Некоторые аспекты эффективной реализации параллельных программ

[1]Вл.В.Воеводин, [2]П.А.Церетели
[1]Научно-исследовательский вычислительный центр МГУ
[2]Факультет прикладной математики и компьютерных наук ТГУ

Абстракт
Эффективное использование параллельных вычислительных систем на сегодняшний день является одной из основных проблем, т.к. пиковая производительность и реальная производительность, показанная на реальных программах, достаточно сильно отличаются друг от друга. Причин этому может быть несколько. Иногда это может быть «по вине» алгоритма и нет никакой возможномти улучшить результат, а иногда с помощью некоторых преобразований в программе можно улучшить производительность. Для достижения высокой эффективности структура алгоритма должна быть согласованна со структурой и особенностями используемой параллельной вычислительной системы. Большое разнообразие структур и архитектур современных суперкомпьютеров не позволяет разработать общий подход для создания эффективных параллельных программ. Поэтому актуальной является разработка методов и программных средств, которые помогут пользователю исследовать структуру и свойства программ. В работе обсуждаются возможности одной такой системы – V-Ray.

Введение

На современном этапе развития научно-технического прогресса решение проблемы эффективного использования суперкомпьютеров является одной из стратегических задач. Производительность супер-ЭВМ на протяжении последних 2-3 десятилетий возрастала в среднем на порядок в каждые 5 лет. Согласно последней редакции списка top500, содержащего 500 наиболее мощных суперкомпьютеров мира, на июнь 2000 года в мире было установлено более 10 компьютеров с пиковой производительностью более 1 TFLOPS (трлн. оп./сек.). В американской правительственной программе «Ускоренная стратегическая компьютерная инициатива» ASCI намечен выход на уровень производительности супер-ЭВМ 100 трлн. оп./сек. в 2004 г. [1] Но в том же top500 вместе с пиковой производительностью приводится и реальная производительность этих же вычислительных систем, показанная на тестовой программе LINPACK, и легко заметить разницу между этими двумя понятиями. Например, один из мощнейших суперкомпьютеров мира ASCI Blue Mountain (SGI, Inc.), установленный в Лос-Аламосской национальной лаборатории США, имеет 6144 процессоров и пиковую производительность более 3 TFLOPS, но лучший результат, показанный на тесте LINPACK, равен лишь 1,6 TFLOPS.

Причин, по которым теоретическая и реальная производительность столь сильно расходятся, достаточно много. Иногда это может быть «по вине» алгоритма и нет возможности улучшения результата, а иногда путем небольшого изменения в программе можно значительно уменьшить время реализации и, тем самым, увеличить производительность вычислительной системы на данной программе.

Что такое пиковая производительность? Допустим, у некоторой вычислительной системы есть устройство сложения и умножения. Ее пиковая производительность считается исходя из того, что оба устройства работают одновременно с полной нагрузкой. Однако, если в нашей задаче нет операций умножения (допустим, вычисляем сумму элементов массива), то больше половины пиковой производительности мы уже никак не получим. Поэтому для эффективного использования параллельных вычислительных систем проблема соответствия структуры алгоритма структуре и особенностям архитектуры используемого компьютера

выходит на первый план. Если программист хочет работать на пределе производительности компьютера, он должен учитывать его особенности.

Рассмотрим такой пример. Предположим, что на векторно-конвейерном суперкомпьютере CRAY Y-MP C90 надо реализовать операцию вида

$$A(i,j,k)=A(i,j,k-1)+b(j,i)+b(j,i), \quad i=1,..,1000,\ j,k=1,\ldots,40$$

Большинство программистов, не имеющих опыта программирования на подобных вычислительных системах, напишет программу в таком виде:

```
for (i=1; i<=1000; i++)
for (j=1; j<=40; j++)
for (k=1; k<=40; k++)
A[i,j,k]=A[i,j,k-1]+B[j,i]+B[j,i];
```

Реализуя этот фрагмент на CRAY Y-MP C90, мы получим производительность около 20 MFLOPS (млн. оп./сек.) при пиковой производительности 960 MFLOPS.

