АРХИТЕКТУРНАЯ И КОМПИЛЯТОРНАЯ ОПТИМИЗАЦИИ ВЫЧИСЛЕНИЯ БПФ НА МНОГОЯДЕРНОМ ПРОЦЕССОРЕ

Черемнов А.Г., Аврамчук В.С.
Научный руководитель: Аврамчук В.С., к.т.н., доцент
Томский политехнический университет, 634050, Россия, г. Томск, пр. Ленина, 30
Email: 8xandr@gmail.ru

Эффективность вычисления БПФ зависит не только от мощностей используемого оборудования, но и от качества программного кода.

Хороший прирост производительности можно получить, используя компилятор от компании Intel, а также при помощи низкоуровневого взаимодействия с процессором ЭВМ на уровне инструкций (язык ассемблер).

К скалярным оптимизациям относят свертку констант, протяжку констант и протяжку копий [1].

Под сверткой констант понимается процесс вычисления констант при непосредственно самой компиляции приложения, под протяжкой констант – подстановка величин известных констант в выражение, а под протяжкой копий – процесс замены переменных их значениями [2]. Пример исходного кода протяжки констант и протяжки копий на языке C++ приведён ниже:

```
int x = 14;
int y = 7 - x / 2;
=> ПРОТЯЖКА КОНСТАНТ=>
int x = 14;
int y = 7 – 14 / 2;
y = x;
z = 3 + y
=> ПРОТЯЖКА КОПИИ =>
z = 3 + x
```

Удаление повторных вычислений также является скалярной оптимизацией.

Большинство скалярных, межпроцедурных локальных и глобальных оптимизаций компилятор Intel способен проводить самостоятельно, если, например, использовать при сборке специальный ключ /O3 [3]. Большинство оптимизаций циклических конструкций с целью уменьшения времени выполнения кода компилятор также проводит самостоятельно.

В данной работе использована среда программирования Microsoft Visual Studio 2012 Proffessional и кроссплатформенная библиотека Intel TBB в качестве инструмента параллельной разработки.

Для реализации параллельного вычисления БПФ использован алгоритм Кули-Тьюки [4], который обладает явным рекурсивным свойством. Пример вычисления БПФ для последовательности из 8-ми элементов приведён на рисунке 1[5].

Первая архитектурная оптимизация используемая в этой работе заключается в упаковке входных значений в векторные регистры xmm0-xmm15 при вычислении БПФ двух точек.

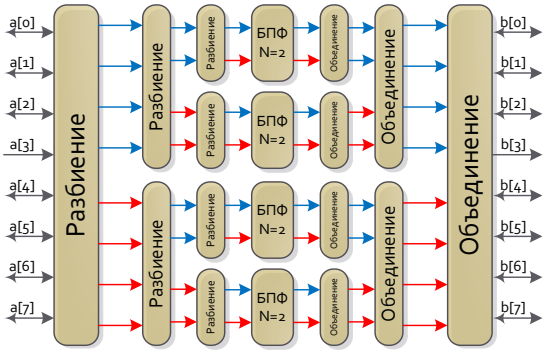

Рис. 1, Разбиение и объединение последовательности при N=8.

БПФ для двух точек вычисляется в соответствии с формулой для ДПФ [6]:

$$b_k = \sum_{i=0}^{N-1}\left(a_i \cdot e^{-j\cdot\frac{2\pi}{N}\cdot i\cdot k}\right),\ k = 0,\ldots,N-1,$$

где N=2 – размер выборки сигнала, a_i - мгновенные значения сигнала, b_k – значения коэффициентов ряда Фурье. Применяя формулу Эйлера [7], можно свести экспоненциальный вид к вычислению значений синуса и косинуса, разбивая комплексное представление на действительные и мнимые части.

Тригонометрические операции над векторными регистрами проводить нельзя, допустимы только операции сложения, вычитания, умножения и деления [1], поэтому далее раскладываем гармонические функции в ряд Тейлора, для того, чтобы можно было со всеми входными значениями оперировать как с одним вектором.

Для проведения упаковки данных в векторные массивы существует две ассемблерные инструкции [1]:

- MOVDQA—Move Aligned Double Quadword;
- MOVDQU—Move Unaligned Double Quadword.

Первая инструкция позволяет упаковать данные за 1 такт процессорного времени, вторая занимает значительно больше времени. Поэтому если заранее выровнять данные в оперативной памяти ЭВМ на 16 бит, то возможно получить дополнительный прирост за счет быстрой упаковки значений в вектора, а затем после вычислительной операции – быстрой распаковкой вектора.

Пример упаковки данных на языке ассемблер выглядит следующим образом:

```
movdqa  XMMWORD PTR [edx+ecx*4], xmm1
```

edx – указатель на начало части массива, ecx – индекс текущего копируемого элемента.

Распаковка производится аналогичным

способом.

Второй архитектурной оптимизацией является ручная загрузка данных из относительно медленной оперативной памяти в кэш до того, как эта память непосредственно потребуется процессору.

