УДК 004.382.2

ИССЛЕДОВАНИЕ МАСШТАБИРУЕМОСТИ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТОВ АНАЛИЗА ПАРАЛЛЕЛЬНЫХ ПРИЛОЖЕНИЙ НА ПРИМЕРЕ МОДЕЛИ АТМОСФЕРЫ NH3D[1]

А.С. Антонов, А.М. Теплов

Рассмотрен подход к исследованию масштабируемости параллельных приложений с использованием средств анализа их работы. Для изучения масштабируемости в описанной методике используется наряду с профилированием приложения анализ его узких мест с помощью трассировщика. Приведен краткий обзор показателей эффективности работы параллельной программы, обзор подходов к изучению масштабируемости параллельных программ и инструментов для исследования параллельных приложений; описание методики исследования масштабируемости программы, а также детальное описание программы, которая использовалась для отработки методики. Приведены результаты исследования описанной параллельной программы с использованием профилирования работы и изучения трассировщиком. Для этих целей в качестве трассировщика был выбран Intel Trace Analyzer and Collector. В заключительной части сделаны выводы о применимости использованных инструментов анализа работы параллельных приложений для исследования масштабируемости.

Ключевые слова: масштабируемость, анализ эффективности, трассировка, профилирование, инструментарий для анализа параллельных приложений.

Введение

Параллельное программирование всегда связано с рядом сложностей, с которыми не сталкиваются программисты, создающие последовательные программы. Но только параллельные вычисления позволяют добиваться максимальной производительности, которая так необходима во многих современных научных расчетах. Современный численный эксперимент, проводимый на огромном суперкомпьютере, генерирует огромные объемы данных и может проходить до нескольких суток.

Вопрос рационального использования вычислительных систем, способных справиться с такими вычислениями, стоит очень остро. Он напрямую связан с масштабируемостью – свойством параллельных программ, характеризующим зависимость показателей эффективности работы параллельной программы от числа использованных процессоров. В данной статье описана методика исследования масштабируемости параллельной программы с использованием инструментов для анализа параллельных приложений и использовании полученных данных для построения картины масштабируемости приложения на примере атмосферной модели NH3D.

1. Масштабируемость параллельных программ

Для оценки качества параллельной программы введен ряд показателей [1], например:

Ускорение (speedup) $S = T_1/T_p$, где T_p — время исполнения распараллеленной программы на p процессорах, T1 — время исполнения исходной последовательной программы.

[1] Статья рекомендована к публикации программным комитетом Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: поиск новых решений – 2012».

Вычислительная сложность задачи W — количество основных вычислительных шагов лучшего последовательного алгоритма, необходимых для решения задачи на одном процессоре. W является некоторой функцией от размера входных данных программы. Если для простоты предположить, что каждый основной вычислительный шаг выполняется за единицу времени, то получим $W = T_1$.

Эффективность $E = S/p$ определяет среднюю долю времени выполнения параллельного алгоритма, в течение которого процессоры реально используются для решения задачи.

Стоимость (cost) вычислений $C = pT_p$.

$T_0 = pT_p - T_1$ — суммарные накладные расходы (total overhead). При увеличении p значение, как правило, возрастает.

Если T_1 фиксировано, то при увеличении числа процессоров p эффективность E, как правило, уменьшается за счет роста накладных расходов T_0. Получение большого ускорения за счет большого числа процессоров обычно приводит к снижению эффективности. А если фиксировано число процессоров p, то эффективность E зачастую можно повысить, увеличивая вычислительную сложность решаемой задачи W (а значит, и время T_1).

Отношение ускорения или эффективности к объему используемых ресурсов или размеру задачи характеризует то, насколько рационально программа способна использовать параллельную вычислительную систему. Для этого вводится понятие масштабируемости параллельной программы. Масштабируемость – свойство программы, определяющее зависимость её ускорения или эффективности, получаемых на вычислительной системе, от объема использованных для этого ресурсов и размера задачи. На основании данных о масштабируемости можно сделать выводы о необходимости оптимизации программы и причинах возникающих проблем.

Масштабируемость параллельных программ можно подразделить на несколько основных типов:

• Сильная масштабируемость (strong scaling) — зависимость производительности от количества процессоров p при фиксированной вычислительной сложности задачи ($W = const$).

• Масштабируемость вширь (wide scaling) – зависимость производительности от вычислительной сложности задачи W при фиксированном числе процессоров (p=const)

• Слабая масштабируемость (weak scaling) — зависимость производительности от количества процессоров p при фиксированной вычислительной сложности задачи в пересчете на один узел ($W/p = const$).

2. Подходы к изучению масштабируемости

Существует несколько подходов к изучению масштабируемости программ. Методы исследования масштабируемости по способу получения данных можно разделить на две группы: аналитические [2, 4] и эмпирические [3].

Эмпирический подход – это анализ работы реальной программы, основывающийся на данных, полученных при многократном выполнении программы на конкретной вычислительной системе. Этот подход используется наиболее часто.

