COREMARK — РЕАЛЬНЫЙ СПОСОБ ОПРЕДЕЛИТЬ ПРОИЗВОДИТЕЛЬНОСТЬ ЦП

ШЭЙ ГЭЛ-ОН (SHAY GAL-ON), директор отдела разработки программного обеспечения, EEMBC руководитель центра EEMBC Technology Center
МАРКУС ЛЕВИ (MARKUS LEVY), президент EEMBC

В статье рассматривается новый эталонный тест CoreMark консорциума EEMBC, позволяющий эффективно оценить производительность центральных процессоров. В этом тесте производительность увязывается с исполнением простого кода, используя общие для практически всех приложений структуры данных и алгоритмы.

Уже не раз предпринимались попытки найти показатель, который полностью характеризовал бы возможности центрального процессора. Такими единицами измерения быстродействия процессора как МГц, МОПС или МФЛОПС легко пользоваться, но они не характеризуют действительный потенциал производительности. Тест Dhrystone был первой удачной попыткой увязать показатель производительности DMIPS с исполнением реального кода. Однако теперь этот показатель, сослуживший хорошую службу отрасли, перестал работать. Попытка использования BogoMIPS для оценки скорости ЦП тоже ни к чему не привела.

Мы представляем тест CoreMark, который можно бесплатно загрузить с сайта www.coremark.org. CoreMark увязывает производительность с исполнением простого кода, используя общие для практически всех приложений структуры данных и алгоритмы. Более того, при разработке этой тестовой программы консорциум Embedded Microprocessor Benchmark Consortium (EEMBC) выбрал такую реализацию, чтобы все вычисления запускались значениями в процессе выполнения программы, что предотвращает исключение кода при оптимизации в процессе компиляции.

СТРУКТУРА COREMARK

Чтобы оценить значение CoreMark, следует проанализировать ее состав — списки, строки и матрицы. Списки, как правило, используют указатели и характеризуются комбинациями обращений к памяти не по последовательному интерфейсу. При тестировании ядра ЦП обработка списков применяется, в первую очередь, для определения того, как быстро используются данные для сканирования списка. Если списки больше, чем доступный кэш ЦП, при их обработке также проверяется эффективность кэша и иерархии памяти.

Средства обработки списочных структур данных позволяют анализировать, выполнять поиск или сортировку в соответствии с разными параметрами на основе содержимого списочных элементов данных. Для определения корректности выполнения операций CoreMark осуществляет 16-бит циклический контроль избыточностью (CRC) на основе данных, содержащихся в элементах этого списка. Поскольку CRC также часто используется во встраиваемых приложениях, это вычисление включено в CoreMark.

Во многих простых реализациях списка программы помещают списочные элементы с обращением к malloc. Однако во встраиваемых системах с ограниченной памятью списки, как правило, ограничены специальными блоками памяти, которыми управляет программист. CoreMark использует такой метод, чтобы не обращаться к коду библиотеки (malloc/free).

CoreMark разделяет имеющееся пространство данных на два блока. Один из них содержит сам список, другой — элементы данных. Это разделение также применяется при разработке встраиваемых систем, где данные собираются в буфере, а их указатели содержатся в списках (иногда — в кольцевых буферах). Элементы **data16** инициализируются на базе данных, которые недоступны во время компиляции:

```
typedef struct list_data_s {
ee_s16 data16;
ee_s16 idx;
} list_data;
```

Каждый элемент data16 состоит из двух 8-бит частей, причем верхние 8 бит содержат оригинальное значение для нижних 8 бит. В нижних 8 битах содержатся следующие данные:

0..2: тип функции для расчета значения.
3..6: тип данных для операции.
7 : показатель для предварительно вычисленного или кэшированного значения.

Эталонный тестовый код модифицирует элемент data16 при каждой итерации.

Элемент idx содержит исходный порядок следования элементов списка, благодаря чему CoreMark может восстановить исходный список, повторно не инициализируя его (требования для систем с малым объемом памяти):

```
typedef struct list_head_s {
struct list_head_s *next;
struct list_data_s *info;
} list_head;
```

Начало списка видоизменяется во время каждой итерации эталонного теста, а указатели next модифицируются, например, при сортировке списка. С каждой последовательной итерацией эталонного теста алгоритм сортирует список в соответствии с информацией в элементе data16, выполняет тестирование, восстанавливает исходный порядок в списке, сортируя и перезаписывая данные. На рисунке 1 показана базовая структура механизма доступа к связанным спискам.

