параллельных/произвольных лучей с интерполяцией, Линейное масштабирование вокселей, Поиск максимальных значений на луче

Текущее состояние библиотеки:

Раздел библиотеки	Core	Vector	Signal	Image	Algebra	Video	Graphics	Volume	Всего
Количество функций	5	374	153	98	306	111	246	25	1318

На данный момент разработаны версии библиотеки под следующие платформы:

lnux-x86 –Си версия для Intel x86

msvs-e3m(e3s)/msvs-e3m64(e3s64) – 32/64 битная версия для Эльбрус (Эльбрус-2с+) [1]

msvs-e3m128(e3s128) – защищенный режим для Эльбрус (Эльбрус-2с+)

linux-e90 - 32 битная версия для МЦСТ-R500S [2]

linux-e90v9/linux-e90v964 – 32/64 битная версия для МЦСТ R1000 [3]

2. Типы данных библиотеки

Библиотека eml работает с 8, 16, 32, 64 - разрядными целыми (знаковыми и беззнаковыми), плавающими типами одинарной и двойной точности, а так же с комплексными типами. Комплексный тип является структурой из двух элементов базового типа. Например: eml_8u - 8 разрядное беззнаковое целое (unsigned char), eml_32s

- 32 разрядное знаковое целое(int). Для комплексных типов в конце добавляется “c”, например: eml_16sc, eml_32fc

3. Пример оптимизации функции

В архитектуре Эльбрус, для большинства типов данных предусмотрены специальные операции над упакованными данными. При этом не всегда получается адаптировать алгоритм функции для работы с упакованными данными с помощью языка Си, так как компилятор не всегда справляется со сложными шаблонами. Поэтому приходится использовать специальный набор псевдо функций, которые на самом деле представляют собой ассемблерную вставку, задающий конкретную специфическую операцию над упакованными данными. В результате программа пишется не на ассемблере, а на языке Си, что ускоряет разработку и отладку. Далее приведены примеры оптимизации функций (для случая, когда вектор выровнен на 8 байт).

Пример 1

Нахождение индекса максимального элемента в векторе с плавающей точкой одинарной точности.

eml_Vector_MaxIndex_32F (const eml_32f * pSrc, eml_32s len, eml_32s * pMaxIndex)

```
{
    eml_64u *sp = (eml_64u *) pSrc;
    eml_64u max1, maxind1, indt, inc = 0x0000000200000002LL;
    eml_32s i;

    max1 = sp[0];
    maxind1 = 0x0000000100000000LL;
    indt = 0x0000000300000002LL;

    for (i = 0; i < (len >> 1); i ++) {
        eml_64u tmpmax = e3m_pfmaxs (max1, sp[i]);
        eml_64u mask = e3m_pfcmpeqs (tmpmax, max1);
        maxind1 = (maxind1 & mask) | (indt & ~mask);
        max1 = tmpmax;
        indt += inc;
    }
/* обработка последнего элемента */
.........
/* получение итогового результата */
.........
}
```

Используемые команды архитектуры Эльбрус-2c+: *pfmax* - максимум 2 флотов, *pfcmpeqs* - сравнение 2-х флотов на равно с формированием битовой маски. В данном цикле мы

находим максимальный четный и нечетный элементы, путем сравнения сразу двух элементов. По полученной маске мы запоминаем индексы максимальных элементов. Затем, при получении итогового результата, мы сравниваем полученные максимумы четных и нечетных элементов и возвращаем индекс результата.

Четыре итерации полученного цикла компилятор планирует в три такта широкой команды [4]. При этом одна итерация обрабатывает сразу 2 элемента, в итоге теоретическая производительность составляет 0,38 такта на элемент. При реализации обычной Си версии компилятор планирует одну итерацию (обрабатывающую один элемент) в один такт.

Производительность для вектора длины 1024 на машине Эльбрус-2с+ (тактов на элемент цикла): обычный Си код *1.12*, оптимизированный вариант 0.*51*, *у*лучшение на *119%*.

Пример 2

Вычисление комплексно-сопряженного 16 разрядного целого знакового вектора

eml_Vector_Conj_16SC (const eml_16sc * pSrc, eml_16sc * pDst, eml_32s len)

```
{
    eml_64u *sp, *dp, src, mask1 = 0xffff0000ffff0000, mask2 = 0x1000000010000;
    eml_32s i;

    sp = (eml_64u *) pSrc;
    dp = (eml_64u *) pDst;
    for (i = 0; i < ((len >> 1); i++) {
        dp[i] = e3m_paddsh (sp[i] ^ mask1, mask2);
    }
}
```

Используемые команды архитектуры Эльбрус-2с+: *paddsh* – сложение 4 знаковых шортов.

В данном цикле мы вначале с помощью команды “^” pDst[i].im заменяем на “–pDst[i].im – 1”, затем прибавляем 1. Действительная часть при этом не меняется.

В итоге получаем 3 операций на обработку сразу 2 элементов или 1 такт на 2 итерации цикла (на 2 элемента), в итоге теоретическая производительность составляет 0,25 тактов на элемент. При реализации обычной Си версии компилятор планирует одну итерацию в один такт.

Производительность для вектора длины 1024 на машине Эльбрус-2с+ (тактов на элемент цикла): обычный Си код 1.*08*, оптимизированный вариант *0.34, у*лучшение на *218%*.