Аналитический подход – это получение необходимых характеристик программы с помощью моделирования хода её выполнения по заданному принципу на менее мощной вычислительной системе и построения прогноза масштабируемости данной программы на целевой вычислительной системе.

Аналитический подход подразумевает один из видов моделирования:

– алгебраический анализ;

– синтаксический разбор исходного текста программы;

– абстрактная интерпретация;

– симулирование выполнения.

При комбинировании методов эмпирического и аналитического исследования получают так называемые смешанные методы исследования масштабируемости.

Из смешанных методов выделяют квазианалитический метод, основанный на моделировании объема вычислений на каждый узел и запуске на целевой системе. Такой подход требует большего внимания к моделированию коллективных операций. Это связано с тем, что операции типа точка-точка на целевой конфигурации компьютера будут выполняться столько же времени, а время выполнения коллективных обменов будет зависеть от числа реально участвующих процессов.

Построение модели программы возможно только при условии полного и детального знания алгоритма работы и реализации. Для построения качественной модели требуется использование специального инструментария и аккуратный перенос на модель всех деталей работы программы.

3. Категории инструментов для исследования эффективности и масштабируемости параллельных приложений

Существующие инструменты для исследования масштабируемости программ можно разделить на несколько групп:

• инструменты для анализа показателей метрик эффективности работающей программы (профилировщики, трассировщики, средства пакетного запуска, средства для инструментирования кода);

• инструменты для моделирования работы реального приложения (утилиты, моделирующее время выполнения на основе реальных трасс, исходного кода, ручного сбора данных, других моделей);

• инструменты для моделирования потенциальной масштабируемости разрабатываемого приложения (моделируют масштабируемость разрабатываемого приложения на предполагаемой архитектуре).

Профилирование – процесс получения данных, которые обобщают поведение программы: сколько раз была вызвана некоторая функция; время выполнения функции; дерево или граф вызовов в программе; значения аппаратных счетчиков и т.п. Таким образом, профилирование помогает определить «узкие места» и «горячие точки» в программе.

Профилирование может реализовываться через:

• семплирование – периодические прерывания ОС или прерывания по значению аппаратных счетчиков;

• инструментирование – вставка в программу дополнительного кода, измеряющего производительность.

Сбор трасс подразумевает запись потока событий: вход/выход в какой-то участок кода (функцию, цикл и др.); взаимодействие процессов (нитей) — прием/передача сообщений и т.д., запись того, в каком процессе/нити и когда случилось каждое событие.

Трасса позволяет анализировать производительность и корректность программы с точки зрения того, какие процессы происходят в каждый конкретный момент её выполнения.

На данный момент отсутствует инструмент, позволяющий анализировать масштабируемость приложения. Поэтому для исследования необходимо использовать сразу нескольких инструментов, которые бы позволили проводить анализ. В данной статье рассматривается связка профилировщика и трассировщика. Профилировщик необходим для определения участков кода с проблемами масштабируемости. Трассировщик необходим для установления причин проблем и путей возможного решения. Такая связка инструментов позволяет не только получить данные о масштабируемости как всего приложения, так и его участков, но и данные об особенностях его работы, а также рекомендации по эффективному запуску приложения.

Существует довольно большой набор инструментов, позволяющих проводить различные типы анализа работы приложения. Профилировку приложения можно проводить как вручную, измеряя время работы отдельных участков кода программы прямо в коде, так и любыми специализированными профилировщиками. Из множества профилировщиков хотелось бы выделить класс профилировщиков, способных собирать данные, не зависящие от текущей архитектуры, на которой выполняется приложение. Профилировщики такого класса позволяют по собранным данным подобрать оптимальную архитектуру для выполнения приложения. К таким инструментам можно отнести Rogue Wave Threadspotter. Существуют также и инструменты, в которых присутствует как профилировщик, так и трассировщик с визуализатором результатов анализа. К таким инструментам можно отнести TAU Toolkit. С его помощью можно получить больше информации о работе приложения. Также стоит выделить такие инструменты как Vampir, Scalasca, Periscope и Score-P, которые интегрируются как с TAU так и друг в друга и дополняя свои результаты исследования данными другого контекста. Данные инструменты позволяют при их совместном использовании провести комплексный анализ работы параллельного приложения и не только выявить места его наименьшей эффективности, но и определить наиболее вероятные причины проблем.

Для работы был выбран инструмент для трассировки и анализа приложений Intel Trace Analyzer and Collector, хотя для этих целей подходит практически любой трассировщик со сходным функционалом.

Intel Trace Analyzer and Collector – инструмент, позволяющий проводить сбор трасс с последующим анализом визуализированной трассы. Он состоит из двух основных составляющих: Trace Collector, осуществляющий сбор трасс, и Trace Analyzer, осуществляющий визуализацию и анализ. Поддерживается сбор трасс параллельных приложений на языках C/C++, Java, Fortran, использующих MPI, явные и неявные нити (OpenMP) или гибридную модель (MPI+OpenMP). Существуют версии для Linux и Microsoft Windows для архитектур IA32, Intel64.

