ГОСУДАРСТВЕННЫЙ НАУЧНЫЙ ЦЕНТР РОССИЙСКОЙ ФЕДЕРАЦИИ
ИНСТИТУТ ФИЗИКИ ВЫСОКИХ ЭНЕРГИЙ

ИФВЭ 2004–5
ОАФ

И.А. Качаев

PPE —ПАКЕТ ДЛЯ ПРОФИЛИРОВАНИЯ ПРОГРАММ ПОД ОС VMS

Протвино 2004

УДК 519.686.2 М–24

Аннотация

Качаев И.А. PPE — пакет для профилирования программ под ОС VMS: Препринт ИФВЭ 2004–5. – Протвино, 2004. – 16 с., 4 табл., библиогр.: 2.

Пакет PPE предназначен для профилирования программ, работающих под управлением операционной системы VMS. Он позволяет строить распределение времени работы программы по подпрограммам и строкам исходного текста программы методом гистограммирования счетчика команд. Такая информация полезна при оптимизации работы программ. Модификация программы пользователя для профилирования не нужна. Пакет является функциональным аналогом программы PCA (Performance and Coverage Analyzer) фирмы Digital.

Abstract

Kachaev I.A. PPE — Profiling Package for VMS System.: IHEP Preprint 2004–5. – Protvino, 2004. – p. 16, tables 4, refs.: 2.

PPE is a profiling package for VMS system. It permits to obtain distribution of CPU time over subroutines and source code lines inside program. Distribution is made using PC sampling method. Results are useful for program optimization. Program modification is not required. PPE is a hand-made functional analog of PCA (Performance and Coverage Analyzer) program from Digital.

[1] E-mail: kachaev@mx.ihep.su

1. Назначение

Пакет подпрограмм PPE предназначен для профилирования программ, работающих под операционной системой VMS. Под профилированием мы здесь понимаем получение распределения времени работы программы по подпрограммам и строкам исходного кода программы. Подобная информация полезна при оптимизации программы. Модификация исходного текста программы для профилирования не нужна. Функционально PPE является самодельным аналогом пакета PCA [1] (*Performance and Coverage Analyzer*) фирмы Digital.

Основным результатом работы PPE является текстовый файл с таблицей, представляющей процессорное время работы программы в разбивке по подпрограммам (точнее, модулям компиляции) и строкам исходного кода программы. Примеры работы пакета PPE представлены в **табл. 1–4**. При профилировании в терминах текста программы поддерживаются языки Фортран и Си, на уровне подпрограмм — все языки ОС VMS. Выдается также промежуточный файл, где время работы загистограммировано по адресам памяти с точностью до машинной инструкции. Время работы исчисляется как время, затраченное внутри данной подпрограммы, суммирования по дереву вызова подпрограмм нет. Анализ выполнения системных сервисов и операций ввода-вывода не производится.

Пакет PPE написан на VAX ассемблере и Фортране. В настоящее время он работает под ОС VMS для процессоров ALPHA/AXP, при необходимости пакет может быть адаптирован для VAX/VMS небольшим изменением подпрограмм, работающих с картой загрузки и листингами компилятора.

2. Принцип работы

Как и программа PCA, пакет PPE состоит из двух частей — коллектора (ядра) и анализатора (форматтера). Коллектор представляет собой набор подпрограмм, встраиваемых в программу пользователя и обеспечивающих сбор данных о времени ее работы методом гистограммирования счетчика команд (PC sampling).

Коллектор инициализируется механизмом `LIB$INITIALIZE` [2] до начала работы программы пользователя и не требует явного вызова. Информацию о счетчике команд коллектор получает, запрашивая прерывание по таймеру с периодом около 1 мс (чаще пре-

рывания по таймеру VMS V7.1 просто не выдает) и извлекая PC из блока аргументов подпрограммы — обработчика прерывания. Полученное значение гистограммируется в динамически выделяемой памяти, по окончании работы программы гистограмма записывается в текстовом виде на диск. Запись оформлена в виде обработчика выхода (exit handler), поэтому срабатывает почти всегда (см. ниже пункт 3.3) и как и инициализация прозрачна для пользователя.

Анализатор — это отдельная программа, представляющая собранные данные в удобном для пользователя виде. Используя информацию об адресах, загруженных в память подпрограмм и строк исходного кода внутри них, эта программа строит из гистограммы счетчика команд распределение процессорного времени по подпрограммам и строкам исходного кода.

Информацию об адресах подпрограмм пакет получает из карты загрузки программы, создаваемой при ее сборке программой LINK с ключом /MAP. Фактически читается информация о загруженных модулях компиляции, точнее начальные и конечные адреса соответствующих исполняемых программных секций. Информация об относительных внутри модуля адресах машинных инструкций, соответствующих строкам исходного кода программы, добывается посредством разбора листинга компилятора, порождаемого ключом `/MACHINE_CODE`. Формат этих листингов никак не стандартизирован, поэтому на данной стадии анализа возможны сбои, однако практически метод вполне надежен.

Таким образом, для своей работы коллектор PPE запрашивает следующие системные ресурсы:

- блок памяти для гистограммы (не более 1 Мб);
- прерывание по таймеру с идентификатором 997 и периодом около 1 мс.

Это означает, что коллектор практически никак не влияет на работу обычных вычислительных программ и не требует никаких привилегий для своего использования. Прерывания с периодом 1 мс на современных процессорах потребляют 1–2% CPU. Максимальное время работы коллектора составляет около 24 суток CPU (определяется переполнением внутренних счетчиков), но и в случае переполнения ничего плохого с пользовательской программой не произойдет. Надежность коллектора достаточно высока, практически он тестировался на программе, потребившей около двух недель CPU.

Пакет никак не использует механизм вызова подпрограмм и передачи аргументов в языке, на котором написана программа, а потому совместим с любыми библиотеками и любыми языками программирования (по крайней мере, теоретически). Тестирование, однако, проводилось на Фортране и Си.

