Инновационная компания Promwad

Методы профилирования и оптимизация кода для процессоров семейства Blackfin

Игорь Лекторов

Структура доклада

- Средства ***GCC*** и библиотеки оптимизации
  - стандартные опции компилятора
  - применение встроенных функций компилятора
  - использование кэша ***L1*** процессора ***Blackfin***
  - библиотека ***libbfdsp***
- Измерение производительности и профилирование
  - применение ***Valgrind***
  - аннотация кода с помощью ***Gcov***
  - утилита ***Oprofile***
- Пример оптимизации кода библиотеки ***Ffmpeg***
  - оптимизация кода с помощью профилирования и использования возможностей компилятора ***Blackfin***

Основные термины

- **Профилирование** – сбор характеристик работы программы, таких как время выполнения отдельных фрагментов, число верно предсказанных условных переходов, число кэш промахов и т. д.
- **Профилировщик** – инструмент, используемый для анализа работы программы

Оптимизация с помощью средств GCC и доступных библиотек

Применение стандартных опции компилятора

- ***-O0*** – отсутствие оптимизации
- ***-O1*** – уменьшение размера кода без оптимизации
- ***-O2*** – оптимизация за счет увеличения длины кода
- ***-O3*** – то же что и ***-O2***, включая разворачивание циклов и делая функции *inline*
- ***-Os*** – оптимизация по размеру
- ***-ffast-math*** – эмуляция операций с плавающей точкой

2009 © Инновационная компания Promwad

Оптимизация с помощью средств GCC и доступных библиотек

➲ Применение опции компилятора для теста ***whetstone***

Опции компилятора	Размер, байт	Время, с
-O0	33464	549
-O1	22576	411
-Os	22400	385
-O2	22552	381
-O2 -ffast-math	19520	314
-O0 -mfast-fp	31088	190
-O2 -mfast-fp	20176	156
-O1 -ffast-math -mfast-fp	17064	131
-O3	25128	118
-O3 -ffast-math	21072	50
-O3 -ffast-math -mfast-fp	18664	14

Оптимизация с помощью средств GCC и доступных библиотек

Применение встроенных функций

- функции для работы с целыми числами
- функции для работы с комплексными числами
- операции перестановки байтов в слове
- операции умножения младших частей слова на старшие

Не реализованы возможности использования циклических буферов и записи в соответствии с обратным двоичным порядком, специфические для БПФ

Средства GCC и библиотеки для оптимизации

- Использование кэша ***L1***

  Преимущества: повышение производительности за счет большей скорости доступа к памяти.

- Размещение приложения в ***L1***

  CFLAGS += -fno-jump-tables
  LDFLAGS += -pie -Wl,--sep-code -Wl,--code-in-l1 -Wl,-z,now

- Размещение данных в ***L1***

  CFLAGS += -Wl,--data-in-l1

- Размещение отдельных функций и переменных в ***L1***

```
void foo() __attribute__ ((l1_text))
int var __attribute__ ((l1_data_A))
CFLAGS += -fno-jump-tables
```

- Динамическое выделение памяти в ***SRAM***

```
void *sram_alloc (size_t size, unsigned long flags)
int sram_free (void *addr)
void *dma_memcpy (void *dest, const void *src, size_t size)
```


2009 © Инновационная компания Promwad

Средства GCC и библиотеки для оптимизации

Библиотека ***libbfdsp***

Частично портированная с ***VisualDSP++*** библиотека, содержащая набор функций для цифровой обработки сигналов:

- БПФ
- Операции свертки
- Комплексные умножения и сложения векторов
- Расчет фильтров с КИХ и БИХ

2009 © Инновационная компания Promwad

Измерение производительности и профилирование

Valgrind

Профилировщик для **PC** имеющий общую инфраструктуру, позволяющий использовать различные инструменты профилирования

