ЛАБОРАТОРНАЯ РАБОТА №1

ПЛАН

По дисциплине: Вычислительные системы, сети и телекоммуникации
Тема занятия:Измерения вычислительной мощности вычислительной системы
Цель занятия: Измерить вычислительную мощность с помощью бенчмарка Linpack
Количество часов: 2

Содержание работы

Как известно, FLOPS – это единица измерения вычислительной мощности компьютеров в операциях с плавающей точкой. Для выяснения возможностей супер- и просто компьютеров существуют чуть более приближенные к реальным вычислительным задачам бенчмарки, например, SPEC: SPECint и SPECfp. И, тем не менее, FLOPS активно используется в оценках производительности и публикуется в отчетах. Для его измерения давно уже использовали тест Linpack, а сейчас применяют открытый стандартный бенчмарк из LAPACK.

FLOPS – это количество вычислительных операций или инструкций, выполняемых над операндами с плавающей точкой (FP) в секунду.

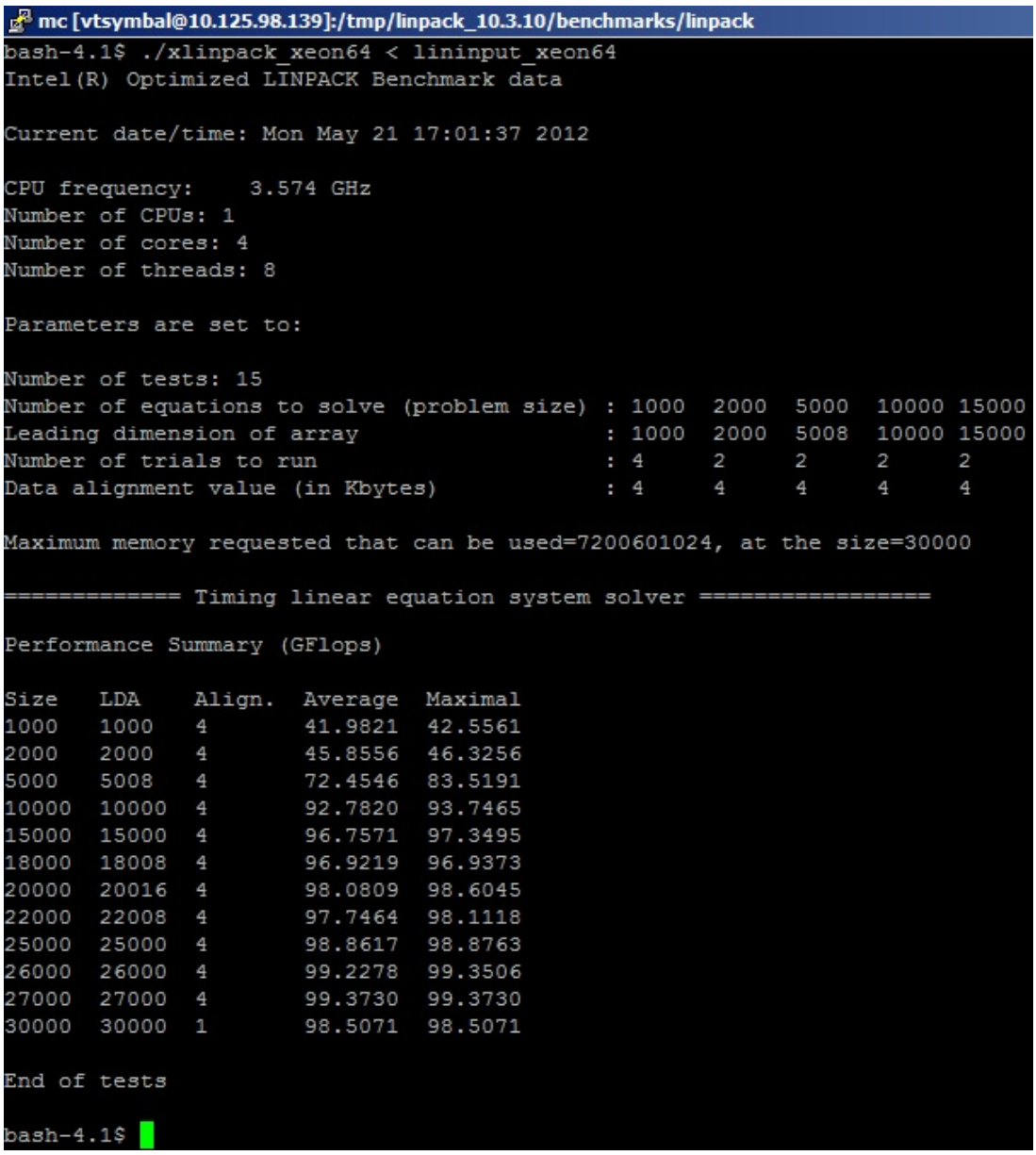
Значение FLOPS, опубликованное для конкретной системы, – это характеристика прежде всего самого компьютера, а не программы. Ее можно получить двумя способами – теоретическим и практическим. Теоретически - сколько микропроцессоров в системе и сколько исполняемых устройств с плавающей точкой в каждом процессоре. Все они могут работать одновременно и начинать работу над следующей инструкцией в конвеере каждый цикл.