3. Использование

Для того чтобы воспользоваться PPE в простейшем варианте и получить разбиение времени работы программы по подпрограммам, необходимо сделать следующее.

3.1. Компиляция и сборка программы

Никаких изменений в тексте программы и на стадии компиляции не требуется. Пакет PPE совершенно не чувствителен к тому, скомпилирована программа с оптимизацией или без нее, с дебаггером или без. В отличие от дебаггера он не требует отключения

оптимизации и не замедляет работу программы, поэтому профилировать можно рабочую версию программы. Однако следует иметь в виду, что результатом работы пакета является разбиение времени работы программы по *модулям компиляции*. По умолчанию все компиляторы операционной системы VMS делают из входного файла единый модуль компиляции, что может быть неудобно, если в файле хранится более одной подпрограммы. Для компилятора Фортрана положение можно исправить — чтобы каждая подпрограмма и функция была скомпилирована в отдельный модуль, необходимо указать ключ компиляции /SEPARATE. Для компилятора языка Си это невозможно по самой природе языка.

При сборке программы необходимо обеспечить подключение библиотеки PPE, включение в программу модуля инициализации и получение карты загрузки. Все подпрограммы пакета находятся в библиотеке `PPE_DIR:PPE.OLB`, из которой необходимо принудительно загрузить модуль `PPE$LOAD`. Команда сборки может выглядеть примерно следующим образом:

```
$ LINK/MAP=MYPROG.MAP MYPROG,MYLIB/LIB,PPE_DIR:PPE.OLB/LIB/INCL=PPE$LOAD
```

Поскольку, в отличие от PCA, PPE не подключается как альтернативный дебаггер, программа обычно собирается без ключа `/DEBUG`. По той же причине программа с загруженным коллектором *может* быть собрана с ключом `/DEBUG` и запущена с отладчиком. Пользовательские библиотеки могут быть любыми, в том числе и библиотеками ЦЕРНа. Названия всех подпрограмм пакета начинаются с символов `PPE_` и `PPE$`, так что конфликта имен возникать не должно.

3.2. Запуск программы на исполнение

Для того чтобы при запуске программы активировать коллектор и начать сбор данных для профилирования, необходимо определить логическое имя `PPE_ON` с произвольным значением. Если коллектор не активирован (логическое имя `PPE_ON` не определено), его присутствие никак не сказывается на работе программы. Далее программа может быть запущена обычным образом — из макрокоманды, командой RUN, как команда VMS и т.п. Типичная последовательность запуска может выглядеть следующим образом:

```
$ DEFINE PPE_ON 1
$ RUN MYPROG
PPE$INIT: monitored address range is 00030000 - 0003112F (Hex)
... сообщения, выдаваемые программой пользователя ...
PPE$OUTPUT: output file is DKA100:[KACHAEV.PPE]MYPROG.PPE;1
```

При старте и окончании программы будут выданы сообщения, которые указывают диапазон адресов гистограммы счетчика команд и название файла с результатами работы коллектора. Если в следуюший раз необходимо запустить программу без профилирования, достаточно удалить логическое имя `PPE_ON` командой DEASSIGN.

Результатом работы коллектора является текстовый файл с гистограммой распределения времени работы программы по адресам памяти. По умолчанию этот файл будет создан в той же директории и с тем же именем, что и исполняемый файл исследуемой программы, но с расширением `.PPE`. Разумеется, пользователь должен иметь право записи в соответствующую директорию/файл. При каждом запуске создается новая версия

файла, никакие файлы не уничтожаются. Пользователь может в явном виде указать выходной файл пакета, определив логическое имя PPE_OUT:

```
$ DEFINE PPE_OUT TEST.PPE
```

В этом случае к имени выходного файла будут применимы стандартные правила VMS: если не указано явно, файл будет создан в текущей директории, а расширение имени файла будет .PPE.

Дополнительные возможности управления коллектором. Для уменьшения размера кода, встраиваемого в программу, коллектор PPE не имеет интерфейса пользователя. Однако некоторые параметры коллектора могут быть изменены посредством определения описанных ниже логических имен. Если имена определены, все они должны транслироваться в строки текста, которые интерпретируются как положительные десятичные целые числа. Если логическое имя не существует, а также в случае любой ошибки соответствующий параметр сохраняет свое значение по умолчанию.

PPE_MEM указывает максимальный размер памяти, в страницах по 512 байт, который коллектор может использовать под гистограмму счетчика команд. Размер по умолчанию равен 2048 страниц (1 Мб), максимальный разрешенный размер — 32768 страниц (16 Мб). Алгоритм использования памяти следующий. Первоначально гистограмма запрашивается на весь диапазон адресов исполняемого кода программы с размером бина 4 байта (размер инструкции процессора Alpha/AXP). При размере счетчика также 4 байта объем памяти под гистограмму фактически равен объему исполняемого кода программы. Если при этом ограничение на объем памяти не удовлетворено, размер бина удваивается до тех пор, пока оно не будет выполнено. Таким образом, для исключительно больших программ с объемом кода более 1 Мб размер бина гистограммы становится более одной инструкции. Указав увеличенный размер памяти, этого явления можно избежать. Однако практика показывает, что и при величине бина 8–64 байт результаты анализа оказываются вполне осмысленными.

PPE_INT указывает интервал прерываний для сбора информации в 100 нс единицах времени, т.е. число 10000 соответствует 1 мс. Значение по умолчанию есть 5000, или 0.5 мс, что для VMS V7.2 приблизительно соответствует 1 мс реального времени. Практически этот интервал можно только увеличить, так как чаще прерывания просто не выдаются. Заметим, что в PCA V4.5 минимальный интервал составляет 10 мс.