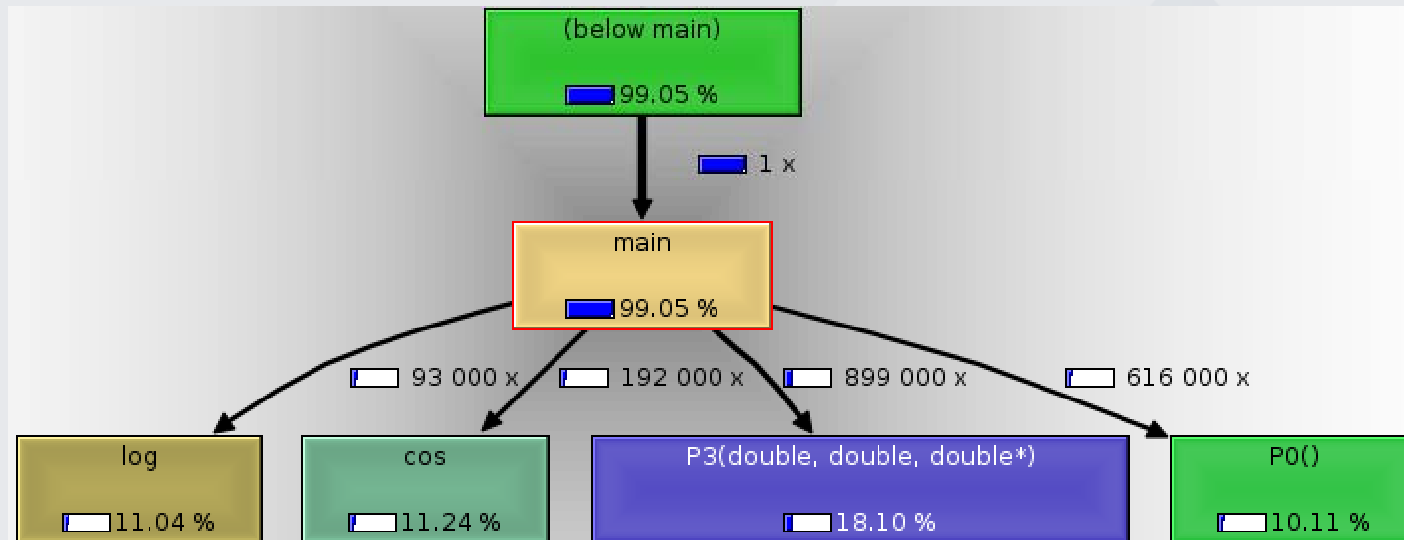
Опции компиляции:

CFLAGS += -O0 -g

Запустить valgrind:

valgrind --tool=callgrind ./wetstone

Открыть созданный файл ***callgrind.out.9023*** в ***KCachegrind***



2009 © Инновационная компания Promwad

Измерение производительности и профилирование

Gcov

Утилита из коллекции ***GCC*** для аннотации исходного кода

- Скомпилировать с флагами ***-fprofile-arcs -ftest-coverage -g -O0***
- Запустить приложение
- Сопировть файлы ****.gcda*** в каталог с исходными кодами
- Выполнить: ***bfin-linux-uclibc-gcov *.gcda***

```
     -:  267:
 32001:  268:    for (I = 1; I <= N7; I++)
     -:  269:    {
 32000:  270:        X = T * DATAN(T2*DSIN(X)*DCOS(X));
 32000:  271:        Y = T * DATAN(T2*DSIN(Y)*DCOS(Y));
     -:  272:    }
```

Измерение производительности и профилирование

Oprofile

Консольная утилита для профилирования приложений

bfin_opcontrol – управляющий скрипт

oprofiled – демон, собирающий информации о профилировании.

opreport – выводит отчет о профилировании

- Инициализация

> bfin_opcontrol --init

> oprofiled -e ' ' --no-vmlinux --image=/bin/whetstone &

> bfin_opcontrol --start

- Выполнение приложения

> whetstone

- Вывод данных профилирования

> bfin_opcontrol --dump

> opreport -l

Измерение производительности и профилирование

- Результат работы ***Oprofile*** для теста ***whetstone***

Profiling through timer interrupt

samples	%	symbol name
1128	31.9005	____divdf3
580	16.4027	____unpack_d
454	12.8394	____muldf3
403	11.3971	___fpadd_parts
376	10.6335	____pack_d
356	10.0679	____muldi3
122	3.4502	____adddf3
44	1.2443	_P3
30	0.8484	_P0
20	0.5656	____subdf3
10	0.2828	_PA

Пример оптимизации кода библиотеки ffmpeg

- Измерение использования ресурсов системы

Опции демона ***Oprofile***:

--image=/usr/bin/ffmpeg

Кодирование ***wav*** → ***ogg***:

> ffmpeg -i audio.wav audio.ogg

samples	%	app name
143251	76.2448	libavcodec.so.51.48.0
28086	14.9487	no-vmlinux
10520	5.5992	libavformat.so.52.1.0
3866	2.0577	libgcc_s.so.1
664	0.3534	libuClibc-0.9.29.so
508	0.2704	libavutil.so.49.5.0
336	0.1788	libm-0.9.29.so

