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 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 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, как я всем вам показывал (естественно не всем, а тем, кто был на парах ☺). 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:

```
$mpiicc  mympi.cpp –o mympi.out
```
//компиляция MPI-программы. 2 буквы I в слове mpiicc не случайны

В файле описания задачи для qsub:

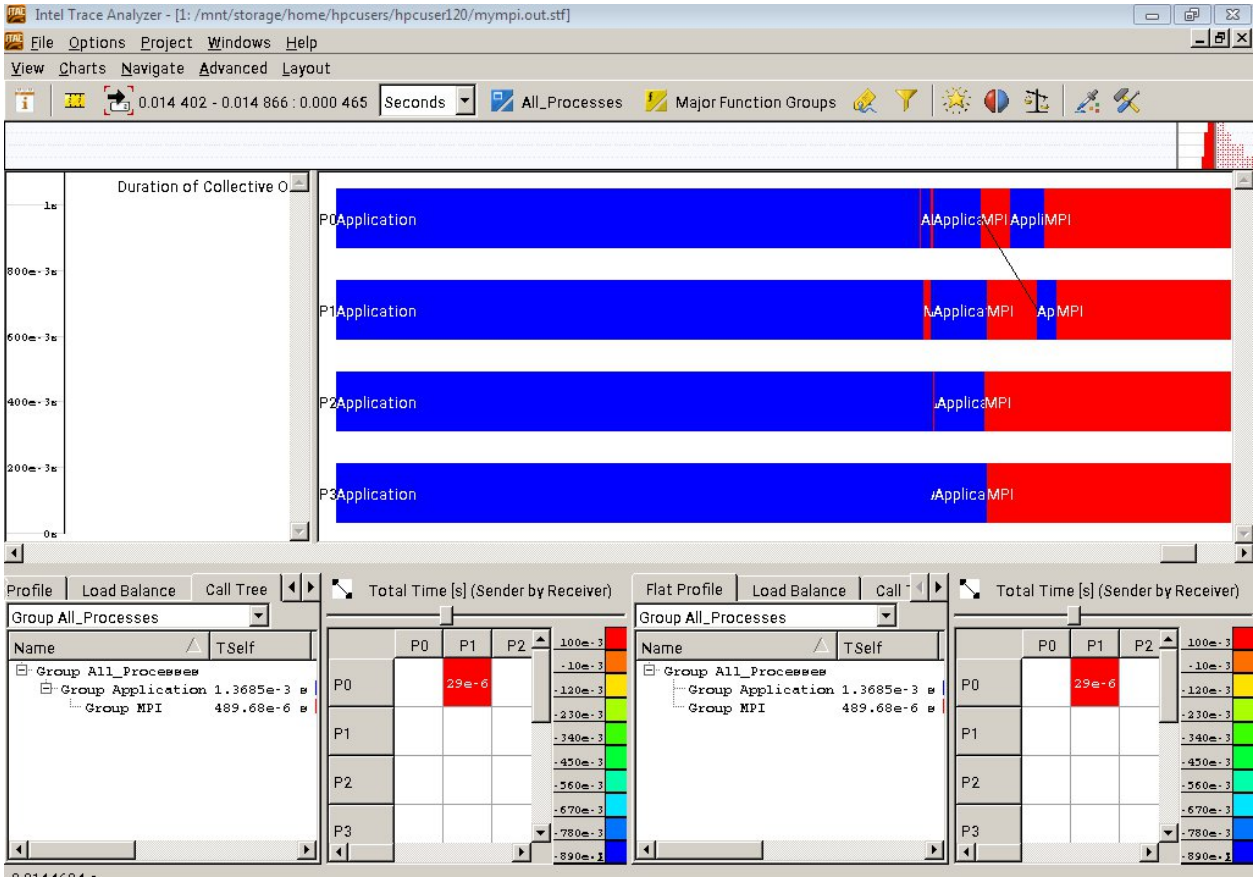
```
$mpirun -trace -machinefile $PBS_NODEFILE -np $MPI_NP ./mympi.out
```

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 5 лабораторной по ЭВМ и ПУ.