4. Описание методики исследования масштабируемости

Исследование масштабируемости модели проходило в несколько этапов.

На первом этапе производился сбор данных о работе приложения и о том, какие данные могут быть использованы для оценки скорости работы. Для этого по выбранным критериям собирались данные о работе приложения в начальном виде. Производился запуск задачи фиксированного размера на разном количестве используемых процессоров. Это необходимо для определения начальной картины масштабируемости приложения.

Полученные данные говорят о наличии проблем масштабируемости и необходимости более глубокого анализа.

Измерялось время выполнения различных частей программы. Изначально в силу итерационной структуры работы алгоритма обработки данных был предложен метод измерения числа выполненных итераций за фиксированное время, но от него пришлось отказаться из-за того, что на этапе инициализации модели время выполнения различных процедур отличалось довольно сильно. Это влияло на число выполненных итераций, и потому более точным оказалось измерение среднего времени итерации отдельно от времени выполнения инициализации.

Исследуемая программа запускалась на разных конфигурациях вычислительной системы. Начиная с минимальной конфигурации число использованных процессоров пошагово увеличивалось.

После каждого запуска анализировались собранная статистика работы приложения. Данные фиксировались в логе работы. Из полученных данных о времени работы вычисляли ускорение работы и эффективность. На следующем этапе приложение инструментировалось для профилирования и определения ускорения различных частей кода. Проводилась еще одна серия запусков, в которой собирались данные о работе отдельных частей программы. Те участки, время которых росло с ростом числа процессоров, рассматривались более детально на предмет причин проблем масштабируемости и путей повышения эффективности.

При использовании Intel Trace Analyzer and Collector для углубленного анализа, данные собирались схожим образом. После каждого запуска, собранная трасса конвертировалась и архивировалась для дальнейшего анализа.

По времени выполнения отдельных частей программы вычислялись их характеристики, такие как ускорение и эффективность работы.

5. Описание модели NH3D

Исследование масштабируемости параллельного приложения проводилось на примере трехмерной гидротермодинамической негидростатической модели атмосферы [5], разрабатываемой в НИВЦ МГУ имени М.В. Ломоносова.

Программа написана на языке Фортран с использованием технологии параллельного программирования MPI. Общий объем кода составляет примерно 75 тысяч строк. В основе параллельной реализации модели для многопроцессорных систем с распределенной памятью лежит двумерное разбиение трехмерной расчетной области.

Модель построена на пошаговом преобразовании данных. Почти все время выполнения программного кода приходится на цикл интегрирования уравнений модели по времени main_loop в головной программной единице nh3d_main. Подпрограммы, вызываемые в этом цикле и реализующие компоненты численного алгоритма, не имеют логических ветвлений и выполняются детерминировано.

Такая структура программы позволила существенно упростить анализ производительности и профилирование, потому что на каждом шаге модели выполняются идентичные операции. Для уменьшения объемов собираемой информации число шагов модели было сведено к минимуму, который давал достаточную статистику работы.

Цель исследования заключалась в оптимизации ключевых процедур исходного кода приложения, позволяющей добиться повышения масштабируемости этих процедур на

больших конфигурациях вычислительной системы. Работа состояла из нескольких этапов классической схемы оптимизации приложения:

1. Профилирование приложения с выделением узких мест (процедур) при использовании большого количества вычислительных ядер. Профилирование проводилась на двух уровнях. На первом уровне диагностики отдельно оценивалось время выполнения приложения в целом и процедур библиотеки MPI. Минимальной конфигурацией была выбрана конфигурация с 4 процессорами. На меньших конфигурациях работали другие логические ветки кода. Для этого использовался пакет ITAC (Intel Trace Analyzer and Collector). На втором уровне производилась оценка времени выполнения отдельных процедур приложения. Для этих целей проводилось несколько типов профилирования. По результатам профилирования выбирались процедуры, время выполнения которых становилось критичным для времени выполнения одного шага модели в целом. При выборе процедур принимались во внимание особенности структуры кода, численные алгоритмы и организация MPI-обменов в данном приложении. На основе собранных данных были выявлены наиболее узкие места работы приложения и последующее более детальное инструментирование исходного кода для анализа с помощью Trace Analyzer.

2. Оптимизация выбранных процедур проводилась, исходя из полученных данных на всех этапах анализа. Так, на первом этапе анализа была выявлена проблема с выводом в файл. Модель регулярно записывала результаты в файл используя только один процесс. Время вывода было существенным даже на небольших конфигурациях вычислительной системы и не уменьшалось с ростом числа процессоров. Поэтому на больших конфигурациях время вывода в файл составляло существенную долю времени работы всей модели. Регулярность вывода не была критичной для работы алгоритма, и потому вывод в файл остался в конце всей работы, также планируется переход на параллельный вывод данных.