Поскольку размеры указателей на центральные процессоры находятся в диапазоне 8—16 бит, количество инициализированных для списка элементов вычисляется таким образом, чтобы список содержал то же число элементов независимо от размера указателя.

МАТРИЧНАЯ ОБРАБОТКА

Во многих алгоритмах используются матрицы и массивы, что увеличивает возможности оптимизации этого типа обработки данных. Эти алгоритмы определяют эффективность операций жестко ограниченного цикла, а также способность ЦП и соответствующей цепочки инструментов использовать такие ускорители ISA (instruction set architecture — структура системы команд) как устройства умножения с накоплением и SIMD-инструкции (single-instruction, multiple-data — с одним потоком команд и многими потоками данных). Эти алгоритмы состоят из жестко ограниченных циклов, которые производят итерации во всей матрице. CoreMark выполняет простые операции во входных матрицах, в т.ч. умножение на константу, вектор или другую матрицу. CoreMark также тестирует секции данных в матрице, извлекая биты из каждого элемента матрицы для операций. Для проверки выполнения всех этих операций CoreMark повторно проводит CRC по результатам тестирования матрицы.

В матричном алгоритме CoreMark имеющееся пространство данных разделено на три части: выходную матрицу (с 32-бит значением в каждой ячейке) и две входных матрицы (с 16-бит значениями в каждой ячейке). Эти входные матрицы инициализируются на основе входных значений, недоступных при компиляции. Во время каждой итерации эталонного тестирования входные матрицы меняются на основе входных данных, которые не вычисляются при компиляции. Входные матрицы восстанавливаются на последней операции. Та же функция запускается, чтобы в точности повторить ту же обработку.

ОБРАБОТКА КОНЕЧНЫМИ АВТОМАТАМИ

У центрального процессора имеется важная функция — способность манипулировать управляющими операторами иначе, чем циклами. Конечный автомат на основе операторов переключения или условный оператор — идеальные кандидаты на проверку этой способности. В двух стандартных методах используются либо операторы переключения, либо таблица переходов. В силу того, что CoreMark уже использует второй из этих методов в алгоритме обработки списка для тестирования поведения при нагрузке и хранении, для реализации управляющей структуры блока ЦП в CoreMark применяются оператор переключения и условный оператор.

Конечный автомат тестирует входную строку, чтобы определить, являются ли входные данные числом, и принимает состояние invalid, если это не число. На рисунке 2 схематично изображен простой конечный автомат с девятью состояниями с потоком входных байтов, который обеспечивает просмотр всех доступных состояний на основе входных данных, недоступных при компиляции. Этот конечный автомат просматривает весь входной буфер.

Для проверки правильности операции CoreMark считает, сколько раз было пройдено каждое состояние.

ПРОФИЛИРОВАНИЕ В COREMARK

Поскольку CoreMark содержит несколько алгоритмов, интересно посмотреть, как поведение этого теста меняется со временем. Например, при анализе процентного отношения выполненного кода управления (пробы берутся через каждую тысячу циклов) и ошибочного прогнозирования ветви на рисунке 3 становится очевидным, где вызывается алгоритм матрицы. Например, между точками 330 и 390 показатель ошибочного прогнозирования ветви мал, а процент управления операциями высок, что указывает на жестко ограниченные циклы.

По умолчанию для CoreMark требуется лишь 2 Кбайт под все данные. Этого минимального объема памяти достаточно для поддержания работы микроконтроллеров с минимальной памятью, поэтому вполне вероятно, что CoreMark является стандартной метрикой производительности для любого ядра ЦП. На рисунке 4 представлена картина обращения к памяти во время эталонного теста. Эта информация представлена в виде процентного количества операций обращения к памяти, которое следует спустя некоторое время после предыдущего обращения. Легко понять, что пики возникают при переключении между разными алгоритмами (поскольку каждый алгоритм использует лишь часть от одной трети всего имеющегося пространства данных).

РЕЗУЛЬТАТЫ

На сайте www.coremark.org представлено свыше 120 результатов, полученных с использованием CoreMark. В таблице 1 приведены интересные данные о производительности нескольких процессоров разных производителей. Анализируя все полученные результаты, которые зависят от версии компилятора и флагов, становится понятным, что необходимо учитывать все эти детали, иначе сравнение

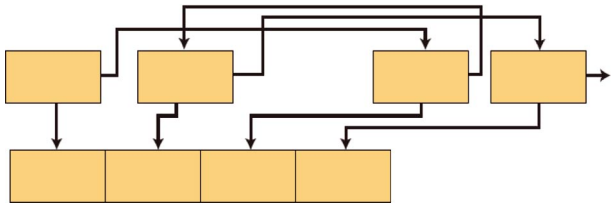

Рис. 1. **Базовая структура механизма доступа к связанным спискам. Каждый элемент списка имеет указатель данных и указатель перехода к следующему элемента списка**