На кластере запускаете строку:

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

После чего в окне xming покажется следующая картинка:



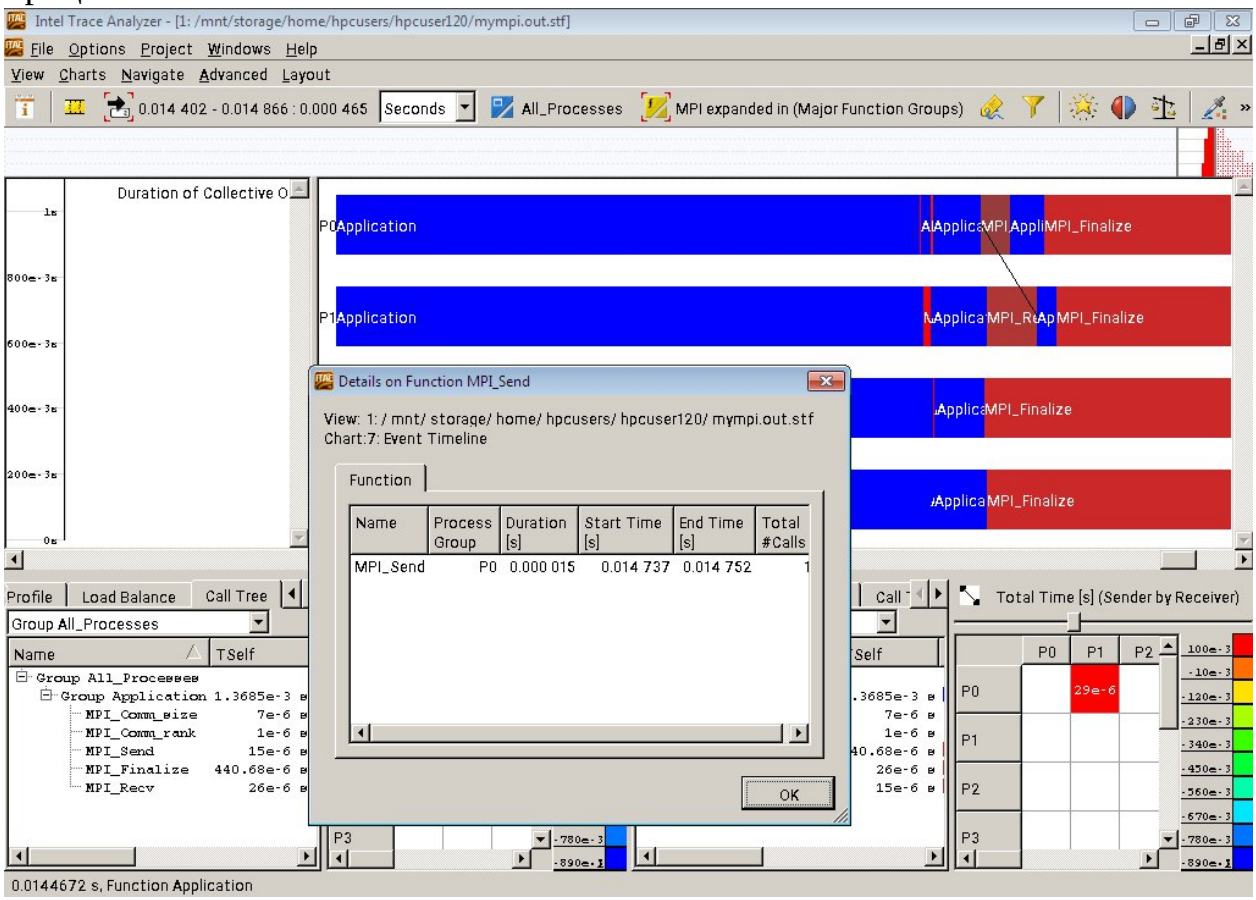
Traceanalyzer может показать множество разных представлений, но вам достаточно продемонстрировать Event Timeline (Charts -> Event Timeline).

