ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №1 ПО КУРСУ ОПП

1. Написать 2 программы (последовательную и параллельную с использованием MPI) на языке C/C++, которые реализуют итерационный алгоритм решения системы линейных алгебраических уравнений вида $Ax=b$ в соответствии с выбранным вариантом. Здесь A– матрица размером N×N, x и b – векторы длины N. Тип элементов – double.

2. Параллельную программу реализовать с тем условием, что матрица A и вектор b инициализируются на каком-либо одном процессе, а затем матрица A «разрезается» по строкам на близкие по размеру, возможно не одинаковые, части, а вектор b раздается каждому процессу.
Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).

3. Замерить время работы последовательного варианта программы, а также время работы параллельного при использовании различного числа процессорных ядер. Минимально на 2, 4, 8, 16, 24 (на каждом из наших узлов кластера по 12 ядер), но чем на большем количестве различного числа процессов будут выполнены замеры, тем лучше. Также чем больше замеров будет выполнено на одном и том же числе процессов, тем лучше. В этом случае для построения графиков следует брать минимальное время.
Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.

4. Выполнить профилирование двух вариантов программы с помощью jumpshot или ITAC (Intel Trace Analyzer and Collecter) при использовании 16- и или 24-х ядер.

5. На основании полученных результатов сделать вывод.

КОММЕНТАРИЙ К ПРАКТИЧЕСКОЙ РАБОТЕ №1 ПО КУРСУ ОПП

Про «идеологию высокопроизводительных вычислений»

Во время написания своих параллельных программ в этом курсе вам нужно «вжиться в шкуру» прикладного исследователя-вычислителя. Его задача состоит не в том, чтобы распараллелить, а в том, чтобы постараться максимально сократить время работы своей большой программы. И распараллеливание – это только один из инструментов ускорения программы. То есть вы должны стараться избавляться от всех лишних

вычислений, стараться экономить время работы на всем, на чем можно, распараллеливать не только самые тяжелые операции (например, умножение матрицы на вектор), а применять параллелизм везде, где это разумно. Например, для сложения/вычитания/умножения на скаляр, вычисление модулей больших векторов уместно использовать распараллеливание. Конечно нужно помнить и о тех моментах, о которых мы говорили в предыдущем семестре. Например, для того, чтобы эффективно использовать предвыборку данных в кэш, матрицу нужно стараться обходить по строкам, а не по столбцам и т.д.

То есть эффективность с точки зрения сокращения времени должна быть вашей целью.

Про работу на кластере НГУ

Головная страница: http://nusc.nsu.ru/wiki/doku.php

Обязательно почитайте раздел «Правила работы».

Запуск тяжелых вычислительных программ ТОЛЬКО через очередь: http://nusc.nsu.ru/wiki/doku.php/doc/pbs/pbs

Работа с MPI: http://nusc.nsu.ru/wiki/doku.php/doc/mpi/mpi

Работа с OpenMP: http://nusc.nsu.ru/wiki/doku.php/doc/openmp/openmp

Про профилирование

Широкую известность приобрели 2 инструмента профилирования MPI-программ: бесплатный Jumpshot и коммерческий Intel Trace Analyzer and Collector (ITAC).

Здесь расскажу про ITAC, потому как это средство обладает бОльшим функционалом и возможностями для анализа. Естественно, что лучше выбирать интеловскую реализацию MPI. Для того, чтобы воспользоваться ITAC нужно в файл ~/.bashrc вписать 2 строки, загружающие необходимые переменные окружения:

```
source /opt/intel/composer_xe_2015.2.164/bin/iccvars.sh  intel64
source /opt/intel/itac/8.1.3.037/bin/itacvars.sh
```

Вписав эти 2 строки и переоткрыв putty-терминал, можно делать так:

```
$mpiicpc  mympi.cpp –o mympi.out
```
//компиляция MPI-программы. 2 буквы «i» в слове mpiicpc не случайны

В файле описания задачи для qsub:

```
$mpirun -trace -machinefile $PBS_NODEFILE -np $MPI_NP -perhost 2
./mympi.out
```