Если учесть особенности архитектуры CRAY Y-MP C90, эту задачу можно запрограммировать таким образом:

```
for (k=1; k<=40; k+=2)
for (j=1; j<=40; j++)
for (i=1; i<=1000; i++) {
A[i,j,k]=A[i,j,k-1]+2*B[j,i];
A[i,j,k+1]=A[i,j,k]+2*B[j,i];
}
```

Реализация этого фрагмента дает производительность 700 MFLOPS. При этом эти два фрагмента эквивалентны, так как дают одинаковый результат с точностью до возможных ошибок округления, возникших при замене сложения на умножение, но второй фрагмент намного лучше соответствует особенностям архитектуры целевого компьютера. Отсюда и разница в производительности. [1]

Таким образом, с одной стороны, параллелизм позволяет обеспечить большую скорость обработки данных, но, с другой стороны, процесс разработки эффективных программ становится все более и более сложным. Проблему усугубляет еще и тот факт, что существует огромное количество программ, которые разрабатывались для «обычных» однопроцессорных вычислительных машин. Они решают определенные базовые задачи в различных предметных областях и активно используются разработчиками прикладных программ. Естественно, они написаны без учета особенностей параллельных вычислительных систем. Программисту для достижения высокой эффективности приходится не только осмыслить структуру своей программы, но также изучить и переосмыслить структуру тех библиотечных модулей, которые он раньше использовал особо не задумываясь.

1. Современные параллельные вычислительные системы

Попробуем выделить некоторые основные классы современных параллельных вычислительных систем. Их классификацию можно осуществить по нескольким параметрам. Основным параметром классификации параллельных компьютеров является наличие общей или распределенной памяти. Между двумя этими способами организации памяти находится NUMA-архитектура, где память физически распределена, но логически общедоступна. Если у процессора есть возможность обработки векторов в конвейерном режиме, мы имеем дело с векторно-конвейерной архитектурой. Такие процессоры могут объединяться в системы с общей или распределенной памятью. Итак, выделим следующие классы:

1. Массивно-параллельные системы (MPP) состоят из однородных вычислительных узлов. Каждый узел содержит один или несколько процессорных устройств, локальную память, коммуникационный процессор или сетевой адаптер. Иногда узлы содержат еще жесткие диски и/или другие устройства ввода-вывода. Прямой доступ к памяти других узлов явным образом невозможен. К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду

(высокоскоростная сеть, коммутатор и т.п.). Такими системами являются: IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, Hitachi SR8000.

2. Симметричная мультипроцессорная система (SMP) состоит из нескольких однородных процессоров и массива общей памяти. Память, как правило, разделена на несколько блоков. Все процессоры могут обратиться к любому месту памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины, либо с помощью коммутатора. HP 9000 V-class, N-class; SMP-сервера и рабочие станции на базе процессоров различных фирм (Intel, IBM, Compaq и др.) являются представителями данного класса.

3. Система с неоднородным доступом к памяти (NUMA) состоит из однородных базовых модулей (плат). Модули имеют небольшое число процессоров и блок памяти. Они объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом, доступ к локальной памяти в несколько раз быстрее, чем к удаленной. На принципе NUMA-архитектуры построены SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000.

4. Основным признаком параллельных векторных систем (PVP) систем является наличие специальных векторно-конвейерных процессоров. Процессоры имеют конвейерные функциональные устройства для выполнения отдельных операций и предусмотрены команды обработки векторов. Как правило, несколько таких процессоров (1-32) могут работать одновременно над общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций, или они могут быть объединены с помощью коммутатора (аналогично MPP). Примеры: NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: CRAY-1, CRAY J90/T90,CRAY Y-MP C90, CRAY SV1, серия Fujitsu VPP.

5. Кластерные системы представляют собой набор рабочих станций (или даже персональных компьютеров) общего назначения. Они используются в качестве дешевого варианта массивно-параллельного компьютера. Для связи узлов применяется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet, SCI) на базе шинной архитектуры или коммутатора. Если кластер объединяет компьютеры разной архитектуры, в этом случае говорят о неоднородных кластерах. Узлы кластера могут одновременно использоваться в качестве пользовательских рабочих станций.

