MiT

SAMPL – Self-Adapting Matrix Processing Library

Пакет автоматически адаптируемых базовых подпрограмм обработки матриц

Руководство пользователя

23.04.2013

Версия 1.0

ОГЛАВЛЕНИЕ

SAMPL – Self-Adapting Matrix Processing Library 3
Инструментальные средства 3
Подготовка инструментальной машины 4
Содержимое каталога \Sampl 5
Запуск инструментария 6
Интерпретация результатов запуска инструментария 8
Демонстрационный пример 11
Файлы, необходимые генерируемой программе 12
Тело сгенерированной программы 12
Синхронизация параллельных процессов 15
Записи активации вычислительных гранул 15
Прочие файлы 16
Использование сгенерированной программы 17

SAMPL – Self-Adapting Matrix Processing Library

SAMPL представляет собой пакет инструментальных средств, позволяющий автоматически синтезировать параллельные программные реализации матричных операций, адаптированные под выбранную аппаратную платформу.

Целевая аппаратная платформа может быть построена на базе многоядерного процессора, ядра в котором неоднородны, а организация перемещения информации по иерархии памяти требует явного управления со стороны прикладной программы (В данной реализации имеется поддержка процессоров MC(12)24, MC0226, в дальнейшем список поддерживаемых процессоров может быть расширен).

После выбора аппаратной платформы и матричной операции, а также – указания конфигурации матриц, варианта их разбиения на прямоугольные блоки и количества ядер, задействованных в обработке данных, SAMPL автоматически синтезирует исходные тексты подпрограммы, написанные на языке Си. В процессе работы производится структурная адаптация программы к уровню вычислительного параллелизма, а также – возможностям транспортной подсистемы процессора. Сгенерированная подпрограмма является замкнутой (не использует стороннюю функциональность) и может быть включена в состав проекта пользователя.

Инструментальные средства

В пакет разработанных инструментальных средств входят: генератор внутреннего графового представления параллельной программы, распараллеливающая система и генератор исходных текстов SAMPL-программы на языке Си.

Существующие инструменты *позволяют генерировать* SAMPL-программы, соответствующие следующим BLAS-программам:

- программа матрично-матричного умножения (**sgemm**)[1];
- программа матрично-векторного умножения (**sgemv**);
- программа решения СЛАУ с нижнетреугольной матрицей коэффициентов и одним вектором правых частей (**strsv**);

[1] В скобках приведены обозначения программ, принятые для библиотеки BLAS

- программа решения СЛАУ с нижнетреугольной матрицей коэффициентов и несколькими векторами правых частей (**strsm**).

> **ВАЖНО**: В данной версии инструментария на указанные программы наложены следующие **ограничения**:
>
> - все скалярные коэффициенты в программах (**alpha**, **beta**) равны 1;
> - все матрицы имеют одну и ту же размерность по всем измерениям;
> - клетки, на которые разбиваются матрицы, имеют одну и ту же размерность по всем измерениям;
> - программы **strsv** и **strsm** не предполагают замещения результирующих данных «на месте».

Текущая версия инструментария позволяет сгенерировать *любую из указанных* ранее SAMPL-программ для процессоров **МС-0226** и **МС-24**. Возможно использовать инструментальные средства отдельно от скриптового окружения, однако создание тестового проекта и интеграция сгенерированных исходных текстов в него должны осуществляться вручную.

Подготовка инструментальной машины

Далее приведен список требований и действий, направленных на подготовку инструментальной машины к работе с пакетом SAMPL:

- На инструментальной машине должна быть установлена ОС **Windows XP** или ОС **Windows 7**.
- Необходим доступ к сети Интернет.
- Необходимо установить языковые средства **Ruby**[2].

> **ВАЖНО**: Текущая версия инструментария тестировалась в среде Ruby 1.9.2.

- Необходимо запустить инсталлятор и следовать его указаниям.

Теперь машина готова к использованию инструментальных средств.

[2] http://rubyforge.org/frs/download.php/75127/rubyinstaller-1.9.2-p290.exe

Содержимое каталога \Sampl

Ниже в Табл. 1 представлен перечень каталогов и файлов, содержащихся в каталоге \Sampl, а также приведены пояснения к их содержимому.