Поэтому для подсчета теоретического максимума для данной системы нам нужно только перемножить все эти величины с частотой процессора – получим количество FP операций в секунду. Все просто, но такими оценками пользуются, разве что заявляя в прессе о будущих планах по построению суперкомпьютера.

Практическое измерение заключается в запуске бенчмарка Linpack. Бенчмарк осуществляет операцию умножения матрицы на матрицу несколько десятков раз и вычисляет усредненное значение времени выполнения теста. Так как количество FP операций в имплементации алгоритма известно заранее, то разделив одно значение на другое, получим искомое FLOPS.

Библиотека Intel MKL (Math Kernel Library) содержит пакет LAPACK, — пакет библиотек для решения задач линейной алгебры. Бенчмарк построен на основе этого пакета. Считается, что его эффективность находится на уровне 90% от теоретически возможной, что позволяет бенчмарку считаться «эталонным измерением».

Запускам бенчмарк из пакета Intel MKL на системе и получаем следующие результаты:



Оценка FLOPS программы

Чтобы исследовать соизмеримые результаты, в качестве нашего высокопроизводительного приложения будем использовать пример перемножения матриц.

Пример реализации перемножения матриц, написанный на языке C, можно найти в директории Samples пакета Intel VTune Amplifier XE. Воспользуемся формулой Nflop=2*(M^3) для подсчета FP операций (исходя из базового алгоритма перемножения матриц) и померим время выполнения перемножения для случая алгоритма multiply3 при размере симметричных матриц M=4096. Для того, чтобы получить эффективный код, используем опции оптимизации –O3 (агрессивная оптимизация циклов) и –xavx (использовать инструкции AVX) C-компилятора Intel для того, чтобы сгенерировались векторные SIMD-инструкции для исполнительных устройств AVX. Компилятор нам поможет узнать, векторизовался ли цикл перемножения матрицы. Для этого укажем опцию –vec-report3. В результатах компиляции видим сообщения оптимизатора: «LOOP WAS VECTORIZED» напротив строки с телом внутреннего цикла в файле multiply.c.

```
mc [vtsymbal@10.125.98.139]:~/MatrixMul/matrix/src
bash-4.1$ make icc
/opt/intel/composerxe/bin/icc -g -O3 -c ../src/util.c -D_ICC -D_LINUX
/opt/intel/composerxe/bin/icc -g -O3 -xavx -vec_report3 -c ../src/multiply.c -D_ICC -D_LINUX
../src/multiply.c(22): (col. 5) remark: loop was not vectorized: not inner loop.
../src/multiply.c(23): (col. 9) remark: loop was not vectorized: not inner loop.
../src/multiply.c(24): (col. 10) remark: loop was not vectorized: existence of vector dependence.
../src/multiply.c(49): (col. 2) remark: loop was not vectorized: unsupported loop structure.
../src/multiply.c(50): (col. 3) remark: loop was not vectorized: not inner loop.
../src/multiply.c(52): (col. 4) remark: LOOP WAS VECTORIZED.
../src/multiply.c(68): (col. 5) remark: loop was not vectorized: not inner loop.
../src/multiply.c(69): (col. 9) remark: loop was not vectorized: not inner loop.
../src/multiply.c(70): (col. 13) remark: loop was not vectorized: not inner loop.
../src/multiply.c(71): (col. 17) remark: loop was not vectorized: not inner loop.
../src/multiply.c(72): (col. 21) remark: loop was not vectorized: not inner loop.
../src/multiply.c(75): (col. 25) remark: LOOP WAS VECTORIZED.
/opt/intel/composerxe/bin/icc -g -O3 util.o multiply.o matrix.o  -o matrix.icc -lpthread -lm
bash-4.1$
```