Такое разнообразие параллельных архитектур очень затрудняет разработку единого подхода к созданию эффективных программ. Более того, каждый представитель вышеприведенных классов имеет свои особенности, которые тоже следует учитывать. Поэтому очень сложно говорить об эффективной переносимости программного обеспечения. Программист должен хорошо знать не только структуру реализуемого алгоритма, но и архитектуру целевого компьютера, чтобы согласовать структуру алгоритма с особенностями используемой машины.

В свое время языки программирования освободили пользователей от необходимости знания структуры и особенностей конкретных ЭВМ. Они стали посредниками между пользователем и вычислительной техникой и взяли на себя функции адаптации программ к конкретным ЭВМ. Однако если компиляторы и операционные системы «обычных» однопроцессорных ЭВМ по большей части справлялись с этой задачей, то с появлением параллельных вычислительных систем ситуация стала намного хуже. Существуют языки параллельного программирования, параллельные компиляторы и операционные системы, но практически все они требуют, чтобы пользователь сам указывал те свойства алгоритмов и программ, которые влияют на эффективность реализации. Обнаружить такие свойства крайне сложно (например, выделить циклы, итерации которых можно выполнить параллельно). Поэтому на повестку дня встала разработка методов и инструментальных средств, которые помогут исследовать свойства алгоритмов и программ, обнаружить потенциал параллелизма и узкие места, которые мешают эффективной реализации на параллельной вычислительной системе, а потом приспособить программу к структуре целевого компьютера.

2.Факторы, влияющие на реальную производительность

Вернемся сейчас к нашему примеру и попробуем проанализировать, из-за чего второй фрагмент работает в 35 раз быстрее первого. Если внимательно посмотрим на 1-ый фрагмент, то обнаружим, что самый внутренний цикл по k не является параллельным, так как для вычисления A[i,j,k] требуется значение, вычисленное на предыдущей итерации A[i,j,k-1]. Так как CRAY-MP C90 является векторно-конвейерной вычислительной системой, в этой ситуации нельзя будет задействовать возможность векторной обработки и, следовательно, весь массив A будет вычисляться поэлементно.

Во втором фрагменте переставлены циклы, после чего итерации внутреннего цикла по i стали независимыми, т.е. для фиксированного j и k можно одновременно вычислить все элементы массива с первым индексом от 1 до 1000, а значит и будет возможность использовать потенциал векторной обработки. Более того, вычисления по k разбиты на k и k+1, что позволяет эффективно использовать векторные регистры вычислительной системы.

Какие же факторы влияют на эффективность программ? Вот некоторые из них. В первую очередь следует иметь в виду закон Амдала: если доля последовательных операций программы равна q $(0 \le q \le 1)$, то максимальное ускорение S, которое можно получить на p процессорах, не превосходит величины $\dfrac{1}{q+\dfrac{1-q}{p}}$.

Кроме закона Амдала на эффективность программы влияют конфликты при обращении к памяти, ограниченная пропускная способность каналов передачи данных, несбалансированность (т.е. неравномерная нагрузка) функциональных устройств и процессоров. Векторная обработка вносит дополнительные факторы; это - необходимость секционирования длинных векторов, ограниченный набор векторных регистров и многое другое. Согласно закона Амдала, если в программе всего лишь 2% последовательных операций, то на ускорение больше чем в 50 рассчитывать не приходится независимо от того, сколько процессоров используются. И эта теоретическая оценка не учитывает массу других нюансов, которые также снижают эффективность программы. Если мы используем многопроцессорную систему, которая содержит несколько сот процессоров, для высокой эффективности почти вся программа должна исполняться в параллельном режиме, что можно обеспечить только после детального анализа всей программы. При программировании для "обычных" последовательных компьютеров необходимости в таком анализе практически никогда не было.

3.Система анализа и визуализации структуры программ

Исследование программы и структуры алгоритма может быть произведено на основе статического и динамического анализа. Статический анализ предполагает обработку только исходного кода, т.е. программы, записанной на каком-либо языке программирования, тогда как динамический анализ связан с выполнением программы на целевом или каком-нибудь другом компьютере.