Таблица 1. Содержимое каталога \Sampl

Каталог/файл	*Содержимое*
common\bat	Вспомогательные bat-скрипты для поддержки сборки тестовых проектов
common\sil	Исходные файлы библиотеки поддержки времени исполнения SIL (SAMPL Intermediate Layer)
common\mc*.h	Файлы с описанием системных регистров процессоров «Мультикор»
common\blas*.h	Файлы с описанием интерфейса библиотеки BLAS
common\sampl_shared_databuf.c	Объявление буфера временного хранения промежуточных результатов в основной памяти
mc_0226\asm	Исходные тексты вычислительных гранул, написанные на языке ELas
mc_0226\include	Заголовочные файлы, содержащие описание формата записи активации[3] вычислительных гранул
mc_0226\ granule_<num>_info.xml	Файлы, содержащие описания вычислительных гранул
mc_0226\ procModel.xml	Описание модели процессора МС-0226
mc_24\asm	Исходные тексты вычислительных гранул, написанные на языке ELas.
mc_24\include	Заголовочные файлы, содержащие описание формата записи активации вычислительных гранул
mc_24\ granule_<num>_info.xml	Файлы, содержащие описания вычислительных гранул
mc_24\ procModel.xml	Описание модели процессора МС-24
out\asm	Исходные тексты сгенерированной программы

[3] Под записью активации понимается совокупность значений фактических параметров, передаваемых при вызове вычислительной гранулы (в отличие от классического механизма передачи параметров вызываемой подпрограмме через стек, в данном случае все параметры помещаются в пакет, который копируется в локальную память DSP-ядра с привлечением DMA-ядра).

Каталог/файл	*Содержимое*
out\include	
out\src	
out\xml	Промежуточные файлы, генерируемые инструментарием
out\project.prj	Файл настроек для тестового проекта
projects\ sampl_<name>_<arch>_sisd	Каталог тестового проекта для программы <name> и модели процессора <arch>
scripts	Каталог, содержащий вспомогательные скрипты поддержки процесса синтеза программы
templates	Каталог, содержащий шаблонные файлы для генерации скриптовых файлов компоновщика для тестового проекта
tools	Каталог, содержащий инструментарий
tools\dll	Каталог, содержащий некоторые из используемых инструментами динамически загружаемых библиотек
tools\graphGen	Каталог, содержащий скрипты, предназначенные для генерации графового представления генерируемой программы
tools\ sampl_mc_12_backend.exe	Кодогенератор для процессоров семейства «Мультикор»
tools\ sampl_resource_alloc.exe	Распараллеливающая система
rakefile	Скрипт системы rake, содержащий исполнимое описание процесса синтеза SAMPL-программ и их интеграции в тестовые проекты

Запуск инструментария

Для запуска комплекса инструментов SAMPL необходимо перейти в каталог **Sampl**, а затем набрать команду вида:

```
rake emit proc=<proc_name> prog=<prog_name> tasksize=<task_size> tilesize=<tile_size> cores=<num_of_cores> [wcccount=<num_of_weakly_connected_components>]
```

, где:

- **proc_name** – **имя модели процессора**, на котором планируется запускать сгенерированную программу (на данный момент доступны следующие варианты: { **mc_24** | **mc_0226** });
- **prog_name** – краткое имя генерируемой программы (на данный момент доступны следующие варианты: { **sgemm** | **sgemv** | **strsm** | **strsv** });
- **task_size** – размерность решаемой задачи в элементах;

ВАЖНО: В текущей версии инструментария размерность всех матриц и векторов по всем измерениям равна **task_size**.

- **tile_size** – максимальная размерность клетки в эелементах при клеточном разбиении матриц;

ВАЖНО: Текущая версия инструментария допускает указание лишь одной размерности клетки, что приводит к тому, что любая обрабатываемая матрица разбивается на клетки *квадратной конфигурации* размерностью **tile_size** (за исключением клеток последнего столбца и последней строки, которые могут иметь прямоугольную конфигурацию).

- **num_of_cores** – количество вычислительных ядер для распараллеливания вычислений;
- **num_of_weakly_connected_components** – количество слабо связных компонент графа, для которых отображение заданий на вычислительные ядра производится в течение одной итерации (данный параметр является необязательным). В случае отсутствия данного параметра распределение вычислительной нагрузки производится для всего графа в течение одной итерации.