PPE_MODE может принимать значение 0 или 1. Под VMS системный сервис $SETIMR позволяет указывать интервал времени между прерываниями как в единицах реального (флаг 0), так и процессорного (флаг 1) времени. Практически второй вариант означает, что прерывания выдаются, когда программа работает, и не выдаются в состояниях ожидания. Таким образом, состояния ожидания исключаются из анализа. Однако для того типа анализа, который выполняет пакет PPE, разница между этими вариантами невелика. В любом случае состояния ожидания будут записаны как *системное* время, которое практически не анализируется. Сравнив результаты, полученные в обеих модах, можно узнать, чем программа занимается в системе — работает или ожидает чего-либо. По умолчанию принята мода 0, т.е. прерывания выдаются в реальном времени. Моде 0 или 1 соответствуют команды PCA SET PC_sampling и SET CPU_sampling соответственно.

Таким образом, чтобы ограничить использование памяти до 0.5 Мб, установить прерывания через 10 мс CPU, указать выходной файл TEST.PPE и запустить профилирование, следует написать

```
$ DEFINE PPE_MEM 1024
$ DEFINE PPE_INT 100000
$ DEFINE PPE_MODE 1
$ DEFINE PPE_OUT TEST.PPE
$ DEFINE PPE_ON 1
$ RUN MYPROG
```

3.3. Завершение работы программы

При нормальном завершении работы программы выходной файл будет записан автоматически. Однако бывают случаи, когда программа завершается аварийно. Как правило, даже в этом случае результат профилирования будет сохранен. Поскольку запись выходного файла осуществляет обработчик выхода (exit handler), файл с результатами профилирования будет записан всегда, когда обработчик вызывается операционной системой. Это заведомо происходит в следующих аварийных случаях:

- фатальная ошибка программы (деление на ноль, access violation, etc.);
- применение к процессу, в котором работает программа, системного сервиса `$FORCEX`;
- при работе в очереди заданий (batch) безопасны:
  - превышение времени счета;
  - удаление задания из очереди, т.е. команда `$ DELETE/ENTRY`;
  - перезагрузка операционной системы оператором (shutdown/reboot);
- при запуске программы с терминала прерывание ее командами `^C`, `^Y` и затем `$ EXIT` также является безопасным.

Насколько автору известно, информация будет потеряна только в следующих случаях:

- при запуске программы с терминала прерывание ее командами `^C`, `^Y` и затем `$ STOP`;
- уничтожение процесса, в котором работает программа, командой `$ STOP process_name`;
- применение к процессу, в котором работает программа, системного сервиса `$DELPRC`;
- аварийная перезагрузка операционной системы.

Таким образом, фактически профилирование может происходить при исполнении программы в очереди заданий в течение нескольких суток.

3.4. Обработка результатов

Файл с результатами профилирования можно перевести в читабельный вид с помощью программы PPEOUT. Для этого ее следует запустить как внешнюю команду OS VMS:

```
$ PPEOUT :== $ PPE_DIR:PPEOUT
$ PPEOUT map_file ppe_output formatted_output
```

Параметрами программы PPEOUT являются соответственно файл с картой загрузки, файл с результатами работы коллектора и форматированный выходной файл, который и есть результат работы. Примеры выходного форматированного файла представлены в табл. 1–4. Этот файл состоит из трех частей — заголовка, таблицы времени работы

программы по модулям компиляции, и, при соответствующем запросе, таблицы времени работы программы по строкам исходного кода. Как получить эту последнюю часть, описано в разделе 4. Примеры использования программы PPEOUT приведены в разделе 5.

Заголовок выходного файла идентифицирует компьютер и тип процессора, на котором было выполнено профилирование, а также дату, файл с исследуемой программой, командную строку, с помощью которой она была запущена. Указан диапазон адресов, для которого строилась гистограмма счетчика команд (PC), интервал таймера прерываний и другая техническая информация, в том числе ширина бина гистограммы и счетчики попаданий PC в различные области программы. Заканчивается заголовок оценкой процессорного времени, потребленного программой, и его подразделения на время пользователя (затраченного в адресном пространстве программы), время затраченное в операционной системе (системных сервисах, в том числе операциях ввода-вывода, ожиданиях и т.п.) и в пространстве P1 (обычно невелико).

Во второй части файла в каждой строке таблицы указано имя подпрограммы (фактически, модуля компиляции), время ее работы и его (времени) статистическая погрешность, суммарное время работы данной подпрограммы и всех вышеперечисленных, количество попаданий счетчика команд в данный модуль, стартовый адрес модуля и его длина в байтах. Время работы указано в процентах от полного времени работы программы в моде пользователя.

Следует иметь в виду, что время здесь измеряется количеством случаев, когда при прерывании счетчик команд был обнаружен внутри данного модуля, т.е. это нечто среднее между процессорным и физическим временем — когда программа ждет ввода-вывода, время идет, когда она ожидает очереди на процессор, время все же стоит. Однако при длительных ожиданиях (ввода-вывода, локального флага и т.п.) большую часть времени счетчик команд находится не в пользовательской программе, а в соответствующем системном сервисе, так что в итоге выдаваемое PPE время работы подпрограммы достаточно надежно можно считать процессорным временем. Еще более "приблизить" его к процессорному можно определением логического имени `PPE_MODE`, как описано в разделе 3.2.

Нужно отметить также, что в применении к процессорам Alpha/AXP сам метод гистограммирования счетчика команд (PC sampling) имеет определенные ограничения. А именно, данное семейство процессоров имеет так называемые "неточные исключения" (inexact exeptions), т.е. в общем случае невозможно определить точно, какая именно инструкция и по какому адресу исполнялась на момент возникновения прерывания. В результате любой пакет профилирования, работающий методом гистограммирования счетчика команд, может обнаружить счетчик команд даже в области памяти, где нет команд процессора, например в областях выравнивания между программными секциями. В пакете PPE в этом случае программа PPEOUT выдаст на терминал предупреждение о невозможности использования части данных (пакет PCA имеет более развернутые средства доклада о подобных ошибках). От компилятора Фортрана можно потребовать использования синхронных исключений (см. HELP FORTRAN /SYNCH), но это замедляет работу программы, и при этом вы будете профилировать *другую* программу. К счастью, эффект асинхронности обычно не настолько велик, чтобы о нем беспокоиться.