Какие же средства могут понадобиться пользователю для анализа и исследования структуры программы. Рассмотрим некоторые такие средства на примере системы визуализации и исследования структуры программы V-Ray. [3,4]

При исследовании больших прикладных программных пакетов пользователь не всегда имеет достаточно полное понимание того, как программа устроена, т.е. какие процедуры используются и как они вызываются. Понимание макроструктуры программы может помочь ему определить основные процедуры, на оптимизацию которых нужно обратить внимание. Незнание макроструктуры может быть связано с тем, что разные части программы могут быть написаны разными людьми или просто программа может состоять из нескольких сотен процедур. В этих случаях очень трудно без специальных средств осознать макроструктуру программы. На рис.1 приведено изображение графа вызовов программы, полученное с помощью системы V-Ray. Каждой вершине этого графа соответствует некоторая процедура

программы (подпрограмма или функция в терминах Фортрана). Из вершины V_1 в вершину V_2 идет дуга в том случае, если из V_1 вызывается процедура V_2.

На такой граф вызовов можно наложить и некоторую другую информацию. Например, раскрасить вершины графа разными цветами, соответствующими максимальной глубине вложенных циклов в данной процедуре. Такая информация может быть полезна для пользователя, так как, как правило, в циклах и кроется весь потенциал параллелизма. Можно граф вызовов показать и с изображением циклического профиля каждой процедуры. На рис.2 приведено именно такое изображение графа. Здесь в каждой вершине изображена структура циклов и, в отличие от графа, приведенного на рис.1, если процедура вызывается из n разных мест, соответствующая вершина дублируется n раз. Такая форма представления в некоторых случаях может быть более удобна.

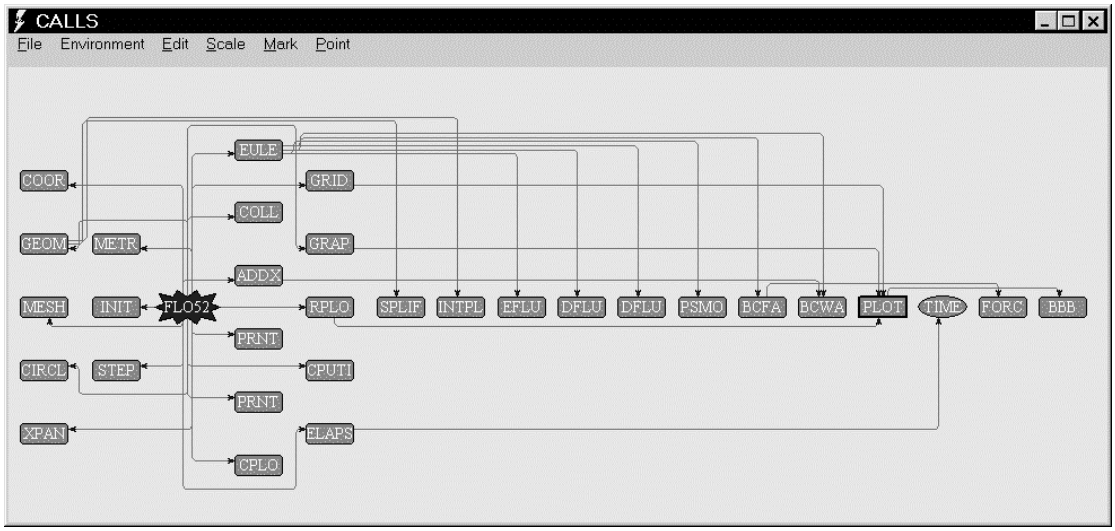

Рис. 1

Граф вызовов строится с помощью статического анализа. Динамический анализ позволяет собрать дополнительную полезную информацию, которую можно будет наложить на нем. Когда пользователь приступает к адаптации программы, естественно, исследование лучше начать с тех частей программы, которые вносят наибольший вклад в общее время реализации. Поэтому пользователю неплохо было бы знать какие фрагменты программы за какое время выполняется. Это позволит ему выделить те фрагменты (процедуры, циклы или другие части программы), которым следует уделить особое внимание при оптимизации. Но для того чтобы получить динамические характеристики программы, ее нужно специальным образом подготовить – вставить в нужных местах вызовы служебных функций, которые будут замерять время в контрольных точках. Система V-Ray позволяет в автоматическом режиме вставить такие вызовы в программу. В начале и в конце фрагмента ставятся вызовы функции времени и вычисляются такие показатели, как общее время выполнения фрагмента, число ее вызовов, минимальное и максимальное время работы. Такой способ сбора динамической информации называется профилировкой. После реализации на компьютере модифицированной программы, собранную в нужном формате информацию можно наложить на граф вызовов. На рис.3 приведен именно такой граф. Из него видно, что выполнение подпрограммы STEP, которая вызывается в цикле из подпрограммы MHD2 занимает более 70% общего времени и, естественно, особое внимание нужно уделить оптимизации и эффективной реализации именно этой части программы.