СОВЕТ: Выбор бо́льшего значения параметра wcccount потенциально увеличивает «плотность» расписания и вероятность параллельного выполнения вычислительных заданий. Использование wcccount потенциально увеличивает вероятность прямой передачи промежуточных результатов между вычислительными ядрами, минуя буфер в основной памяти.

Интерпретация результатов запуска инструментария

В процессе работы инструментария в консоль выводятся диагностические сообщения (см. листинг 1). Пример вывода сообщений при успешной генерации программы приведен ниже.

```
1.  Cleaning...
2.  Generating abstract macro-flow graph...
3.  Time elapsed 0 seconds, 15625000 nsec.
4.  Allocating resources...
5.  Reading the macro-flow graph xml-file...
6.  Reading the processor model xml-file...
7.  Reading some command line parameters...
8.  Allocating memory for various buffers..
9.  Memory allocation results are printed below:
10. <...>
11. Performing space mapping for computational workload...
12. Allocating memory for granules...
13. Restructuring the macro-flow graph...
14. Performing inter-tick transfers mapping...
15. Performing intra-tick transfers space and time mapping...
16. Allocating memory for data blocks...
17. No temporary offload actor was found.
18. Finding out how temporary results are transferred...
19. Local transfers: 1.000000e+002%
20. Direct transfers: 0.000000e+000%
21. Transfers through the temporary store: 0.000000e+000%
22. Performing compactification of a macro-flow graph...
23. Calculating schedule length...
24. Schedule is 8 ticks long.
25. Reversing tick numbers for actors...
26. Writing particular macro-flow graph to the xml-file...
27. Code generation...
28. Generating project patch...
29. Generating loader scripts...
30. ...
31. Moving files to ./out
32. ...
33. Deleting temporary files...
34. Cleaning the project directory...
35. Patching project...
36. ...
```

Листинг 1. Пример вывода в консоль

```
1.  Memory allocation results are printed below:
2.  ----------------------------
3.   Local mem. #           hyp. code  granule code   act. rec. act. rec. ptr   data       multicast  dma-task   tmp. store
4.   Number of buffers are: 1          2              2         2               3          2          2
5.  ----------------------------
6.   Multicast-buffers are used: false
7.  ----------------------------
8.   Size of buffers are:   100        200            48        4               see below  see below  672        156
9.  0                                                                          0          0
10. 1                                                                          632        0
11. 2                                                                          52         0
12. ----------------------------
13.  Buffer size to memory size ratio, %:
14.  Local mem. #           hyp. code  granule code   act. rec.  act. rec. ptr  data       multicast  dma-task   tmp. store  cap.to mem. size, %:
15. 0                       0.000+000  0.000000e+000  0.00e+000  0.000000e+000  0.00e+000  0.00e+000  0.00e+000  2.32e-004   2.324581e-004
16. 1                       6.10e-001  1.221001e+000  0.00e+000  0.000000e+000  0.00e+000  0.00e+000  0.00e+000  0.00e+000   3.052503e+000
17. 2                       0.00e+000  0.000000e+000  7.32e-002  6.103888e-003  9.64e-001  0.00e+000  0.00e+000  0.00e+000   3.058048e+000
18. 3                       0.00e+000  0.000000e+000  0.00e+000  0.000000e+000  7.93e-002  0.00e+000  0.00e+000  0.00e+000   2.380516e-001
19. 4                       6.10e-001  1.221001e+000  0.00e+000  0.000000e+000  0.00e+000  0.00e+000  0.00e+000  0.00e+000   3.052503e+000
20. 5                       0.00e+000  0.000000e+000  7.32e-002  6.103888e-003  9.64e-001  0.00e+000  0.00e+000  0.00e+000   3.058048e+000
21. 6                       0.00e+000  0.000000e+000  0.00e+000  0.000000e+000  7.93e-002  0.00e+000  0.00e+000  0.00e+000   2.380516e-001
22. 7                       0.00e+000  0.000000e+000  0.00e+000  0.000000e+000  0.00e+000  0.00e+000  2.05e+000  0.00e+000   4.101562e+000
```

Листинг 2. Пример вывода информации о распределении памяти

СОВЕТ: Чтобы диагностические сообщения оставались на экране после завершения работы инструментальных средств необходимо предварительно открыть окно командного интерпретатора. Это можно сделать, например, (для Windows XP) выбрав **Пуск** -> **Выполнить**, набрав **cmd** и нажав клавишу **ENTER**, (для Windows 7) выбрав **Пуск**, набрав **cmd** в строке поиска файлов и программ и нажав **ENTER**.