Второй оптимизацией, основанной на результатах профилирования, стало использование модернизированного варианта двух процедур, используемых на каждом шаге модели. Профилирование показало, что на больших конфигурациях вычислительной системы основу времени каждого шага составляло время выполнение двух процедур: MOMENT_MPI и THERMO_MPI. Процедура MOMENT_MPI решает три уравнения движения, а процедура THERMO_MPI — уравнение притока тепла и уравнения переноса излучения в атмосфере. Ускорение обеих процедур переставало расти при увеличении числа процессоров больше 25. На конфигурациях около 100 процессоров время увеличилось в 10 раз для процедуры MOMENT_MPI и в 40 раз для процедуры THERMO_MPI по сравнению со временем их выполнения на 25 процессорах. Более детальное профилирование показало, что процедуры используют адаптацию одного и того же алгоритма усреднения данных на границе области SPONGE, что и приводило к таким результатам. Отключение его использования в настройках работы модели позволило существенно уменьшить время одного шага модели и масштабируемость всего приложения в целом. После этого ускорение шага модели продолжало расти до конфигураций вычислительной системы, содержащих в полтора раза больше процессоров.

6. Исследование модели NH3D с помощью ITAC

При исследовании модели NH3D использовался пакет ITAC. На первом этапе исследования анализ трассы позволил исследовать общий характер поведения параллельной программы на вычислительной системе. В собранной трассе без пользовательского инструментирования исходного кода программы отображались два класса функций: время,

затраченное на вызовы функций MPI, и время выполнения кода приложения. Даже без дополнительного инструментирования исходного кода приложения можно определить ряд узких мест работы программы. Визуально легко определить наличие последовательных участков программы. Такие участки соответствуют выводу программы в файл, а также процессу инициализации данных приложения. Они ухудшают масштабируемость всего приложения в целом, потому что не получают никакого ускорения на любом масштабе вычислительной системы. На собранной трассе можно установить, какие функции MPI увеличивают разбалансировку работы приложения. На рис. 1 показана трасса работы приложения, где работу выполняет только один процесс. На рис. 2 показана трасса работы приложения, где работа процессов разбалансирована.

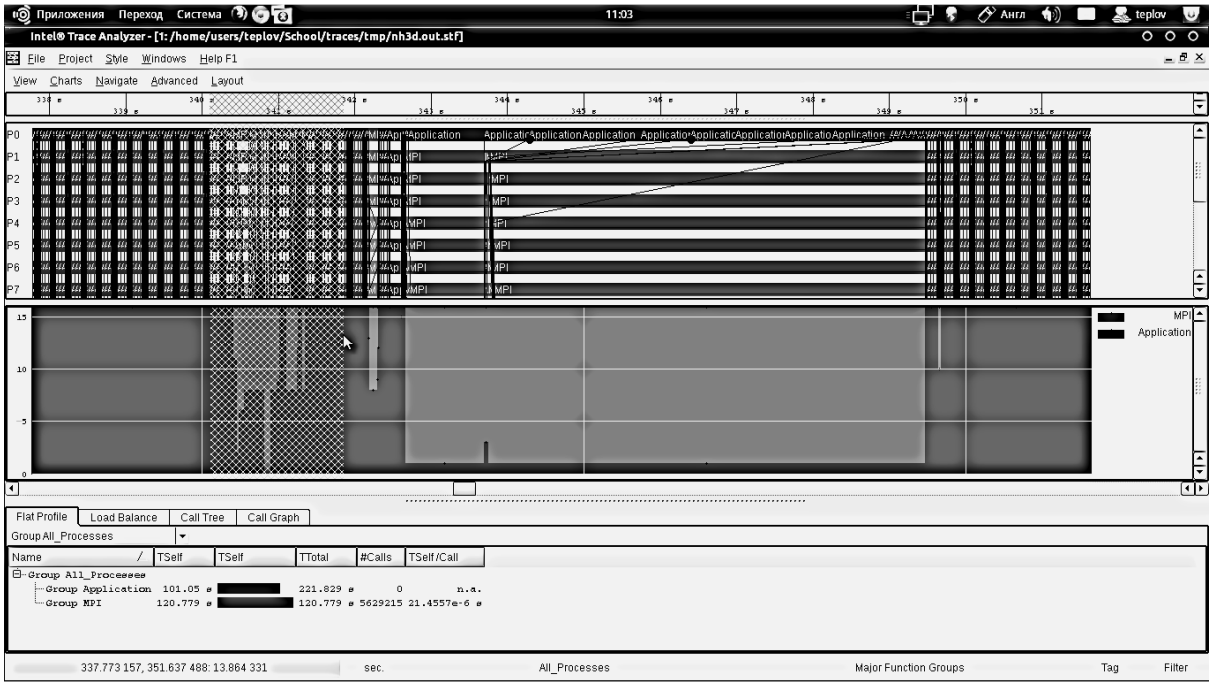

Рис 1. Пример участка трассы работы приложения, на котором работу выполняет только один процесс

Для выявления потенциальных узких мест удобно пользоваться количественной диаграммой (на рисунках находится под диаграммой событий). На ней отображается число процессов, участвующих в выполнении каждой операции. Идеальная картина – все процессы заняты одной и той же деятельностью. Если наблюдается «лестница», то этот участок не оптимален. Таких мест в модели NH3D достаточно много.