4. Работа с не выравненными адресами

При оптимизации функций мы стараемся считывать элементы вектора сразу по 8 байт, соответственно вектора, которые мы используем, должны быть выравнены на 8 байт. Но пользователь может в функцию подать вектор, выравненный на размер элемента. Например, если подается вектор с элементами типа *short*, то вектор может быть выровнен только на 2 байта. В таком случае, мы производим выравнивание вектора внутри нашей функции. Разберем это на примере функции вычисление абсолютной величины элементов 16 разрядного целого знакового вектора

eml_Vector_Abs_16S (const eml_16s * pSrc, eml_16s * pDst, eml_32s len)

Вначале мы вычисляем количество байт, на которые вектора не выровнены и определяем адреса, с которых начинаем работать с вектором.

```
{
    eml_64s *sp1, *dp;
    eml_64s diff, align1, src1, dst;

    eml_32s i;
/* вычисляем количество байт до выровненности на 8 от pDst */
    diff = ((-(eml_addr)(pDst))&(7));
/* вычисляем следующий выравненный на 8 адрес от pDst */
    dp = (void*)((eml_8u*)(pDst) + (((eml_addr)(pDst))&(7));
    diff >>= 1;
    len -= diff;
/* вычисляем количество байт от выровненного адреса до (pSrc + diff) */
    align1 = (((eml_addr)(pSrc + diff)) & 7);
/* вычисляем предыдущий выравненный на 8 адрес до (pSrc + diff) */
    sp1 = (void*)((eml_8u*)(pSrc + diff) - ((((eml_addr)(pSrc + diff)) & 7));

/* обрабатываем начало вектора до выровненного dst адреса */
    for (i = 0; i < diff; i++) {
        pDst[i] = abs (pSrc[i]);
    }
```

Далее мы или выполняем цикл для не выравненного случая:

```
        align1 <<= 9;
        for (i = 0; i < (len >> 2); i++) {
/* делаем склейку из двух соседних 8 битовых элементов */
            src1 = e3m_scrd (e3m_insfd (sp1[i], align1, sp1[i + 1]), align1 >> 6)
/* вычисляем абсолютную величину 4 шортов */
            eml_64s tmp = e3m_psrah (src1, 15);
            dst = src1 ^ tmp;
            dp[i] = e3m_psubh (dst, tmp);
        }
```

Если же у нас выравненные адреса, то цикл выглядит следующим образом:

```
for (i = 0; i < (len >> 2); i++) {
    src1 = sp1[i];

    eml_64s tmp = e3m_psrah (src1, 15);
    dst = src1 ^ tmp;
    dp[i] = e3m_psubh (dst, tmp);
}
```

Используемые команды архитектуры Эльбрус-2с+: *insfd* - циклически сдвигает вправо операнд1, вставляет произвольное количество самых правых разрядов операнда 3 в самые правые разряды циклически сдвинутого операнда1, *scrd* - циклический сдвиг вправо, psrah - арифметический сдвиг вправо 4-х знаковых шортов, *psubh* - вычитание 4-х шортов.

В не выравненном цикле мы с помощью команд *scrd* и *insfd* из двух соседних двойных слов склеиваем одно и далее производим вычисление абсолютного значения сразу 4 элементов. Компилятор планирует 3 итерации выравненного цикла в 2 такта (теоретическая производительность 0.17), а 1 итерацию не выравненного случая в один такт (теоретическая производительность 0.25). Производительность для вектора длины 1024 на машине Эльбрус-2с+ (тактов на элемент цикла): оптимизированный выравненный вариант *0.27*, оптимизированный не выравненный вариант *0.35* .

Как видим, не выравненный случай медленнее выравненного случая на 29% .

Можно заметить, что у нас есть некоторое расхождение теоретической и реальной производительности. Это объясняется наличием больших накладных расходов, при этом, чем длиннее цикл, тем это расхождение меньше.

Заключение

В подавляющем большинстве случаев, скорость работы функций получается близкой к теоретически возможной. Это достигается благодаря следующим факторам:

1. Выбор алгоритма оптимизации, наиболее подходящего для архитектуры.

2. Подбор оптимального для архитектуры набора операций для выполнения алгоритма.

3. Выбор наилучшего анролла.

4. Оптимизация обращений к памяти.

5. Оптимальное планирование кода.

6. Подбор наилучших опций компилятора.

Сравнительная таблица результатов применения оптимизаций для Эльбрус-2с+, по сравнению с обычной Си версией, собранной для данной платформы:

Раздел библиотеки	Максимальное ускорение	Среднее ускорение	Количество тестов
Vector	50.19	2.16	1234
Signal	42.41	3.00	2064
Algebra	99.34	1.88	5928
Video	10.56	2.37	422

Большое ускорение работы библиотеки в разделе Signal, достигается не только благодаря использованию упакованных операций, но и благодаря изменению алгоритмов функций (в частности БПФ) под особенности архитектуры [5].

Как видим, высокопроизводительная математическая и мультимедийная библиотека архитектуры Эльбрус позволяет пользователю не только упростить работу по написанию своих приложений, но так же ускорить их работу в несколько раз.

Литература

1. Система на кристалле Эльбрус-2с+. http://www.mcst.ru/elbrus_2c_111101.shtml

2. Система на кристалле МЦСТ R-500S. http://www.mcst.ru/b_18-19.shtml

3. Микросхема МЦСТ R1000. http://www.mcst.ru/system-4x.shtml

4. Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». СПб., Питер, 2013.

5. В.Е.Логинов, П.А.Ишин "Оптимизация для архитектуры “Эльбрус” быстрого преобразования Фурье применительно к 32-разрядным числам с плавающей точкой". Вопросы радиоэлектроники серия ЭВТ, выпуск 3, 2012