СОВЕТ: Также оказывается полезным увеличить размер буфера консоли, чтобы сообщения не исчезали из-за его очистки. Для этого необходимо (для Windows XP и Windows 7) щелкнуть левой кнопкой мыши по пиктограмме в левом-верхнем углу окна консоли, выбрать пункт **Свойства** в контекстном меню, во вкладке **Общие** в поле **Размер буфера** ввести **999**.

При синтезе программы производится удаление результирующих файлов, оставшихся после предыдущего запуска, в подкаталоге out (1), генерируется xml-файл, содержащий описание макро-потокового графа (2, 3), производится распределение ресурсов и построение расписания для программы (4-25), завершающееся записью xml-файлов, генерируются исходные тексты программы (27), генерируется файл настроек для тестового проекта (28), генерируется набор скриптовых файлов для компоновщика (29), все сгенерированные файлы перемещаются в подкаталог out (31), удаляются временные файлы (33), каталог тестового проекта очищается от результатов предыдущей сборки, а также - от файлов, относящихся к предыдущей сгенерированной программе (34), процесс синтеза программы завершается переносом сгенерированных файлов в каталог тестового проекта.

ВАЖНО: В данной версии инструментария при расчете процентной доли занятого пространства основной памяти (строка 15 столбец cap. to mem. size, %) не учитывается размер двоичного образа программы, размещаемого в ней. Для того, чтобы получить информацию о занятом программой пространстве памяти, необходимо использовать утилиту **elcore-elvis-elf-objdump**, входящую в поставку инструментов среды разработки MCStudio.

В потоке диагностических сообщений выводится информация о процентном соотношении количества передач результатов через локальную память (19),

передач по прямым каналам, соединяющим модули локальной памяти (20), и передач через временный буфер в основной памяти (21). Кроме того, указывается общее количество этапов в процессе обработки данных (24). Данная информация позволяет предварительно оценить эффективность построенного расписания без запуска сгенерированной программы.

Демонстрационный пример

Допустим, нам необходимо сгенерировать программу матрично-векторного умножения (**sgemv**) для *матриц размером* **30** элементов и разбиения на *клетки размером* **20** элементов, для *процессора* **mc_0226** с распараллеливанием на **2** *вычислительных ядра*.

Для этого необходимо зайти в каталог **\Sampl** и набрать в консоли следующую команду:

```
rake emit prog=sgemv proc=mc_0226 tasksize=30 tilesize=20 cores=2
```

В результате будут сгенерированы исходные тексты программы, которые будут помещены в каталог **\Sampl\out**, затем будет произведена интеграция сгенерированной SAMPL-программы в проект, находящийся в каталоге **\Sampl\projects\sampl_sgemv_mc_0226_sisd**.

Для проверки работоспособности сгенерированной программы необходимо зайти в каталог **\Sampl\projects\sampl_sgemv_mc_0226_sisd** и открыть файл **project.prj** в среде **MCStudio2**. Для сборки проекта необходимо либо нажать клавиши CTRL + F9, либо выбрать пункт меню **Project->Build**. Затем необходимо выбрать пункт меню **Debug->Start->Simulator**, при этом двоичный образ проекта будет загружен в среду программной модели процессора. После этого необходимо нажать клавишу F9, либо выбрать пункт меню **Debug->Run**. Будет произведен запуск программы в среде отладчика, и впоследствии должна сработать точка останова на строке, содержащей вызов функции **infiniteLoop**. Для просмотра результатов работы можно выбрать пункт меню **View->Watches**, в котором должны отобразиться все обрабатываемые матрицы задачи. Для всех тестовых проектов в окне отладчика присутствует флаг проверки результатов вычислений **resultsAreCorrect**, который должен быть взведен.