На втором этапе анализа было определено расположение процедур, замедляющиеся с увеличением числа процессоров. Для выяснения причин отсутствия ускорения проводилось инструментирование исходного кода программы с помощью API, предоставляемого ITAC.

В коде были выделены два класса функций: функции инициализации и функции шага по времени. Для такого разделения необходимо передавать параметр идентификатор вызывающего класса функций.

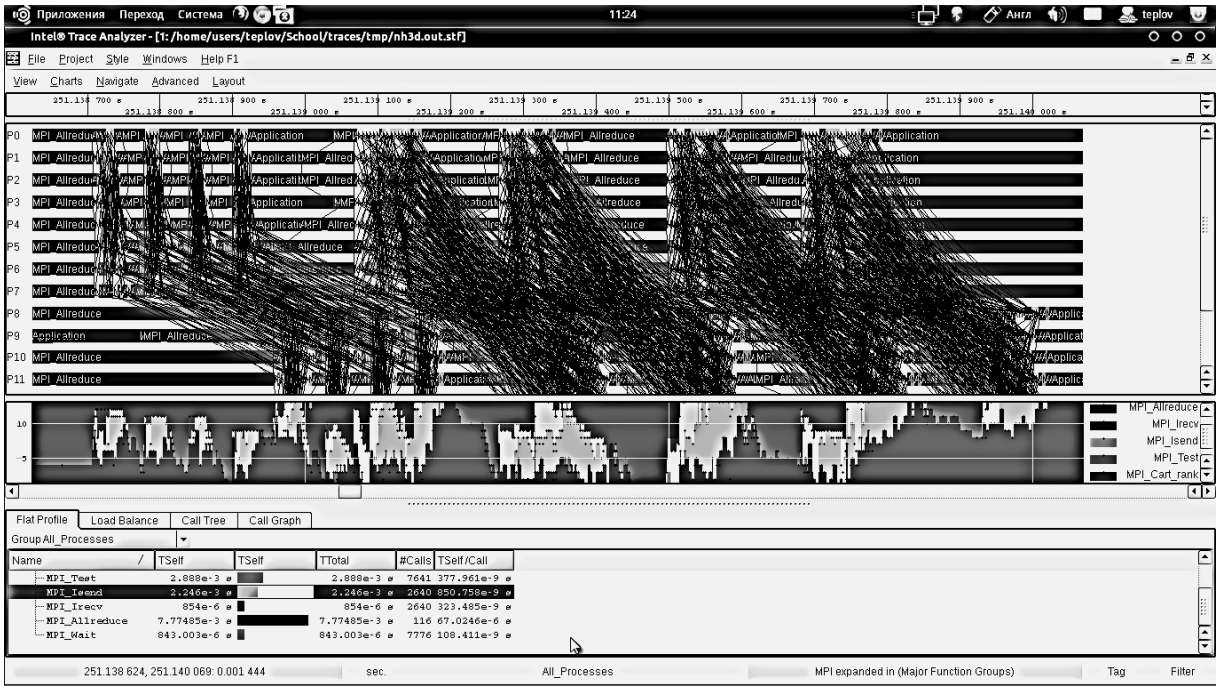

Рис 2. Пример участка трассы приложения,
на котором работа процессов разбалансирована

Сбор трасс проводился с использованием алгоритма усреднения данных на границе области SPONGE и без него. На рис. 3 показана трасса работы процедуры MOMENT_MPI на 25 процессорах с использованием алгоритма SPONGE, а на рис. 4 – без его использования.