4. Профилирование по строкам текста программы

Получить распределение времени работы программы по строкам исходного кода тоже достаточно просто. Небольшие отличия имеются на стадии компиляции и обработки результатов, сборка и запуск программы под управлением PPE происходят так же, как и в предыдущем случае.

Как уже было сказано, информация о соответствии адресов памяти и строк исходного кода извлекается пакетом PPE из листингов компиляции. Поэтому при компиляции с языков Фортран-77, Фортран-90, Си необходимо получить листинги компиляции с описанием сгенерированного машинного кода, т.е. указать ключи компилятора `/LIST/MACHINE_CODE`. При компиляции с Фортрана также настоятельно рекомендуется ключ `/SHOW=NOMAP`, который несколько упрощает формат листинга. Все остальные ключи компилятора, такие как `/DEBUG`, `/OPTIMIZE`, а для Фортрана еще и `/SEPARATE`, `/SYNCH` могут свободно использоваться. Листинги компиляции нужны только для тех файлов, которые вы хотите исследовать на уровне исходного кода, остальные можно компилировать как обычно.

К сожалению, в компиляторе Фортрана (по крайней мере, версии V7.2) имеется досадная ошибка, которая искажает формат листинга (приводит к потере названия первого модуля в первом файле) в том случае, если в командной строке компилятора указано более одного входного файла. Поэтому настоятельно рекомендуется компилировать файлы, указывая их в командной строке по одному. Именно так работают make-подобные утилиты MMS и MMK, которые очень удобны при работе с PPE, поскольку не только позволяют единообразно указать ключи компиляции, но и автоматически поддерживают соответствие исходного кода и листингов компиляции.

После того как вы запустили свою программу под управлением коллектора и получили соответствующий .PPE файл, результат следует обработать программой PPEOUT с расширенным форматом вызова

```
$ PPEOUT map_file ppe_output formatted_output listing listing ...
```

где первые три аргумента те же, что и в предыдущем случае, а именно карта загрузки программы, файл с результатами работы коллектора и выходной файл, а остальные, если присутствуют, представляют собой имена файлов с листингами компиляции, полученными на предыдущем шаге. Поскольку листингов может быть довольно много, в именах файлов с листингами разрешено использование метасимволов VMS '`%`', '`*`', '...', предназначенных для указания групп файлов. Использование метасимволов в первых трех аргументах запрещено. Стандартные расширения для имен файлов: .MAP — для карты загрузки, .PPE — для выходного файла коллектора, .LIS — для форматированного выходного файла и листингов компиляции. Директорией по умолчанию для всех файлов является текущая директория, и если у вас это не так, диск и директорию у всех файлов надо указывать явно. Указание директории для одного файла на другие автоматически не распространяется.

Практически удобно держать все листинги в одной (библиотечной или рабочей) директории и указывать их все по логическому имени, а для форматированных выходных файлов принять какое-либо стандартное расширение, например .PPL, чтобы не путать их с листингами. Тогда команда форматирования результатов для программы MYPROG, у которой листинги лежат в директории `MY_LIB`, может выглядеть так:

```
$ PPEOUT MYPROG.MAP MYPROG.PPE MYPROG.PPL MY_LIB:*.LIS
```

Даже если вы где-то ошибетесь — перепутаете порядок аргументов, пропустите аргумент, захватите "по звездочке" не те файлы — ничего страшного не произойдет. Программа PPEOUT не стирает никакие файлы (для выходного файла создается новая версия), входные файлы открывает только на чтение (т.е. они могут находиться в чужой директории) и, как правило, способна обнаружить и игнорировать некорректный ввод. Разумеется, если программа при работе выдает какие-либо предупреждения, следует проверить входные данные. К сожалению, программа PPEOUT *не способна* проверить соответствие предложенных .PPE файла и карты загрузки, однако соответствие карты загрузки листингам компиляции проверяется (при несовпадении длин модуля, выдаваемых компилятором и сборщиком, описание модуля будет проигнорировано).

Программа анализирует по строкам исходного кода те модули компиляции, которые потребили не менее 1% от процессорного времени, затраченного в моде пользователя. Если в указанных пользователем листингах не будет найден соответствующий модуль, он просто не появится в выходном файле. Таким образом, вы можете указать программе листинги только тех модулей, которые вас интересуют. Формат выходного файла тот же, который был описан в предыдущем разделе. Для каждого модуля компиляции, выбранного для детального анализа, приводится его текст, с правой стороны которого изображена гистограмма распределения времени работы по строкам текста. Слева от графического изображения указано число попаданий счетчика команд в области памяти, соответствующие данной строке, а справа от него дан суммарный процент времени работы модуля от его начала до данной строки включительно. С помощью этого числа легко оценить долю времени, затраченную в любой области модуля компиляции.

5. Примеры

Рассмотрим несколько примеров. В качестве первого примера возьмем SIMP.C — "классическую" программу[1], выполняющую разложение числа на множители методом перебора. Профилирование было выполнено следующим образом:

```
$ CC/LIST/MACH SIMP.C
$ LINK/MAP SIMP,PPE_DIR:PPE/LIB/INCL=PPE$LOAD
$ SIMP :== $USER$BEC:[KACHAEV.ND2]SIMP
$ DEFINE/USER PPE_ON 1
$ SIMP 1000000000000003
PPE$INIT: monitored address range is 00030000 - 00030EAF (Hex)
14902357
67103479
PPE$OUTPUT: output file is DKA100:[KACHAEV.ND2]SIMP.PPE;2
$ PPEOUT :== $PPE_DIR:PPEOUT
$ PPEOUT SIMP.MAP SIMP.PPE SIMP.PPL SIMP.LIS
MAP information file: SIMP.MAP
PPE statistic   file: SIMP.PPE
OUTPUT file: SIMP.PPL
Processing compiler listing SIMP.LIS
File USER$BEC:[KACHAEV.ND2]SIMP.LIS;2
```

[1]Пример предоставлен С. Копиковым.