Файлы, необходимые генерируемой программе

В любой проект, использующий хотя бы одну сгенерированную SAMPL программу должны быть включены следующие файлы:

1. sampl_<arch>_<name>_<parameters>.c(.h);
2. sampl_<arch>_<name>_<parameters>_args.c(.h);
3. granule_#.s(.h);
4. sampl_other_info.h;
5. sampl_dma_#_queue_buf.s;
6. sampl_shared_databuf.c(.h);
7. blas_enum.h;
8. sil_<arch>.c;
9. sil_<arch>_handler.c;
10. sil_<arch>_setup.s(.h);
11. sil_<arch>_ctx.s;
12. sil_sigcnt.s;
13. sil.h.

Файлы (1, 2, 4, 5 и 6(.h)) генерируются автоматически и помещаются в подкаталог **out**, остальные файлы можно найти в подкаталоге **common**, файлы с префиксом **sil_** относятся к библиотеке поддержки времени исполнения SIL (SAMPL Intermediate Layer).

Тело сгенерированной программы

Тело сгенерированной подпрограммы содержится в файле **\out\src\sampl_<arch>_<name>_<parameters>.c**. Интерфейс подпрограммы в целях совместимости идентичен интерфейсу эталонной версии из библиотеки BLAS, объявленному в заголовочном файле **\Sampl\common\blas_dense_proto.h**. Значения фактических параметров добавлены в имя сгенерированной подпрограммы в качестве суффикса **parameters** для того, чтобы можно было использовать несколько сгенерированных подпрограмм в одном проекте.

Каждая подпрограмма является реализацией статически построенного расписания для задачи с *матрицами фиксированного размера при заданном разбиении*

матриц на блоки и заданном наборе значений скалярных коэффициентов, это означает, что любое изменение конфигурации матриц либо изменение конфигурации блоков требует повторной генерации нового экземпляра программы. Несмотря на это, в одном проекте можно использовать несколько сгенерированных программ, поскольку в имя программы входит суффикс, включающий в себя значения параметров, задающих конфигурацию матриц и значения скалярных коэффициентов.

> **ВАЖНО**: Тем не менее, чтобы использовать в рамках одного проекта несколько подпрограмм, предназначенных для решения однотипных задач (например, sgemm) и отличающихся лишь разбиением матриц, необходимо вручную изменить имена файлов и имена подпрограмм во избежание конфликтов, поскольку информация о разбиении не отражается в именах указанных элементов.

Тело подпрограммы представляет собой последовательную программу для управляющего RISC-ядра. Несмотря на то, что программа RISC-ядра последовательная, она порождает процесс параллельной обработки данных, который разбит на этапы.

Каждый этап представляет собой срабатывание трехступенчатого конвейера, включающего следующие стадии:

- подготовка заданий на пересылку для DMA-ядер;
- выполнение пересылок информации с привлечением DMA-ядер;
- выполнение кода вычислительных гранул в DSP-ядрах.

На первых двух этапах конвейер заполняется, на последних двух – очищается. Очистка конвейера производится следующим образом: формируются задания на отвод окончательных результатов из локальной памяти, параллельно с этим производится формирование результатов в локальной памяти, затем результаты отводятся в основную память с привлечением DMA-ядер, после чего управление передается вызывающей программе.

Программа написана в терминах вызовов базовой функциональности промежуточного слоя SIL – Sampl Intermediate Layer. Вызовы вида

```
1.  sil_dma_task_create_2d(...)
2.  sil_dma_task_create_1d(...)
```

предназначены для создания в памяти отдельных заданий на пересылку информации. Это могут быть:

- задания на пересылку клеток матрицы или вектора;
- задания на подкачку кода вычислительных гранул в локальную память;
- задания на подкачку записей активации вычислительных гранул в локальную память.

Все задания на пересылку, выполняемые в течение одного этапа, связаны в линейную цепочку, поэтому *первый* параметр задает адрес размещения текущего задания в памяти, а *последний* – адрес следующего задания в цепочке. *Второй* и *третий* параметры определяют размещение блока информации до и после пересылки. *Четвертый* параметр – набор управляющих флагов, регулирующий логику выполнения задания (в каком направлении будет вестись пересылка, флаг окончания обработки цепочки заданий и т.д.).

Функция с суффиксом _2d предполагает, что будет передаваться блок прямоугольной плотной матрицы, хранящейся в памяти по строкам, причем из матрицы будет «вырезан» блок конфигурации MBxNB элементов, поэтому для таких функций задаются 5, 6 и 7-й параметры, обозначающие: MB, NB (в словах по 4 байта) и количество слов по 4 байта в строке матрицы (leading dimension). Функция с суффиксом _1d принимает лишь размер пересылаемого фрагмента информации в количестве слов по 4 байта (5-й параметр).