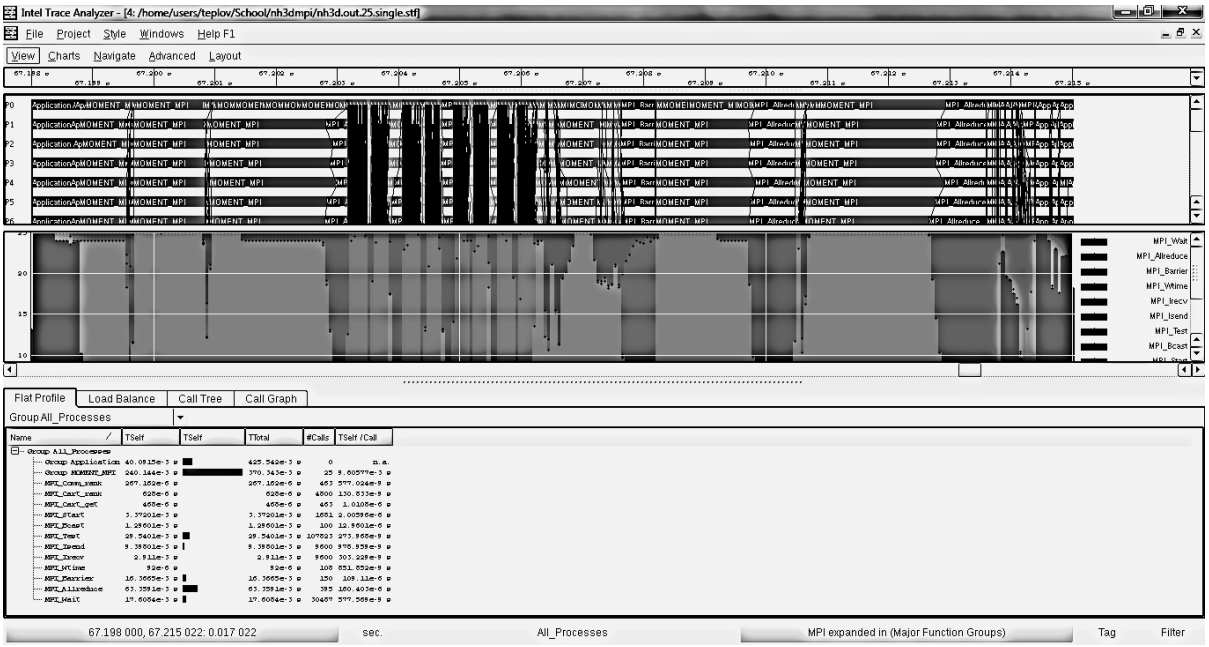

Рис 3. Участок трассы программы,
на котором процедура MOMENT_MPI использует алгоритм SPONGE

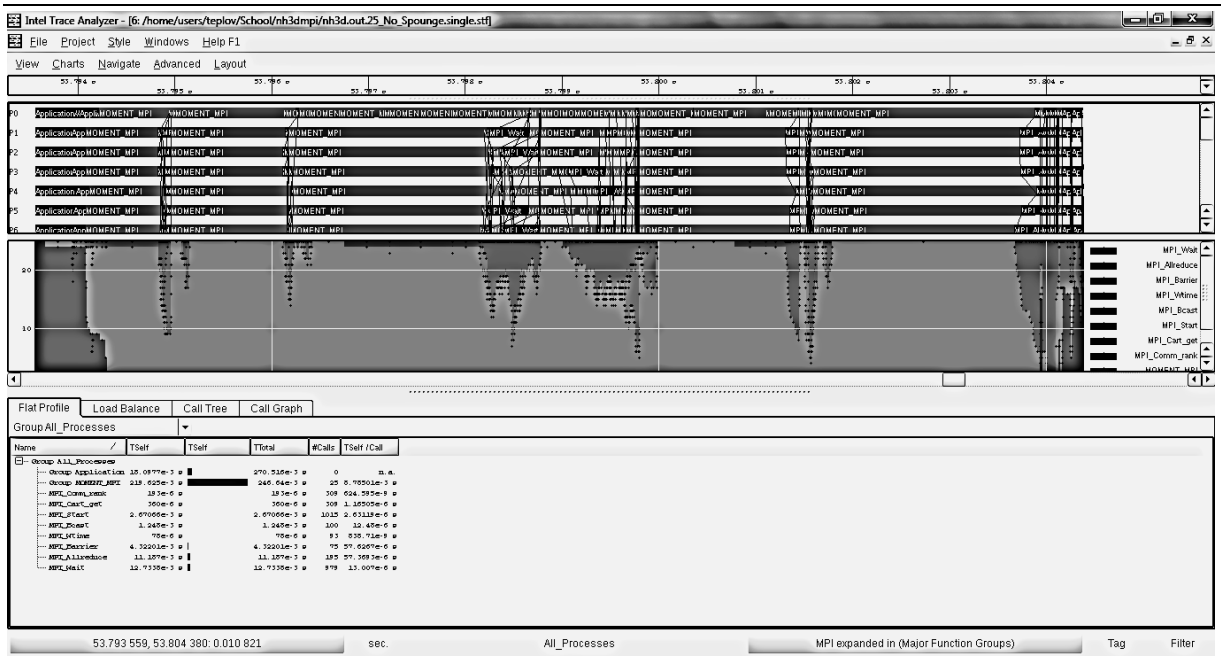

Рис 4. Участок трассы программы,
на котором процедура MOMENT_MPI не использует алгоритм SPONGE

При сравнении двух трасс легко заметить отличия в характере профиля работы процедуры и влияние процедуры SPONGE_MPI. Таким образом, можно заключить, что SPONGE_MPI является узким местом, сильно замедляющим работу приложения. На следующем этапе работы проводилось более глубокое инструментирование шагов модели по времени. На трассу выводились данные обо всех процедурах, которые используют вызовы процедур MPI. Основную сложность при инструментировании вызвали вложенные процедуры, которые используются в нескольких процедурах. Необходимо корректно учитывать отображение на трассе времени выполнения кода вложенных процедур (run time) и кода самой процедуры (self time) всех процедур. Если основу времени выполнения процедуры занимает выполнение вложенной процедуры, то больше внимания при оптимизации необходимо уделить вложенной процедуре.