Проверим, какие инструкции сгенерированы компилятором для цикла перемножения.

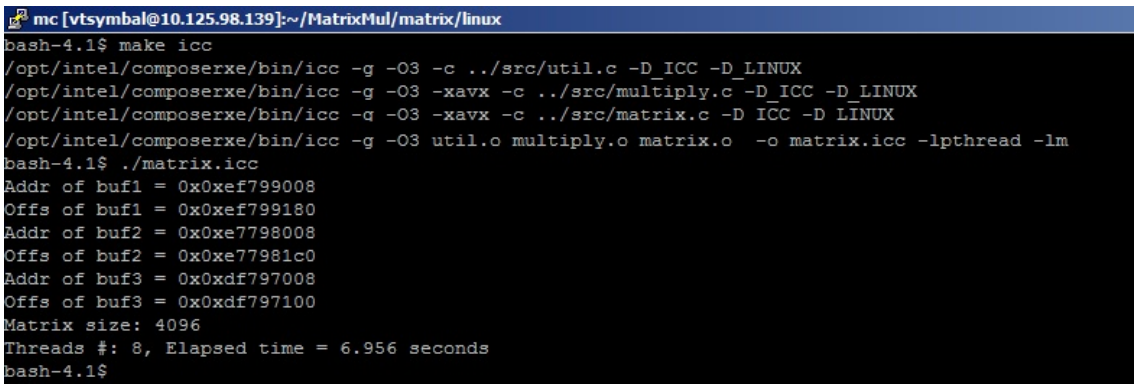
$icl –g –O3 –xavx –S

По тэгу __tag_value_multiply3 ищем нужный цикл — инструкции правильные.

$vi muliply3.s

```
                                # LOE eax edx ecx ebx ebp esi edi
..B4.15:                        # Preds ..B4.15 ..B4.14
..LN342:
   .loc    1  76  is_stmt 1
        vmovupd   32(%edx,%ecx,8), %xmm0                        #76.60
..LN343:
        vmovupd   (%edx,%ecx,8), %xmm1                          #76.60
..LN344:
        vbroadcastsd (%eax,%ebx,8), %ymm3                       #76.50
..LN345:
        vinsertf128 $1, 48(%edx,%ecx,8), %ymm0, %ymm4           #76.60
..LN346:
        vinsertf128 $1, 16(%edx,%ecx,8), %ymm1, %ymm2           #76.60
..LN347:
        vmulpd    %ymm2, %ymm3, %ymm5                           #76.60
..LN348:
        vmulpd    %ymm4, %ymm3, %ymm6                           #76.60
..LN349:
        vaddpd    (%edi,%ecx,8), %ymm5, %ymm7                   #76.60
..LN350:
        vaddpd    32(%edi,%ecx,8), %ymm6, %ymm0                 #76.60
..LN351:
        vmovupd   %ymm7, (%edi,%ecx,8)                          #76.29
..LN352:
        vmovupd   %ymm0, 32(%edi,%ecx,8)                        #76.29
..LN353:
   .loc    1  75  is_stmt 1
        addl      $8, %ecx                                      #75.25
..LN354:
        cmpl      %esi, %ecx                                    #75.25
..LN355:
        jb        ..B4.15       # Prob 82%                      #75.25
..LN356:
```