Для инициализации DSP- и DMA-ядер используются макросы вида

```
1. DSP_#_INIT(...)
2. DMA_#_INIT(...)
```

Первый макрос принимает байтовый адрес точки входа, соответствующий значению, подставляемому в регистр счетчика инструкций, второй макрос – байтовый адрес буфера, предназначенного для хранения цепочки заданий на пересылку.

Макросы вида

```
1. DSP_#_RUN()
2. DMA_#_RUN()
```

предназначены для запуска DSP- и DMA-ядер.

Каждый этап параллельной обработки данных завершается барьерной синхронизацией. Окончание этапа обозначено в программе вызовом макроса вида

```
1. SAMPL_BARRIER(...)
```

, который принимает в качестве параметра общее количество ядер (DSP- и DMA-), которые должны выполнить работу в рамках данного этапа и просигнализировать о завершении работы.

Синхронизация параллельных процессов

Синхронизация параллельных процессов в генерируемой программе базируется на механизме обработки сигналов прерываний. Сигналы прерываний поступают от DSP- и DMA-ядер и обрабатываются RISC-ядром. Механизм синхронизации реализован в следующих файлах:

- sil_<arch>_context.s;
- sil_<arch>_setup.s;
- sil_<arch>_handler.c.

В первом файле содержится код сохранения и восстановления регистров RISC-ядра. Второй файл содержит функцию настройки процессора ISR_ENABLE, активизирующую механизм обработки прерываний (Ко всем шагам в данной функции даны подробные комментарии в исходном тексте). Третий файл содержит функцию-обработчик сигналов прерываний с подробными комментариями.

Каждый сигнал прерывания приводит к увеличению порогового счетчика (определение счетчика находится в файле sil_sigcnt.s), при этом, когда поток управления в программе RISC-ядра достигает вызова макроса SAMPL_BARRIER, в программном цикле (polling) проверяется достижение счетчиком указанного значения (все требуемые устройства «отработали»), после чего пороговый счетчик обнуляется и начинается новый этап.

Записи активации вычислительных гранул

Каждый вызов вычислительной гранулы сопровождается предварительной загрузкой записи активации в локальную память DSP-ядра. Формат записи активации гранулы содержится в файле **granule_#.h**, где # - номер гранулы. Пример объявления формата записи активации представлен ниже:

```
1.  struct g_1_par_elem
2.  {
3.      unsigned int entry;
4.      unsigned int dim_0;
5.      unsigned int dim_1;
6.      unsigned int dim_2;
7.      float alfa;
8.      float beta;
9.      unsigned int inBlockAddr_0;
10.     unsigned int inBlockMcAddr_0;
11.     unsigned int inBlockAddr_1;
12.     unsigned int inBlockMcAddr_1;
13.     unsigned int inBlockAddr_2;
14.     unsigned int inBlockMcAddr_2;
15.     unsigned int outBlockAddr_0;
16. };
```

Для гранулы указывается точка входа (3), размерности решаемой задачи в количестве элементов по каждому из измерений (4-6), значения скалярных коэффициентов (7,8), адреса размещения исходных блоков данных (9-14), а также адрес размещения результирующего блока (15). Содержимое всех необходимых записей активации содержится в файле **sampl_<arch>_<name>_<parameters>_args.c**.

Прочие файлы

Помимо сгенерированных файлов и указанных ранее файлов SIL, реализующих механизмы синхронизации параллельных процессов, для интеграции программы в проект требуется добавить файлы реализации гранул вида **granule_#.s** и заголовочные файлы **granule_#.h**. Файлы реализации гранул находятся в каталоге **Sampl\<arch>\asm**, а заголовочные файлы – в каталоге **Sampl\<arch>\include**.

> **СОВЕТ**: Чтобы определить перечень требуемых гранул необходимо открыть сгенерированный файл **sampl_other_info.h** и просмотреть группу директив *#include* в начале файла.

Для хранения заданий на пересылку в памяти выделяется пара буферов для каждого работающего DMA-ядра. Размер одного буфера DMA_0_QUEUE_FRAGMENT_SIZE задается в заголовочном файле **sampl_other_info.h**, а память под буфер выделяется в файле реализации **sampl_dma_#_queue_buf.s**.