В ходе такого типа анализа была выявлена процедура UPUVT_MPI, которая вызывается из многих процедур, и потому занимает существенную часть времени шага модели. Работа процедуры была проанализирована на основе собранных трасс и оптимизирована, исходя из результатов исследования.

7. Данные о масштабируемости приложения, полученные с помощью ITAC

Целью работы являлся анализ масштабируемости приложения, и потому данные, предоставляемые инструментами исследования параллельных приложений, должны быть рассмотрены в совокупности всех проведенных экспериментов. Представление о масштабируемости приложения возможно получить только после профилирования всех процедур и получения сводной статистики серии экспериментов.

Анализ статистических данных о метриках эффективности параллельного приложения, вне зависимости от выбранной метрики и способа сбора данных, позволяет сделать выводы о наличии проблем масштабируемости приложения. Для данной задачи использование ITAC позволило быстро собрать статистические данные экспериментов.

После определения наличия проблемы для выявления её места в коде использовалась количественная диаграмма. По ней определялись участки кода, в которых происходит разбалансировка работы.

Более детальный поиск проблем с помощью инструментирования исходного кода позволил оценить вклад каждой процедуры в масштабируемость всего приложения и выявить наиболее критичные. После этого статистика о работе выделенного участка собиралась в сводную картину масштабируемости всего приложения и его частей.

После локализации узких мест необходимо сделать выводы о причинах плохой масштабируемости в найденных участках. Узкие места более детально исследовались на диаграммах событий. Изучая в совокупности диаграммы событий (рис 5.A) и качественные диаграммы (рис 5.B), делались выводы о том, какие вызовы MPI приводят к разбалансировкам, а также каков их вклад. Для анализа данных серии экспериментов возможностей анализатора по сравнению различных трасс было недостаточно. Сравнение двух трасс не давало полноценной картины масштабируемости. Однако, сравнивая минимальные и максимальные конфигурации по числу использованных процессоров, можно получить представление о том, как сильно меняется профиль работы.

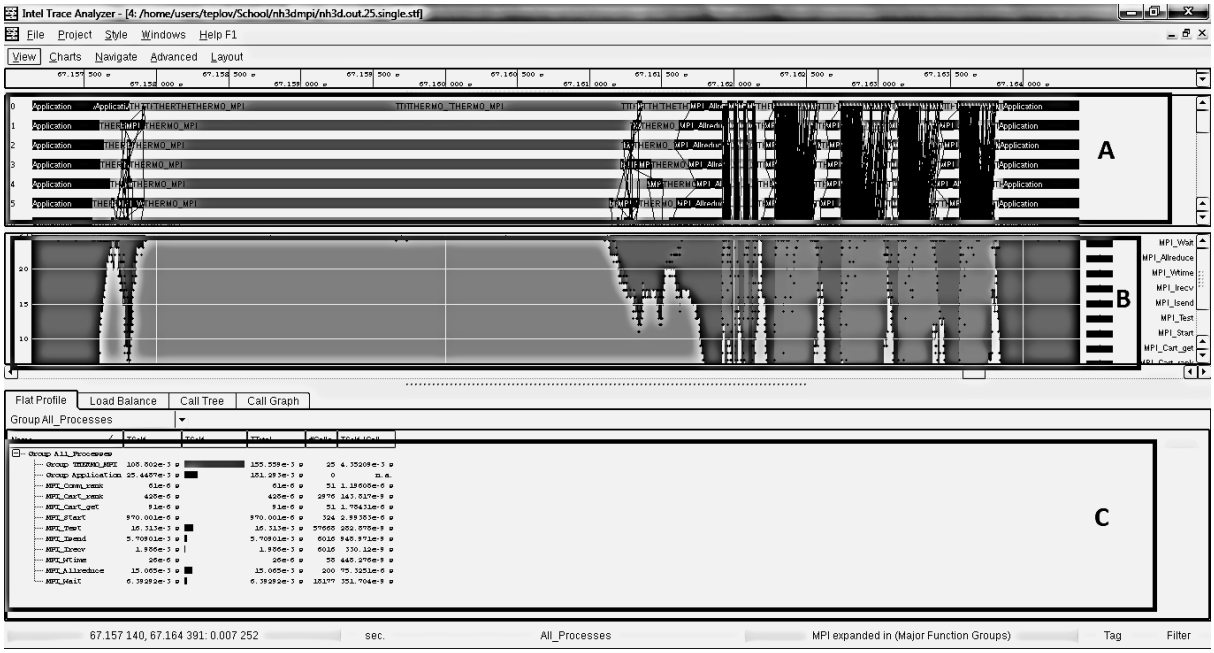

Рис 5. A – диаграмма событий. B – качественная диаграмма.
C – профиль отображенного сегмента трассы

Другой проблемой стал поиск узких мест при больших масштабах вычислительной системы. Общее число отображаемых событий можно сократить, применяя фильтрацию. При таких масштабах визуализацию диаграммы событий невозможно уместить на одном экране, поэтому анализ затруднен в силу отсутствия автоматического определения проблем эффективности в данном инструменте.