Для однократного профилирования логическое имя PPE_ON удобно определить в моде пользователя, поскольку в этом случае оно автоматически удаляется по окончании работы программы. Результаты профилирования представлены в **табл. 1**. Они достаточно очевидны (более 90% времени программа затрачивает в операционной системе, поскольку процессоры Alpha/AXP не имеют операции целочисленного деления, и она выполняется библиотечной программой) и приведены как пример работы пакета PPE с программой, написанной на языке Си.

Следующий пример может быть более содержательным. Профилированию была подвергнута программа KTEST3, написанная на Фортране и выполняющая некоторые теоретико-числовые вычисления. Компиляция и сборка программы были выполнены с помощью make-файла формата программ MMS/MMK. Для профилирования были модифицированы ключи компилятора и сборщика, а также команда сборки. Соответствующий make-файл выглядит следующим образом:

```
# KTEST3.MMS - VMS makefile for KTEST3 program.
# Usage: $ MMK/DESCRIP=KTEST3.MMS
# Note: action line must have <tab> as a first character.

FFLAGS = $(FFLAGS)/LIST/MACH/SHOW=NOMAP
LINKFLAGS = $(LINKFLAGS)/MAP=$(MMS$TARGET_NAME)
PPE_LIB = PPE_DIR:PPE.OLB/LIB/INCL=PPE$LOAD
OBJS = NTH_LLL.OBJ,NTH_CLSN.OBJ,NTH_CLSA.OBJ,DPOFA.OBJ,DPOSL.OBJ,BLAS.OBJ

KTEST3.EXE : KTEST3.OBJ,$(OBJS)
        $(LINK)$(LINKFLAGS) KTEST3.OBJ,$(OBJS),$(PPE_LIB)
```

Профилирование было выполнено следующим образом:

```
$ MMK/DESCRIP=KTEST3.MMS
$ KTEST3 :== $ USER$BEC:[KACHAEV.ND2]KTEST3
$ DEFINE/USER PPE_ON 1
$ KTEST3 1 300023 1 17340  48702  53182  55430  59089 130359
PPE$INIT: monitored address range is 00030000 - 0003C7AF (Hex)
...
PPE$OUTPUT: output file is DKA100:[KACHAEV.ND2]KTEST3.PPE;1
$ PPEOUT KTEST3.MAP KTEST3.PPE KTEST3_PPE.LIS KTEST3*.LIS NTH*.LIS DPO*.LIS
MAP information file: KTEST3.MAP
PPE statistic   file: KTEST3.PPE
OUTPUT file: KTEST3_PPE.LIS
Processing compiler listing KTEST3*.LIS
File USER$BEC:[KACHAEV.ND2]KTEST3.LIS;1
File USER$BEC:[KACHAEV.ND2]KTEST3_PPE.LIS;2
File open failure
Processing compiler listing NTH*.LIS
File USER$BEC:[KACHAEV.ND2]NTH_CLSA.LIS;1
File USER$BEC:[KACHAEV.ND2]NTH_CLSN.LIS;1
...
```

То, что выходной файл KTEST3_PPE.LIS попал в группу листингов KTEST3*.LIS, не препятствует работе программы PPEOUT — он просто будет игнорирован. Результаты профилирования приведены в **табл, 2, 3**. В программе KTEST3 сравниваются три метода вычислений, два из которых оформлены целиком в виде подпрограмм NTH_LLL, NTH_LLD, а третий представлен россыпью всех остальных перечисленных в выходном файле модулей. Таким образом, для оценки структуры потребления времени этим последним методом достаточно профилирования на уровне модулей, в то время как для первых двух желателен анализ на уровне строк программного кода. Результат такого анализа для модуля NTH_LLL представлен в таблицах (в несколько укороченном виде). Видно, что этот модуль состоит из трех крупных частей, отмеченных в комментариях как ортогонализация Грама-Шмидта, редукция длины вектора и перестановка векторов. Из указанного в правой части таблицы интеграла потребленного времени можно сделать вывод, что именно первая часть подпрограммы, ортогонализация, потребила около 66% времени. Этот неожиданный результат предлагает направление возможной оптимизации. Действительно, подробный анализ алгоритма (не программной реализации) показывает, что ортогонализации иногда можно избежать.

Рассмотрим еще один пример работы пакета PPE. Профилированию была подвергнута программа парциально-волнового анализа. Это достаточно большая программа, написанная в основном на Фортране и частично на Си, имеющая около 40000 строк кода. Данные были собраны коллектором во время обычной работы программы в очереди заданий. Результаты приведены в **табл. 4**. Несмотря на то, что программа читает файл объемом около 250 Мб, оказывается, что она является чисто вычислительной (системное время около 1%). По-видимому, ввод-вывод в VMS Фортране является параллельным. Оптимизировать данную программу непросто, так как более 80% времени сосредоточено в одной подпрограмме, и так уже подвергнутой ручной оптимизации (развертке цикла). Отметим, что не следует слишком доверять результатам измерения времени работы отдельных строк программы. Например, в разных строках развернутого цикла эти времена отличаются более чем в два раза. Это может быть как влиянием асинхронности, так и свойством оптимизирующего компилятора. В данном примере все листинги компиляции были сохранены в библиотечной директории, поэтому результаты профилирования были сформатированы командой

```
$ PPEOUT PWLOOP.MAP PWLOOP.PPE PWLOOP.PPL PWA_LIB:*.LIS
```

6. Технические детали

Опишем кратко пользовательские подпрограммы пакета. Все они являются подпрограммами, т.е. не возвращают никакого значения. Формат передачи аргументов соответствует стандарту Фортрана, т.е. передается длинное слово по ссылке. При возникновении любой ошибки управление пользователю не возвращается, выполнение программы прекращается с выдачей либо текстового сообщения, либо системного кода ошибки.