Результат выполнения программы (~7 секунд)



нам дает следующее значение FLOPS = 2*4096*4096*4096/7[s] = 19.6 GFLOPS

Задание лабораторной работы

1. В папке benchmarks\linpack надо запустить runme_xeon32.bat для x86 и x64 Windows, или runme_xeon64.bat для x64 Windows. 64-битная версия должна работать быстрее.
2. Перед запуском этот файл надо открыть в редакторе и изменить значение OMP_NUM_THREADS на полное количество ядер, реально присутствующих в компьютере, 2 для C2D, 4 для C2Q и т.д. Для Linpack 10 этого делать не надо.
3. После этого надо отредактировать файл lininput_xeon32 или lininput_xeon64 соответственно.
4. В этом файле нас интересует -

   1) # **number of tests** - количество различных размеров матриц, которые будут считатся. Должно соответствовать количеству чисел в следующей строке #problem sizes. Имеет смысл поставить просто 1 и в следующей строке указать максимальный размер который влезет в оперативную память

   2) # **problem sizes** -размеры матриц систем линейных уравнений. Чем это число больше, тем больше получится результат. Начиная с некоторого значения (примерно 10000), рост замедлится. Обьем памяти, который нужно для запуска, можно посчитать по формуле 8*N^2. Для 10000 получим 800Mb. Если задать больше чем реально установлено памяти, будет своп и тормоза -этого делать не стоит. Также стоит оставить что-то системе для работы, скажем 0.5 -1GB.

   3) **#leading dimensions** -повторить вторую строку

   4) **#times** - каждая задача с размером матрицы из соответствующего столбца **#problem sizes** будет повторена times раз.

   5) #alignment -оставить 4
5. Результат будет в файле win_xeon32.txt или win_xeon64.txt

Примечание:
Этот тест может также использоваться для проверки правильности выполнения вычислений, подобно Prime.

Для этого надо обратить внимание на значение поля Residual(norm). По моему опыту с этим тестом (под Linux), при "переразгоне" происходят 3 вещи

1) При сильном переразгоне будет BSOD

2) Далее, тест может завершится, но значение Residual(norm) будет большим, много больше 1.

3) Наконец, при совсем небольшом переразгоне значение Residual может быть маленьким, но оно будет заметно менятся от запуска к запуску -для одной и той же problem size (для разных это нормально). Поэтому, лучше оставить всего одно значение размера матрицы (максимальное), но прогнать его несколько раз (число прогонов задается в поле #times)

Пример входного файла для тестирования стабильности для 2GB памяти:

Sample Intel(R) LINPACK data file (lininput_xeon64)

Intel(R) LINPACK data

1 # number of tests

13700 # problem sizes

13700 # leading dimensions

200 #times to run a test

4 # alignment values (in KBytes)

Для 4GB вместо 13700 ставим 21000. Для 8GB -30000 и т.д.

Linpack 10 в некоторых случаях может выводить дополнительную строчку

*Error: info returned =****

Это безопасная "ошибка".

6. В отчете представить скриншоты состояния памяти, загруженности процессора перед началом теста, в течение теста, после завершения теста.
7. Представить результат файла win_xeon32.txt или win_xeon64.txt
8. Описать полученный результат, и параметры влияющие на его получение.

,