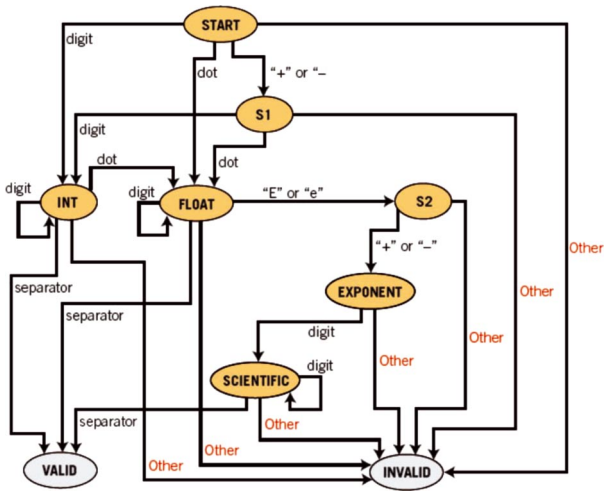

Рис. 2. **Обработка данных конечным автоматом в тесте CoreMark**

окажется некорректным. Правила выполнения программы и составления отчетов CoreMark требуют, чтобы результаты о производительности предоставлялись наряду с точными данными о версиях инструментов.

– Результаты тестирования процессоров Blackfin (1, 2) показывают, что их производительность увеличивается на 10% при переходе с GCC 4.1.2 на GCC 4.3.3, что вполне разумно ожидать при использовании новой версии компилятора.
– Результаты 8 и 9 показывают еще более значительную разницу в 18% при сравнении компиляторов разных версий, тогда как в других случаях (10, 11) и (12, 13) эти отличия выражены не так заметно, т.к. все компиляторы базируются на серии GCC4.
– Компилятор также может изменить размер кода в зависимости от производительности, как видно из результатов (3, 4). В этом случае компилятор и платформа одинаковые, но при меньшем объеме исполняемого кода (Os mpa) производительность падает на 19%, если за основу взять оптимизацию –O3. Эта разница увеличивается до 30% в случае (5, 6) (при использовании ключа mpa). Данные о ключе компилятора приводятся в отчетах на веб-сайте CoreMark.
– Другие опции компилятора определяют, сколько он попыток совершает для оптимизации кода. Результаты (7, 8) показывают, что при переходе от самой безопасной оптимизации (–O2) к нормальной (–O3) производительность увеличивается на 10%.
– При увеличении рабочей частоты память системы и встроенная флэш-память не всегда в состоянии соблюсти соотношение 1:1. У процессора с большей частотой возникает состояние ожидания из-за недостаточного быстродействия флэш-памяти. Если в ней находится код, эффективность (выраженная в единицах CoreMark/МГц) уменьшается: результаты (14, 15) показывают, что это снижение достигает почти 10%. Для (16, 17) эффект возникновения состояния ожидания выражен в еще большей мере, т.к. соотношение между ЦП и памятью поддерживается только до 50 МГц. Если рабочая частота достигает 80 МГц, это соотношение падает до 1:2, что приводит к снижению эффективности на 15%. Однако на 80 МГц абсолютное увеличение производительности на 25% больше по сравнению с 50 МГц.
– Результаты (18, 19) относятся к случаю, когда кэш слишком мал, чтобы в нем поместились все данные. В случае 18 размер кэша равен 2 Кбайт, и все данные умещаются в нем с трудом, однако для аргументов функций, которые должны быть переданы в стек, не остается места. В результате вне кэша возникает небольшой трафик шины, но если кэш увеличить (результат 19), производительность вырастет на 10%, несмотря на то, что длина конвейера стала больше — из трехступенчатого он превратился в пятиступенчатый.

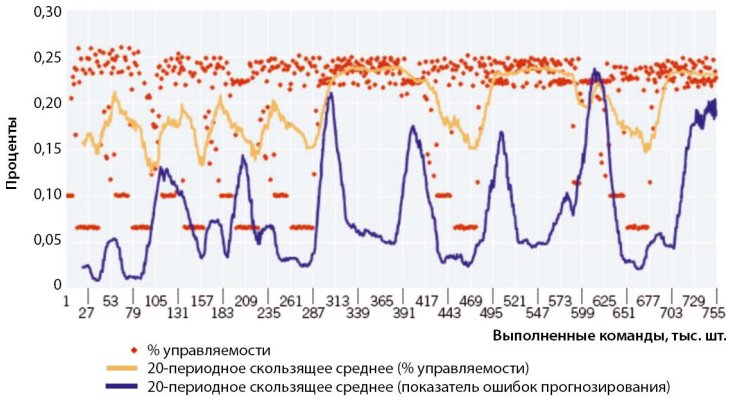

Рис. 3. **Распределение управляющих инструкций и ошибок прогнозирования при выполнении CoreMark. На графике показаны точки ввода данных и скользящее среднее для операции управления, чтобы выделить экстремальные значения**