Функция предвыборки определена в xmmintrin.h и имеет форму[1]:

```
#include <xmmintrin.h>
enum _mm_hint { _MM_HINT_T0 = 3,    (L1)
                _MM_HINT_T1 = 2,    (L2)
                _MM_HINT_T2 = 1,    (L3)
                _MM_HINT_NTA = 0 };
void _mm_prefetch(void *p, enum _mm_hint h);
```

Эта функция подгружает в кэш кэш-линию, начиная с указанного адреса (размер кэш линии 64 байта). Загрузка последующих 4-х значений в кеш 1-го уровня позволяет повысить производительность.

Третьей проведённой архитектурной оптимизацией является уменьшение числа ветвлений, преимущественно в больших циклах, так как большинство современных ЦП пытаются предсказать результат условия (branch prediction). Однако, в общем случае, предсказание ветвления будет ошибочно в 50% случаев. Такие ошибки предсказания приводят к замедлению выполнения алгоритма [5].

Также отметим, что уменьшение количества локальных переменных позволяет компилятору хранить их в регистрах, а не в стеке. Таким образом, объявление переменных непосредственно в процедуре перед использованием позволяет минимизировать количество переменных [2].

Для определения прироста производительности вычислений БПФ с использованием приведенных выше оптимизации был проведён расчёт частотно-временной корреляционной функции [8] для ряда тестовых примеров. Размер выборки варьировался в пределах 8–131072 отсчётов. Входной массив данных представлен комплексными числами. Экспериментальные исследования проведены на трех процессорах фирмы Intel: Core 2 Quad 6700, Xeon® 5160, Core i5-750 и на двух процессорах фирмы AMD: A10-4600M и FX-9590.

В качестве примера, в таблице, приведены временные результаты вычисления БПФ на ЦП AMD FX-9590.

На рисунке 2 представлено отношение времени выполнения реализации алгоритма БПФ без использования оптимизации (t_1) ко времени выполнения реализации алгоритма БПФ с использованием оптимизаций (t_2).

Из рисунка 2 видно, что наибольший прирост производительности имеют процессоры фирмы Intel.

Таблица. Результаты вычисления частотно-временной корреляционной функции (1000 БПФ).

Размер выборки	Время выполнения без оптимизации t1, с	Время выполнения с оптимизациями t2, с
8	0,0005478±0,0000019	0,0003898±0,0000134
16	0,0014117±0,0000068	0,0009326±0,0000286
32	0,0035165±0,0000056	0,0022450±0,0000640
64	0,0086691±0,0000149	0,0054124±0,0001435
128	0,0204394±0,0000245	0,0122061±0,0003222
256	0,0463984±0,0000893	0,0276179±0,0007573
512	0,1051737±0,0000932	0,0630359±0,0019294
1024	0,1484914±0,0001789	0,0978883±0,0033885
2048	0,2379425±0,0003088	0,1471969±0,0072787
4096	0,4388310±0,0012881	0,3039140±0,0103739
8192	0,9493237±0,0012458	0,6133272±0,0184973
16384	1,9102854±0,0024361	1,2074764±0,0431499
32768	3,7159933±0,0068308	2,4440258±0,0567452
65536	7,7787342±0,0134074	5,0421034±0,1267075
131072	17,7679176±0,087885	10,1004592±0,2078006

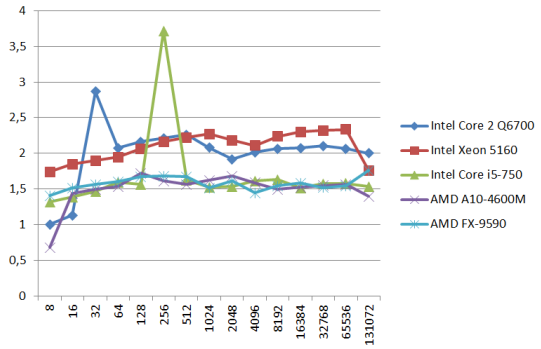

Рис. 2. Зависимость отношения t_1/t_2 от размера выборки

Список литературы

1. Intel® 64 and IA-32 Intel Architecture Software Developer's Manual. URL: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html (Дата обращения: 17.02.2014)
2. Agner Fog. An optimization guide for Windows, Linux and Mac platforms. - Copenhagen University College of Engineering - 2012 - 168 p.
3. Intel® 64 and IA-32 Architectures Optimization Reference Manual. URL: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html (Дата обращения: 17.02.2014)
4. Cooley, James W.; Tukey, John W. "An algorithm for the machine calculation of complex Fourier series". *Math. Comput.* **19**: 297–301
5. Теория и практика цифровой обработки сигналов. URL: http://www.dsplib.ru/index.html (Дата обращения: 17.02.2013)
6. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. – М.: Мир, 1989. – 448 с.
7. М. Я. Выгодский. Справочник по высшей математике. – М.: Наука, 1977. – 870 с.
8. Лунева Е.Е., Аврамчук В.С. Анализ путей повышения эффективности расчетов частотно-временной корреляционной функции // Известия Томского политехнического университета. – 2013. – Т. 322 – № 5. – С. 33–36.