Вывод: большая часть времени расходуется в библиотеке ***libavcodec.so***

2009 © Инновационная компания Promwad

Пример оптимизации кода библиотеки ffmpeg

- Профилирование библиотеки ***libavcodec.so***

--image=/usr/lib/libavcodec.so

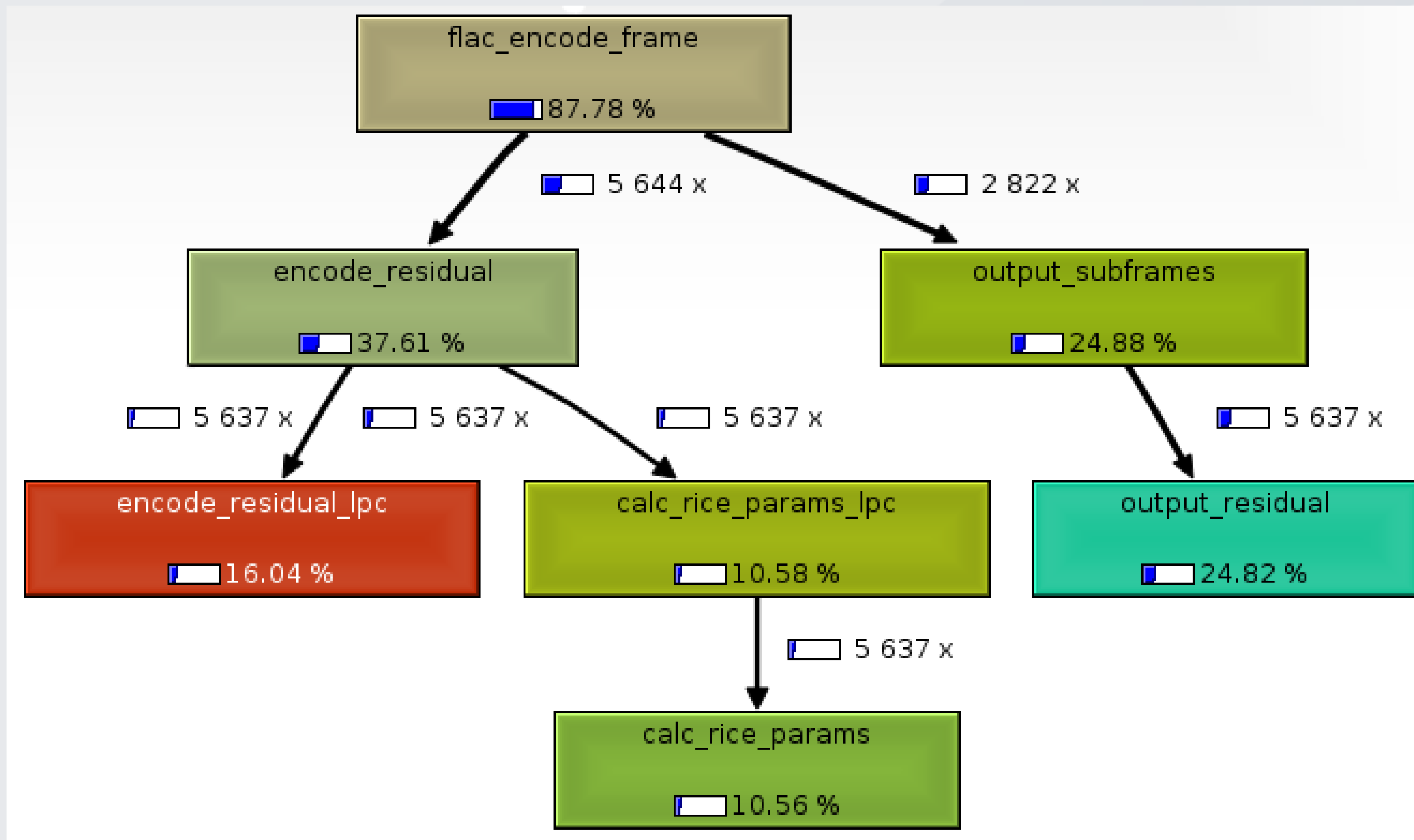
samples	%	symbol name
29417	20.4103	___muldf3
28904	20.0544	__fpadd_parts
22567	15.6576	___muldi3
20285	14.0743	___unpack_d
15761	10.9354	___pack_d
12122	8.4106	___floatsidf

- Использование новой версии кросс-компилятора

samples	%	symbol name
30465	22.6194	___muldf3
28642	21.2659	__fpadd_parts
24633	18.2893	___unpack_d
16997	12.6198	___pack_d
12092	8.9780	___floatsidf
6623	4.9174	___muldi3

Пример оптимизации кода библиотеки ffmpeg

Использование *Valgrind*



2009 © Инновационная компания Promwad

Пример оптимизации кода библиотеки ffmpeg

Размещение функций в памяти ***L1***

В функции ***output_subframes()*** в зависимости от типа принимаемого фрейма вызываются следующие функции:

- ***output_subframe_constant()***
- ***output_subframe_lpc()***
- ***output_subframe_verbatim()***
- ***output_subframe_fixed()***

Разместим их в памяти ***L1***:

```
__attribute__ ((l1_text)) void output_subframe_constant()
__attribute__ ((l1_text)) void output_subframe_lpc()
__attribute__ ((l1_text)) void output_subframe_verbatim()
__attribute__ ((l1_text)) void output_subframe_fixed()
CCFLAGS += -fno-jump-tables
```


www.promwad.com

2009 © Инновационная компания Promwad

Пример оптимизации кода библиотеки ffmpeg

- Измерение времени кодирования ***wav*** → ***ogg***

Опции компилятора	**Время кодирования**
-O2, стандартный компилятор	22м 51с
-O2, обновленный компилятор	19м 35м
-O3 -mfast-fp -ffast-math	17м 38с
-O3 -mfast-fp -ffast-math, размещение некоторых функций в L1	16м 33с

2009 © Инновационная компания Promwad

Спасибо за внимание!

Инновационная компания Promwad

Адрес: г. Минск, РБ, 220073
ул. Ольшевского 22, офис 809

Тел.: +375 17 312-12-46

Email: info@promwad.com

Сайт: www.promwad.com