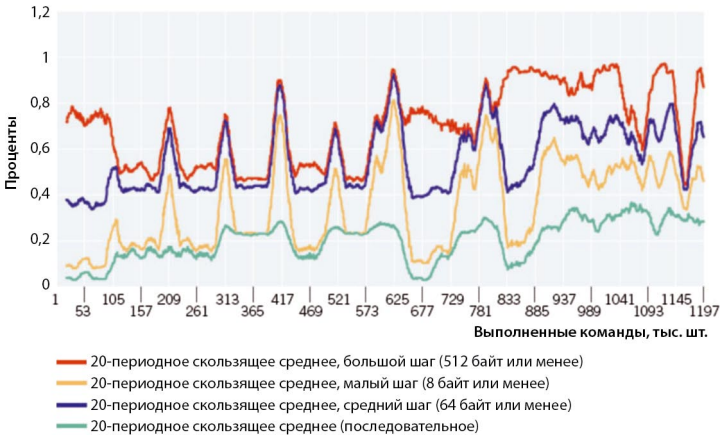

Рис. 4. **Временное распределение обращений к памяти при выполнении теста CoreMark. Около 20% времени эти обращения имеют последовательный характер, причем пики свидетельствуют о работе конечного автомата. В данном случае тестировался механизм кэша и эффективность обращений к памяти в системах без кэша**

Правила выполнения тестов и составления отчетов точно определены. Кроме того, консорциум EEMBC предлагает сертификат оценок CoreMark и стандартизованный метод измерения энергопотребления при выполнении эталонного тестирования.

Таблица 1. **Данные о производительности, CoreMark/МГц**

Процессор	Компилятор	CoreMark/МГц
1. Analog Devices BF536 0.3 393 МГц	gcc4.1.2	1,01
2. Analog Devices BF536 0.3 393 МГц	gcc4.3.3	1,12
3. Microchip PIC24FJ64GA004 32 МГц	gcc4.0.3 (dsPIC30, Microchip v3_20)	0,93
4. Microchip PIC24FJ64GA004 32 МГц	gcc4.0.3 (dsPIC30, Microchip v3_20)	0,75
5. Microchip PIC24HJ128GP202 40 МГц	gcc4.0.3 (dsPIC30, Microchip v3.12)	1,86
6. Microchip PIC24HJ128GP202 40 МГц	gcc4.0.3 (dsPIC30, Microchip v3.12)	1,29
7. Microchip PIC32MX360F512L (MIPS 32 M4K) 72 МГц	gcc3.4.4 MPLAB C32 v1.00-20071024	1,71
8. Microchip PIC32MX360F512L (MIPS 32 M4K) 72 МГц	gcc3.4.4 MPLAB C32 v1.00-20071024	1,90
9. Microchip PIC32MX360F512L (MIPS 32 M4K) 80 МГц	gcc4.3.2 (Sourcery G++ Lite 4.3-81)	2,30
10. NXP LPC1114 48 МГц	Keil ARMcc v4.0.0.524	1,06
11. NXP LPC1114 48 МГц	gcc 4.3.3 (Code Red)	0,98
12. NXP LPC1768 100 МГц	armcc 4.0	1,75
13. NXP LPC1768 72 МГц	Keil ARMCC v4.0.0.524	1,75
14. Texas Instruments OMAP 3530 500 МГц	gcc4.3.3	2,42
15. Texas Instruments OMAP 3530 600 МГц	gcc4.3.3	2,19
16. TI Stellaris LM3S9B96 Cortex M3 50 МГц	Keil ARMCC v4.0.0.524	1,92
17. TI Stellaris LM3S9B96 Cortex M3 80 МГц	Keil ARMCC v4.0.0.524	1,60
18. Xilinx MicroBlaze v7.20d in Spartan xC3S700A FPGA, 3-s	gcc4.1.1 (Xilinx MicroBlaze)	1,48
19. Xilinx MicroBlaze v7.20d in Spartan xC3S700A FPGA, 5-s	gcc4.1.1 (Xilinx MicroBlaze)	1,66