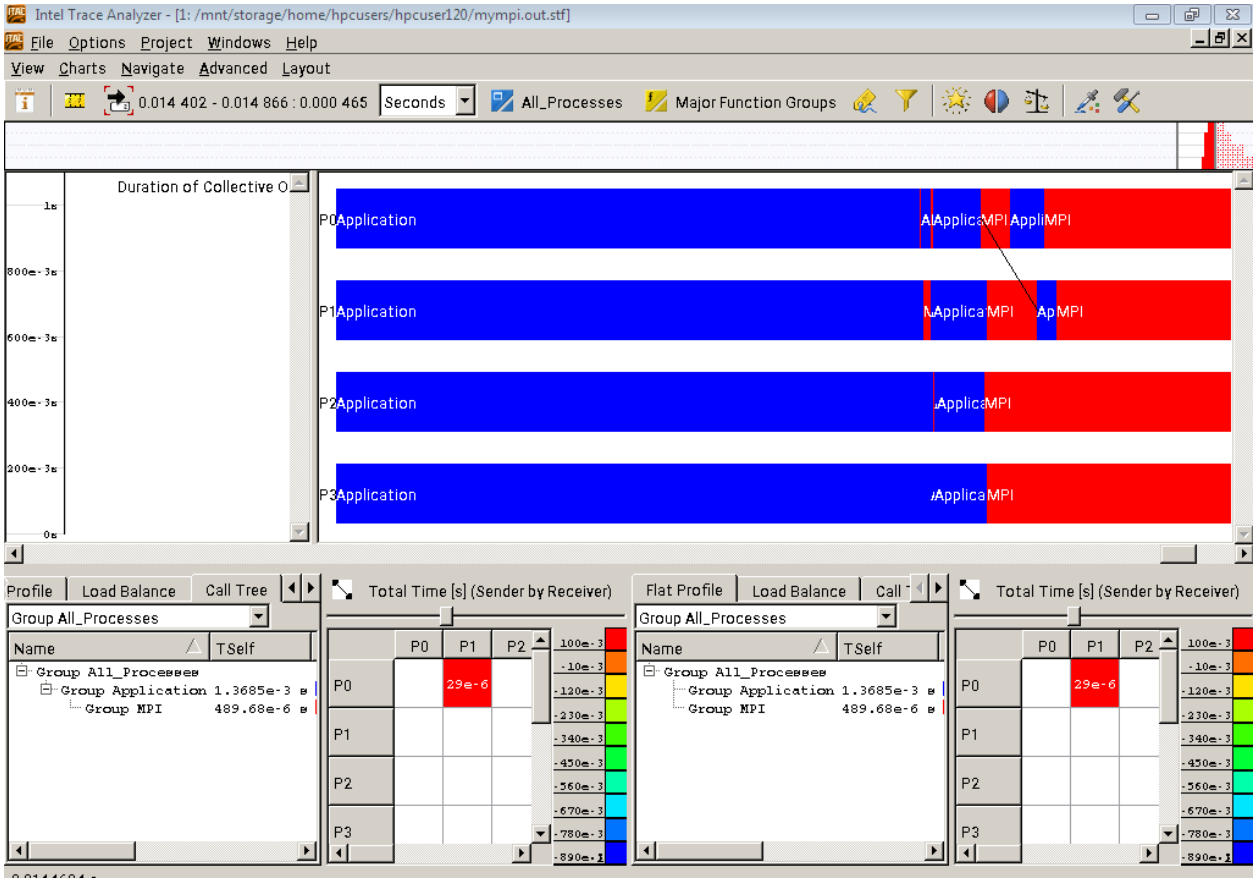
Примечание: опция «perhost» указывает по скольку процессов подряд размещать на каждом узле. То есть в случае «-perhost 2» если у нас, к примеру, 3 узла и 7 процессов, на первом узле будут процесс 0 и 1, на втором узле — 2 и 3, на третьем — 4 и 5, потом 6-й процесс снова будет размещен на 1-м узле.

Во время работы программы формируется файл трассы (mympi.out.stf). Этот файл потом можно посмотреть графической утилитой traceanalyzer. Но как мы знаем, putty предоставляет нам консольный интерфейс и для графических программ мы должны использовать какой-либо X-сервер, например Xming. Как пользоваться Xming-ом я писал вам в комментарии к практической по ЭВМ и ПУ про библиотеку OpenCV.