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 лабораторной 1

1) Матрицу можно генерировать как угодно: читать одним процессом из файла, а потом разрезать ее между остальными, генерировать на каждом процессе свой кусок, генерировать всю матрицу на одном процесса, а потом разрезать ее между остальными. Не стоит только читать из файла всем процесс



ам и НЕЛЬЗЯ раздавать матрицу полностью от одного процесса всем (например при помощи MPI_Bcast).

2) Обязательно при замерах времени запускать последовательную программа на кластере и ее брать за T1 при расчетах ускорения и эффективности.

3) Мерить времена нужно строго на одном железе. То есть и последовательную и параллельные 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 2 параллельные программы на 2,4, 8, 16 и, опционально – 24 ядрах.

4) 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).

Про 2 варианта параллельной программы

При выполнении лабораторной №1 необходимо написать 3 программы: одну последовательную и 2 параллельных. Последовательную программу вы пишите без MPI, используя тот функционал языка С, который хорошо знаете с 1 семестра 1 курса. Параллельные программы, согласно задания, должны различаться тем, что в одном случае вектором *b* обладает каждый процесс, а во втором случае векторы *b* и *x* разрезаны между всеми процессами.

Естественен вопрос: чем принципиально отличается первый параллельный вариант от второго? Ведь во втором варианте вначале работы программы мы можем собрать весь вектор *b* со всех процессов и раздать полученный целый вектор всем процессам (кстати – как это сделать одной MPI-функцией?). А потом запустить тот же самый итерационный механизм, который запускали в первом варианте. Но такая схема работы второй параллельной программы меня не интересует.

Есть несколько способов решения задачи при разрезании векторов *b* и *x* между процессами. Здесь я предложу только один. Предлагаю разрезать матрицу *A* не по строкам, а по столбцам. В каждом варианте нужно вычислять разность ($Ax_n - b$). Если приглядитесь – в методе сопряженных градиентов эта разность с противоположным знаком. Давайте разберем как можно ее посчитать, если матрица *A* разрезана по столбцам, а векторы *x* и *b* распределены между процессами. При этом процессу с номером *i* принадлежат следующие столбцы матрицы и элементы векторов *x* и *b*: *i, n+i, 2*n+i*…, где *n* – число процессов и их нумерация идет с единицы (можете реализовать вариант, когда каждому процессу принадлежат элементы и столбцы с *n*i* по *n*(i+1)*).

Предположим, что у нас порядок матрицы 6 и 3 процесса. Умножение p=*Ax* (для простоты не буду писать нижний индекс) можно выполнить так:

$$p = x1\,[a11\ a21\ a31\ a41\ a51\ a61\,] + x2\,[a12\ a22\ a32\ a42\ a52\ a62\,] + x3\,[a13\ a23\ a33\ a43\ a53\ a63\,] + x4\,[a14\ a24\ a34\ a44\ a54\ a64\,] + x5\,[a15\ a25\ a35\ a45\ a55\ a65\,] + x6\,[a16\ a26\ a36\ a46\ a56\ a66\,]$$

Процессу №1 (еще раз повторю – предположим, что нумерация процессов идет с единицы) принадлежат столбцы 1 и 4 и соответствующие элементы вектора *x*. Он производит умножение своих элементов *x* на свои столбцы *A*,

после чего складывает получившиеся столбцы. Аналогичным образом поступает каждый процесс.

После этого у каждого процесса получается по одному столбцу. Для получения итогового произведения *p* эти столбцы необходимо сложить. Тут необходимо использовать функционал MPI.

Затем необходимо вычислить разницу полученного вектора *p* и вектора *b*. Но вектор *b* распределен аналогично вектору *x*. Значит нужно раздать полученный вектор *p* по процессам и вычесть соответствующие компоненты *b*. То есть наш процесс №1 в итоге посчитает:

p1-b1

p4-b4

Как совершить другие арифметические операции – догадаетесь сами ☺