PPE$LOAD() — стандартная инициализация, без параметров, управляется логическими именами как описано выше в пункте 3.2. Профилирование начинается автоматически. Подпрограмма эта запускается механизмом LIB$INITIALIZE [2] до начала работы программы пользователя, явным образом вызывать ее не нужно.

PPE$INIT(ISTART,IEND[,NPAGE]) — низкоуровневая инициализация, задает явно начальный адрес области профилирования, конечный адрес и максимальное количество

страниц под гистограмму. Количество страниц может быть опущено. Проверяется наличие логического имени PPE_ON, при его отсутствии эта и все последующие подпрограммы при своем вызове не делают ничего. Профилирование автоматически не запускается.

PPE$SET(INTERVAL,MODE) устанавливает интервал прерываний в 100 нс единицах времени, а также тип прерываний — в единицах реального или процессорного времени (см. 3.2);

PPE$START() запускает профилирование или продолжает его, если оно было приостановлено;

PPE$PAUSE() временно приостанавливает профилирование;

PPE$STOP() заканчивает профилирование и записывает файл с результатами на диск. Использует логическое имя PPE_OUT, как описано в пункте 3.2. Подпрограмма эта декларирована как обработчик выхода (exit handler), явным образом вызывать ее не нужно.

Практически, обычному пользователю все эти вызовы не нужны. Единственное мыслимое применение — с помощью вызовов PPE$START() и PPE$PAUSE() выделить для профилирования часть программы или, наоборот, в какой-то области кода профилирование запретить.

Возможные проблемы. Пакет PPE полагается на определенные умолчания, принятые в программе LINK, однако при использовании файла опций пользователь может эти умолчания нарушить. А именно, по умолчанию программные секции с одинаковыми атрибутами загружаются сборщиком в алфавитном порядке. Этот факт используется пакетом PPE в двух местах. Во-первых, выполнение этого правила необходимо для корректной работы механизма LIB$INITIALIZE [2] Во-вторых, пакет PPE определяет границы исполняемого кода программы, добавляя в нее исполняемые программные секции $$$$$CODE и _AAA$CODE нулевого размера и считая, что первая из них расположена до, а вторая после исполняемого кода пользователя (исключая библиотеки). Таким образом, если пользователь изменяет порядок загрузки программных секций операторами CLUSTER или COLLECT файла опций, он должен соблюсти эти условия. Для выполнения первого из них документация [2] рекомендует включить в файл опций следующие операторы:

```
CLUSTER = ,,,, SYS$LIBRARY:STARLET.OLB/include = LIB$INITIALIZE
COLLECT = , LIB$INITIALIZDZ, LIB$INITIALIZD_, LIB$INITIALIZE,LIB$INITIALIZE$
```

7. Установка пакета

Пакет PPE можно взять по адресу http://sirius.ihep.su/~kachaev/vms/ppe.zip, в сетевых VMS архивах или у автора данного описания. Установка его проста. Достаточно распаковать архив в любую директорию, создать логическое имя PPE_DIR, указывающее на эту директорию, и определить программу PPEOUT как внешнюю команду VMS:

```
$ set def MY_DISK:[USER.PPE]
$ unzip ppe.zip
$ define PPE_DIR MY_DISK:[USER.PPE]
$ PPEOUT :== $ PPE_DIR:PPEOUT.EXE
```

Для работы пакета необходимы только файлы PPE.OLB и PPEOUT.EXE. При желании их можно перекомпилировать с помощью макрокоманды MAKE_PPE.COM.

Таблица 1. Результат профилирования для программы на языке Си.

```
VMS PPE profile on BEC, Digital Personal WorkStation,  4-JAN-2004 21:01
Image: DKA100:[KACHAEV.ND2]SIMP.EXE;3
Command line: SIMP 1000000000000003
Address range   30000 -   30EAF (Hex) length    3760 Sample interval  0.5 ms PC

Total count         2168 | System count        2109 | Overflow           0
In range count        59 | P1 space count         0 | Underflow          0
Number of bins       940 | Bin width, bytes       4 | Max bin content    8

CPU time     2.08 s, User time   2.72% System time  97.28% P1 space time  0.00%

          PPE - STATISTIC PRINTOUT  IN DECREASING CPU TIME.
          ***********************

   #  Module name           Time %      Acc %    Hits  Address   Size
   ------------------------------------------------------------------
   1 - SIMP            100.00 +- 0.00  100.00      59    30000    732

         Program text. Module SIMP                     Hits total     59   Acc %
-------------------------------------------------------------------------------
# include <stdlib.h>                                          |
# include <stdio.h>                                           |
# include <math.h>                                            |
                                                              |
# define ordinal  long long /* unsigned __int64 */           |
# define format   "%llu\n"                                    |
                                                              |
ordinal sd (ordinal num)                                      |
{                                                             |
  ordinal lim, cnt;                                           |
  lim = (ordinal) sqrt ((double) num);                        |
  cnt = 2;                                                    |
  if ((num % cnt) == 0) return cnt;                           |
  for (cnt = 3; cnt <= lim; cnt += 2)                      5|X               8.5
    if ((num % cnt) == 0) return cnt;                     54|XXXXXXXXXXXXXX 100.0
  return num;                                                 |
}                                                             |
                                                              |
int main (int argc, char** argv)                              |
{                                                             |
  ordinal word, mul;                                          |
                                                              |
  if ((argc != 2) ||                                          |
        (sscanf (argv[1], format, &word) != 1))               |
        printf ("usage: simp <number> \n");                   |
  else                                                        |
    do {                                                      |
      mul = sd (word);                                        |
      word = word/mul;                                        |
      printf (format, mul);                                   |
    } while (word > 1);                                       |
  return 0;                                                   |
}                                                             |
```