На кластере запускаете строку:

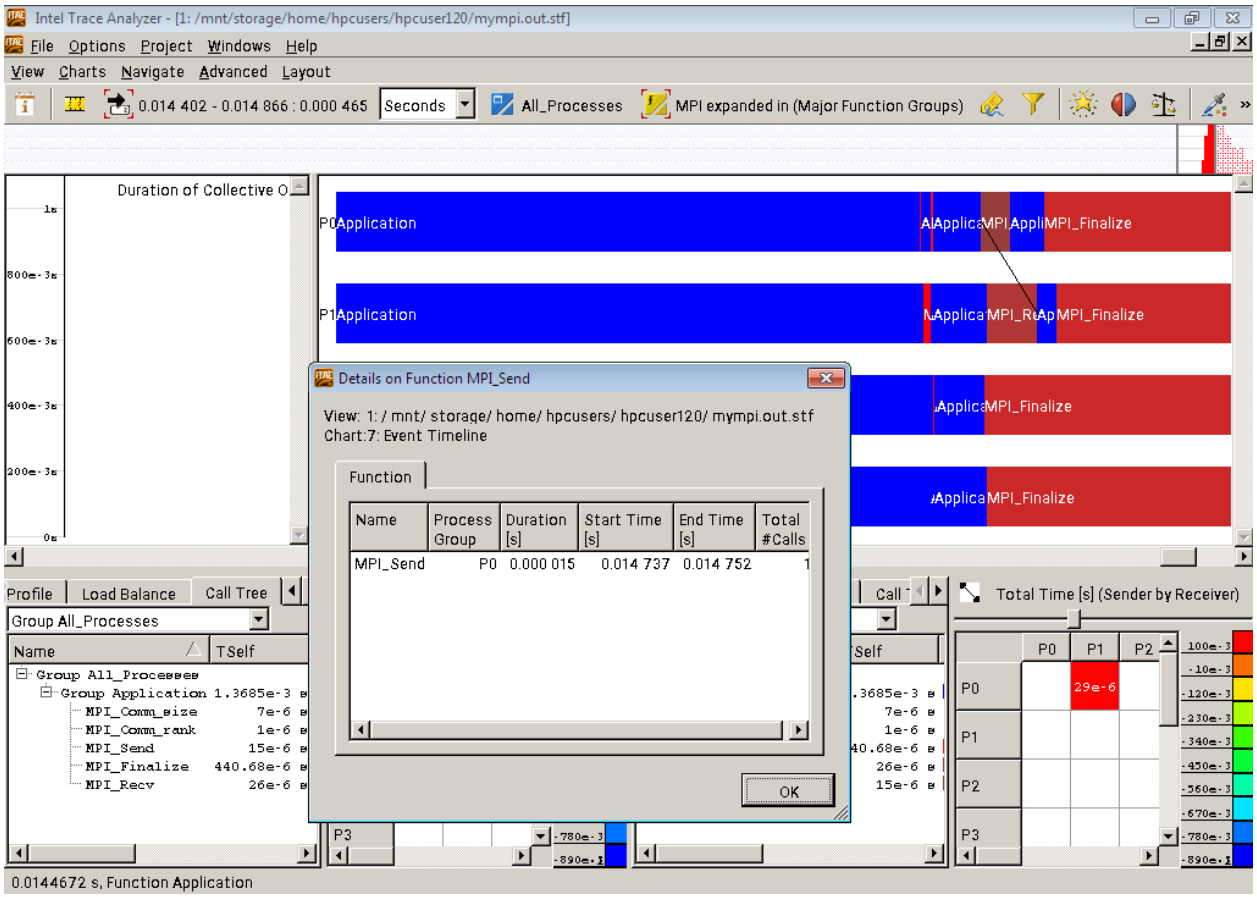
```
$traceanalyzer   mympi.out.stf
```

После чего в окне xming покажется следующая картинка:



Traceanalyzer может показать множество разных представлений, но вам достаточно продемонстрировать Event Timeline (Charts -> Event Timeline).