ВАЖНО: В случае использования нескольких программ в одном проекте, необходимо вручную составить файл **sampl_other_info.h**, включив в него директивы подключения всех используемых заголовочных файлов **granule_#.h**, а также добавить объявления буферов заданий на пересылку dma_#_queue_fragment_buf наибольшего размера DMA_#_QUEUE_FRAGMENT_SIZE.

Поскольку возможности буферизации результатов вычислений в локальной памяти DSP-ядер крайне ограничены, в основной памяти выделяется буфер для временной выгрузки результатов вычислений. Информация о данном буфере хранится в файлах **sampl_shared_databuf.c**(**.h**). Файл реализации не подвержен изменениям, в то время как заголовочный файл генерируется для каждой программы.

ВАЖНО: В случае использования нескольких программ в одном проекте, необходимо вручную выбрать тот файл **sampl_shared_databuf.h**, который содержит объявление буфера наибольшего размера.

Файл реализации **sampl_<arch>_<name>_<parameters>.c** подключает заголовочный файл **blas_enum.h**, содержащий объявление типов некоторых параметров BLAS-программ.

Использование сгенерированной программы

Чтобы использовать сгенерированную программу необходимо произвести ряд действий описанных далее.

Необходимо скопировать в папку проекта следующие файлы из папки **common**:

- sil\SIL.h;
- sil\sil_sigcnt.s;
- sil\<arch>*.h;
- sil\<arch>\sampl_shared_databuf.c.

Необходимо скопировать в папку проекта следующие файлы из папки **<arch>**:

- asm\granule_#.s;
- include\granule_#.h.

Заголовочные файлы гранул необходимо поместить в папку <arch> внутри корневой директории проекта. Перечень необходимых гранул можно найти в сгенерированном заголовочном файле **sampl_other_info.h**.

Необходимо скопировать в папку проекта следующие файлы из папки **out**:

- asm*_queue_buf.s;
- include*.h;
- src*.c.

В случае использования в одном проекте нескольких сгенерированных подпрограмм может возникнуть необходимость в дополнительной корректировке содержимого файлов **sampl_other_info.h**, **sampl_shared_databuf.h** (см. ранее).

В файлах, содержащих вызов сгенерированной подпрограммы, необходимо подключить следующие заголовочные файлы:

- blas_enum.h;
- sil_<arch>_setup.h.

При необходимости использования значений типа bool можно подключить заголовочный файл **SIL.h**.

До первого вызова любой из используемых сгенерированных подпрограмм необходимо добавить вызов подпрограммы `sil_setup()` и передать ей в качестве единственного параметра совокупность управляющих флагов, указывающих на то, от каких ядер разрешена обработка сигналов прерываний. Возможные значения управляющих флагов приведены в файле **sil_<arch>_setup.h**, типовое значение для процессоров MC12(24), MC0226 – `EN_DSP_DMA_CH0`.

При вызове сгенерированной подпрограммы необходимо указать ее имя согласно файлу **sampl_<arch>_<name>_<parameters>.h**, а также указать значения всех параметров согласно объявлению подпрограммы в этом файле. Следует отметить, что в теле подпрограммы при ее выполнении используются только указатели на массивы данных.

Функция `sil_setup()` настраивает вектор прерываний следующим образом: MC24 – 0x80000180, MC0226 – 0xBFC00380. Требуется, чтобы по этим адресам были размещены секции text и data файла **sil_<arch>_ctx.s**. Помимо этого необходимо

разместить секции text и data файла **sampl_dma_#_queue_buf.s** по адресу 0xB8000000 (соответствует внутренней памяти RISC-ядра CRAM). Для уменьшения числа конфликтов при попытке доступа к транспортной подсистеме со стороны RISC-ядра рекомендуется разместить секцию data файла **sil_sigcnt.s** в CRAM сразу за секциями файла **sampl_dma_#_queue_buf.s**. Все секции вычислительных гранул для DSP-ядер необходимо разместить по адресу 0x00000000.

ВАЖНО: По-умолчанию компоновщик MCStudio размещает секции, имеющие одинаковое значение PSection друг за другом, независимо от значения поля VMA (адрес). Чтобы разместить секцию по заданному адресу, необходимо задать другое значение PSection.