Таблица 2. Результат профилирования программы KTEST3.

```
VMS PPE profile on BEC, Digital Personal WorkStation,  5-JAN-2004 19:32
Image: DKA100:[KACHAEV.ND2]KTEST3.EXE;2
Command line: KTEST3 1 300023 1 17340 48702 53182 55430 59089 130359
Address range   30000 -   3C7AF (Hex) length   51120 Sample interval  0.5 ms PC

Total count        28151 | System count         922 | Overflow           0
In range count     27207 | P1 space count        22 | Underflow          0
Number of bins     12780 | Bin width, bytes       4 | Max bin content   648

CPU time    27.36 s, User time  96.65% System time   3.28% P1 space time  0.08%

          PPE - STATISTIC PRINTOUT  IN DECREASING CPU TIME.
          ***********************

   #  Module name           Time %      Acc %    Hits  Address   Size
   ------------------------------------------------------------------
   1 - NTH_LLL          21.30 +- 0.25   21.30    5794    38970   2160
   2 - NTH_LLD          21.19 +- 0.25   42.49    5766    391E0   2544
   3 - NTH_CLSA         19.01 +- 0.24   61.50    5172    35480   5840
   4 - NTH_CLSN         10.52 +- 0.19   72.02    2863    31A40   2064
   5 - DPOFA             6.19 +- 0.15   78.21    1683    39BD0    528
   6 - DPOSL             6.09 +- 0.14   84.30    1657    39DE0    592
   7 - DDOTX             3.32 +- 0.11   87.62     904    3A210    156
   8 - NTH_GRDI          2.90 +- 0.10   90.52     790    33C10    300
   9 - NTH_GRDN          2.74 +- 0.10   93.27     746    332F0   2336
  10 - NTH_CLS3          1.73 +- 0.08   94.99     470    36CF0   1808
  11 - NTH_GRDM          1.21 +- 0.07   96.20     329    33D40    408
  12 - DNORMX            0.94 +- 0.06   97.14     255    3A190    124
  13 - NTH_LLL2          0.85 +- 0.06   97.99     231    37620   1184
  14 - NTH_SVP3          0.74 +- 0.05   98.72     200    38040   2352
  15 - NTH_DUAL8         0.42 +- 0.04   99.14     114    3A800    496
  16 - NTH_GRD3          0.32 +- 0.03   99.46      86    36B50    416
  17 - NTH_SORT3         0.22 +- 0.03   99.68      61    37400    540
  18 - NTH_SORTN         0.18 +- 0.03   99.86      48    33EE0    552
  19 - DSWAPX            0.10 +- 0.02   99.96      28    3A0F0    148
  20 - NTH_SPT           0.03 +- 0.01  100.00       9    30270   2336
  21 - KTEST3            0.00 +- 0.00  100.00       1    30000    624

         Program text. Module NTH_LLL                  Hits total   5794   Acc %
-------------------------------------------------------------------------------
      subroutine nth_lll(n, m, delta, b, u, c, info)      3|                 0.1
*                                                         |
* LLL lattice reduction, with ortogonalization.          |
*                                                         |
      implicit none                                       |
      integer n, m, info(5), k, i, j, jj                  |
      double precision delta, b(n,m), u(m,m), c(m)        |
      double precision t, cold, cnew, uold, sum           |
      integer*8 mm                                        |
*                                                         |
* Start of main loop.                                    |
* Gram-Shmidt procedure. Ortogonalize b(k), find mu(      |
*                                                         |
      k = 2                                               |
      do while (k.le.m)                                  24|                 0.5
      norto = norto + 1                                  13|                 0.7
      if (k.eq.2) then                                   20|                 1.1
        sum = 0.D0                                        |
        do jj = 1, n                                     23|                 1.4
          sum = sum + b(jj,1)**2                         81|X                2.8
        enddo                                             |
        c(1) = sum                        ! ||b(1)||     11|                 3.0
      endif                                               |
```

Таблица 3. Результат профилирования программы KTEST3 — продолжение.

```
      do j = 1, k - 1                                    117|XX               5.1
        sum = 0.D0                                        26|                 5.5
        do jj = 1, n                                     145|XX               8.0
          sum = sum + b(jj,k)*b(jj,j)     ! <b(k),b(    1015|XXXXXXXXXXXXXXX 25.5
        enddo                                             |
        do i = 1, j - 1                                  199|XXX              29.0
          sum = sum - u(i,j)*u(i,k)*c(i)                 353|XXXXXX           35.1
        enddo                                             |
        u(j,k) = sum/c(j)                 ! mu(k,j)      770|XXXXXXXXXXX     48.3
      enddo                                               |
      sum = 0.D0                                          |
      do jj = 1, n                                         75|X                49.6
        sum = sum + b(jj,k)**2            ! ||b(k)||     430|XXXXXXX          57.1
      enddo                                               |
      do j = 1, k - 1                                      46|X                57.9
        sum = sum - u(j,k)**2*c(j)                       342|XXXXX            63.8
      enddo                                               |
      c(k) = sum                          ! ||b^(k)|      18|                 64.1
*                                                         |
* Length reduce k-th vector. Modify u(k,j), j=1..k-1       |
*                                                         |
      do j = k-1, 1, -1                                   105|XX               66.0
        mm = kidnnt(u(j,k))                               144|XX               73.0
        if (mm.ne.0) then                                  39|X                73.7
          nredu = nredu + 1                                47|X                75.6
          do jj = 1, n                                     36|X                76.2
            b(jj,k) = b(jj,k) - mm*b(jj,j)                532|XXXXXXXX         85.4
          enddo                                           |
          u(j,k) = u(j,k) - mm                             57|X                86.4
          do jj = 1, j - 1                                 33|X                87.0
            u(jj,k) = u(jj,k) - mm*u(jj,j)                132|XX               89.2
          enddo                                           |
        endif                                             |
      enddo                                               |
*                                                         |
* Swap b(k), b(k-1) if new b(k-1) will be much short       |
*                                                         |
      uold = u(k-1,k)                                      53|X                90.7
      cold = c(k-1)                                       |
      cnew = c(k) + uold*uold*cold      ! new c(k-1)       62|X                91.8
      if (delta*cold .gt. cnew) then                       83|X                93.2
        nswap = nswap + 1                                  11|                 93.4
        do jj = 1, n                                       37|X                94.0
          t = b(jj,k)                                     126|XX               96.2
          b(jj,k) = b(jj,k-1)                             148|XX               98.8
          b(jj,k-1) = t                                    20|                 99.1
        enddo                                             |
        k = max(k-1,2)                                     38|X                99.8
      else                                                |
        k = k + 1                                          12|                100.0
      endif                                               |
      enddo                                               |
*                                                         |
* End of main loop                                        |
*                                                         |
      info(1) = nredu                                      1|                100.0
      info(2) = nswap                                     |
      end                                                 |
```