Для того, чтобы на таймлайне отображалось не просто слово «MPI» на MPI-функциях, а наименования конкретных операций, нужно кликнуть по одной из них правой кнопкой и выбрать «Ungroup». Тогда картина будет примерно следующей:



Сделать профилирование – это не просто предоставить картинку, а еще провести по ней анализ – на что и почему ушло много или мало времени. А также мочь ответить на вопросы.

Некоторые замечания по практической №1

1) Размерность матрицы брать не менее 1000 x 1000 (меньше не имеет смысла).

2) Изначально матрицу можно получать на одном процессе, читая из файла или генерируя в коде. НЕЛЬЗЯ раздавать матрицу полностью от этого процесса всем (например при помощи MPI_Bcast).
Рекомендую взять симметричную матрицу с утяжеленной главной диагональю (с диагональным преобладанием). Если элементы матрицы будете генерировать в отрезке [-1; 1], то утяжелять можно, к примеру, прибавлением N, где N – размерность матрицы.

3) Обязательно при замерах времени запускать последовательную программу **на кластере** и ее брать за T1 при расчетах ускорения и эффективности.

4) Мерить времена нужно строго на одном железе. То есть и последовательную и параллельную программы с разным количеством процессов – на кластере. И даже лучше всё в одном задании, чтобы точно оно отработало на одних и тех же узлах. То есть в одном задании - последовательная и параллельная программы на 2,4, 8, 16, 24 ядрах и, опционально, на другом количестве ядер в пределах от 2 до 24.

5) В MPI-программе время следует замерять при помощи функции MPI_Wtime между «засечками», первую из которых необходимо установить СРАЗУ после вызова MPI_Init, а вторую — после получения итогового вектора решения на одном процессе (имеется ввиду уже собранного полностью, а не какой-либо части). То есть после второй «засечки» может быть только проверка корректности полученного вектора решения.

6) В основном цикле программы включите счетчик итераций. Если процесс сходится за 1-2 итерации (особенно это характерно для метода сопряженных градиентов), то это абсолютно непоказательно. Ухудшите начальные условия (матрицу, вектор b или сделайте начальное приближение подальше от решения). Этот же счетчик итераций поможет определить вам, что итерационный процесс расходится (приближения не стремятся к решению). В основном цикле программы вы можете наряду с основным условием выхода (отношение определенных норм меньше эпсилон) поставить еще второе — счетчик итераций например больше 50000, поскольку за такое количество итераций ваша задача должна уже сойтись. Если не поставить это условие, а ваш процесс будет расходиться (из-за неудачно выбранного начального приближения или из-за ошибки в программе), то программа будет работать и бессмысленно тратить ресурсы машины до тех пор, пока не «свалится» или ее кто-нибудь не завершит принудительно. ОСОБЕННО это плохо, когда вы запускаете такую программу на кафедральном сервере. На кластере в каждой задаче есть предельное время, после завершения которого она принудительно снимается с выполнения,а на кафедральном сервере нет системы пакетной обработки.

7) В методе простой итерации фигурирует параметр тау, который остается постоянным в течение всей работы программы. В описании практической работы сказано, что если процесс начал расходиться, то нужно поменять этот параметр на противоположный. НО ЭТО НУЖНО ДЕЛАТЬ НЕ «НА ХОДУ», а остановив программу, поменяв знак в исходном коде и запустив программу заново.

8) Для информации — расширения AVX на кластере нет, поэтому если хотите быть большими молодцами и внедрить векторизацию, то пользуйтесь только SSE. Хотя еще раз подчеркну, что я этого не требую.

9) ОБЯЗАТЕЛЬНО встройте конктроль расходимости метода - ограничьте максимальное количество итераций основного цикла например значением 10000, иначе программа будет неограниченно работать и съедать ресурсы машины.

Про отладку MPI-программ

Компиляция с отладочной информацией:

mpiicpc –g mpiprog.cpp –o withdebug.out

Запуск одного процесса под управление gdb, а других – без:

mpiexec -n 1 gdb ./withdebug.out : -n 3 ./withdebug.out