Инструмент не позволил выявить проблемы распределения данных. Анализатор разбалансировки приложения предоставляет также данные об объемах и количестве коммуникационных сообщений.

Заключение

Делая выводы об использовании ITAC в целях исследования масштабируемости приложения, нужно отметить, что данный инструмент предоставляет достаточно данных для того, чтобы сделать однозначные выводы о существовании проблем масштабируемости. Возможностей инструмента достаточно для локализации проблем, а также получения представления о масштабируемости отдельных частей приложения. При детальном анализе узких мест с помощью ITAC можно получить представление о характере проблем исходного кода. Инструмент может анализировать трассы работы на больших конфигурациях вычислительной системы. И потому, несмотря на сложность анализа трасс с большим количеством процессоров, ITAC предоставляет достаточно информации для того, чтобы определить наличие проблем масштабируемости, локализовать узкие места программы и получить представление о характере проблемы.

Работа выполняется при поддержке гранта РФФИ 10-07-00586-а.

Литература

1. Grama, A. Introduction to Parallel Computing. (2nd Edition) / A. Gupta, G. Karypis, V. Kumar. – Pearson, 2003.
2. Иванников, В.П. Оценка динамических характеристик параллельной программы на модели / В.П. Иванников, С.С. Гайсарян, В.А. Падарян // Программирование. – 2006. – № 4.
3. Alabdulkareem, M. Scalability Analysis of Large Codes Using Factorial Designs / M. Alabdulkareem, S. Lakshmivarahan, S.K. Dhall // Parallel Computing. – 2001. – Vol. 27, Issue 9. – P. 1145–1171.
4. Muller-Wichards, D. Scalability of Algorithms: An Analytic Approach / D. Muller-Wichards, W. Ronsch // Parallel Computing. – 1995. – Vol. 21, Issue 6. – P. 937–952.
5. Степаненко, В.М. Численное моделирование мезомасштабной динамики атмосферы и переноса примеси над гидрологически неоднородной территорией / В.М. Степаненко, Д.Н. Микушин // Вычислительные технологии. – 2008. – Т. 13, специальный выпуск 3. – С. 104–110.

Антонов Александр Сергеевич, к.ф.-м.н., старший научный сотрудник, НИВЦ МГУ имени М.В. Ломоносова, asa@parallel.ru.

Теплов Алексей Михайлович, младший научный сотрудник, НИВЦ МГУ имени М.В. Ломоносова, alex-teplov@yandex.ru.

APPLICATION SCALABILITY STUDY USING TOOLS TO ANALYZE THE PARALLEL APPLICATIONS ON THE EXAMPLE OF THE ATMOSPHERE MODEL NH3D

A.S. Antonov, RCC MSU (Moscow, Russian Federation),
A.M. Teplov, RCC MSU (Moscow, Russian Federation)

The article describes an approach to the scalability study using applications analyzing tools. This approach uses application profiling and tracing of the bottle necks for scalability analysis. There is a brief review of parallel program effectiveness marks, approaches to scalability study and performance analysis tools for parallel applications. This article performs detailed description of analyzing methodology and detailed review of parallel application that was analyzed using this methodology. Also in this article was performed the analysis results of described parallel application using profiling and tracing tools. For the analysis the authors used Intel Trace Analyzer and Collector as an example of tracing tool. In conclusion the authors admitted high level of analyzing features of this tool for scalability analysis purposes.

Keywords: scalability, effectiveness analysis, tracing, profiling, parallel application analysis tools.

References

1. Grama A., Gupta A., Karypis G., Kumar V. Introduction to Parallel Computing. (2nd Edition), Pearson, 2003.
2. Ivannikov V.P., Gaysaryan S.S., Padaryan V.A. Evaluation of Dynamic Characteristics of a Parallel Program on a Model. Programing and Computer Software. 2006. No. 4.
3. Alabdulkareem M., Lakshmivarahan S., Dhall S.K. Scalability analysis of large codes using factorial designs. Parallel Computing. 2001. Vol. 27. Issue 9. P. 1145–1171.
4. Muller-Wichards D., Ronsch W. Scalability of Algorithms: An Analytic Approach. Parallel Computing. 1995. Vol. 21. Issue 6. P. 937–952.
5. Stepanenko V.M., Mikushin D.N.. Numerical modeling of mesoscale atmospheric dynamics and pollutant transport over hydrologically diversified area // Computational technologies. 2008. Vol. 13, special issue 3. P. 104–110.

Поступила в редакцию 10 января 2013 г.