Таблица 4. Результат профилирования программы PWLOOP.

```
VMS PPE profile on BEC, Digital Personal WorkStation,  4-DEC-2003 16:07
Image: DKA100:[KACHAEV.S]PWLOOP.EXE;1
Command line: PWLOOP 1560 1600 40 USER$BEC:[KACHAEV.IN]INA7F11.TXT A7F11
Address range  1106D0 -  1CCC4F (Hex) length  771456 timer interval  0.5 ms PC

Total count      5020617 | System count       53869 | Overflow        20449
In range count   4945448 | P1 space count       851 | Underflow           0
Number of bins    192864 | Bin width, bytes       4 | Max bin content 75412
CPU time  4613.95 s, User time  98.91% System time   1.07% P1 space time  0.02%

          PPE - STATISTIC PRINTOUT  IN DECREASING CPU TIME.
          ***********************

   #  Module name           Time %      Acc %    Hits  Address   Size
   ------------------------------------------------------------------
   1 - DMXM11           82.35 +- 0.02   82.35 4072396   13A370   1984
   2 - DERCN1            3.04 +- 0.01   85.38  150224   139DE0    592
   3 - DBTOV             2.34 +- 0.01   87.72  115704   134430   1760
   4 - DLASR             2.33 +- 0.01   90.06  115461   1B41D0  11736
   5 - DCVDOT            2.27 +- 0.01   92.33  112275   144300    384
  ....
 187 - COHPRT           0.00 +- 0.00  100.00       1   11B580    912

         Program text. Module DMXM11                   Hits total 4072396  Acc %
-------------------------------------------------------------------------------
      SUBROUTINE DMXM11(V, G, N)                        170|                 0.0
C                                                        |
C Optimized version with manual loop unrolling.          |
C                                                        |
      IMPLICIT NONE                                      |
      INTEGER N, NREP, MXCUT, LREP, IP, I, J, K, L       |
      DOUBLE PRECISION G(*), V(*)                        |
      PARAMETER (NREP=12, MXCUT=3000)                    |
      DOUBLE PRECISION X(NREP,MXCUT)                     |
      SAVE X, LREP                                       |
                                                         |
      LREP = LREP + 1                                   332|                 0.0
      DO 10 I = 1, N                                   1061|                 0.0
   10 X(LREP,I) = V(I)                                84919|XXX              2.1
      IF (L.LT.NREP) RETURN                              36|                 2.1
C----                                                    |
      IP = 0                                              4|                 2.1
      DO J = 1, N                                      1222|                 2.2
        DO I = 1, J                                   29302|X                2.9
          IP = IP + 1                                  7605|                 3.1
          G(IP) = G(IP)                              220352|XXXXXXXX         8.5
     &      +(X(1,I)*X(1,J)                           416842|XXXXXXXXXXXXXXX 18.7
     &      + X(2,I)*X(2,J)                           271289|XXXXXXXXXX      31.1
     &      + X(3,I)*X(3,J)                           187199|XXXXXXX         35.7
     &      + X(4,I)*X(4,J)                           165466|XXXXXX          39.8
     &      + X(5,I)*X(5,J)                           254862|XXXXXXXXXX      46.0
     &      + X(6,I)*X(6,J)                           397639|XXXXXXXXXXXXXXX 55.8
     &      + X(7,I)*X(7,J)                           232070|XXXXXXXXX       61.5
     &      + X(8,I)*X(8,J)                           155705|XXXXXX          71.1
     &      + X(9,I)*X(9,J)                           318474|XXXXXXXXXXX     79.0
     &      + X(10,I)*X(10,J)                         226793|XXXXXXXXX       84.5
     &      + X(11,I)*X(11,J)                         153990|XXXXXX          88.3
     &      + X(12,I)*X(12,J))                        318841|XXXXXXXXXXX     99.9
        END DO                                             |
      END DO                                               |
      LREP = 0                                             |
      END                                                 831|               100.0
```

8. Заключение

Как указано в тексте программы, первым автором ее является Mike Waters, исходная версия датирована 1985 годом.

Автор данной работы реализовал профилирование в терминах исходного кода, возможность работы без модификации программы пользователя и другие сервисные функции, а также адаптировал пакет для процессоров ALPHA/AXP. Программа имеет статус свободного матобеспечения (*freeware*). Настоящее описание написано в надежде, что программа может быть полезна широкому кругу пользователей.

Список литературы

[1] Описание программы PCA для системы OpenVMS. В Интернет доступно по адресу `http://h71000.www7.hp.com/DOC/73FINAL/5831/5831_toc.htm`

[2] Концепции программирования OpenVMS, часть 35. В Интернет доступно по адресу `http://h71000.www7.hp.com/doc/731FINAL/5841/5841pro_094.html`

Рукопись поступила 19 января 2004 г.

И.А.Качаев.
PPE — пакет для профилирования программ под ОС VMS.

Оригинал-макет подготовлен с помощью системы **LaTeX.**
Редактор Н.В.Ежела.

Подписано к печати 21.01.2004. Формат 60 × 84/8.
Офсетная печать. Печ.л. 2. Уч.-изд.л. 1,6. Тираж 130. Заказ 185.
Индекс 3649.

ГНЦ РФ Институт физики высоких энергий
142284, Протвино Московской обл.

Индекс 3649

П Р Е П Р И Н Т 2004–5, И Ф В Э, 2004