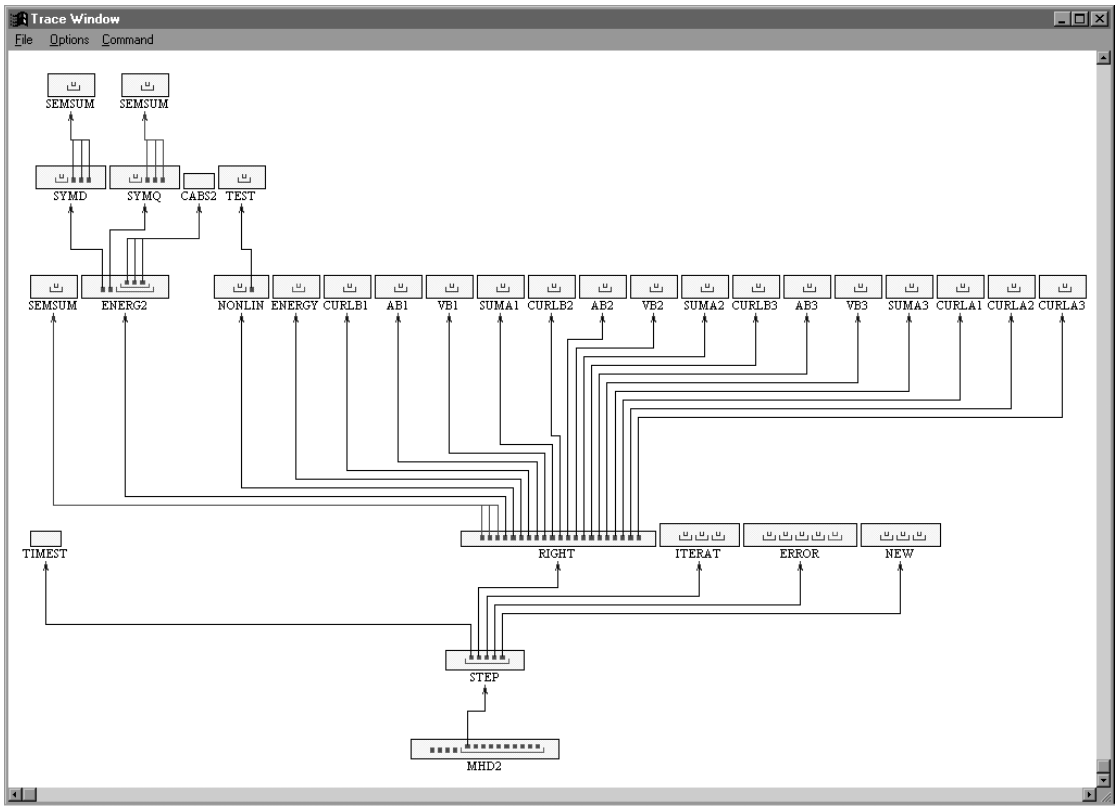

Рис. 2

Знание динамических характеристик программы позволяет также изучить поведение параллельных программ. В этом случае используется другой механизм сбора динамической информации, который состоит в том, что фиксируются моменты прохождения контрольных точек (событий) и называется трассировкой. Это позволяет определить на каком процессоре какое действие выполнялось в данный момент времени и быстро выявлять основные источники накладных расходов, такие, как ожидания на барьерах, ожидания при приеме сообщений, дисбаланс вычислительной нагрузки между процессами и т.д.

Особым случаем на практике является трассировка процедур, обеспечивающих взаимодействие между параллельными процессами, например, трассировка функций MPI. На рис.4 и 5 приведены фрагменты визуализации трасс.

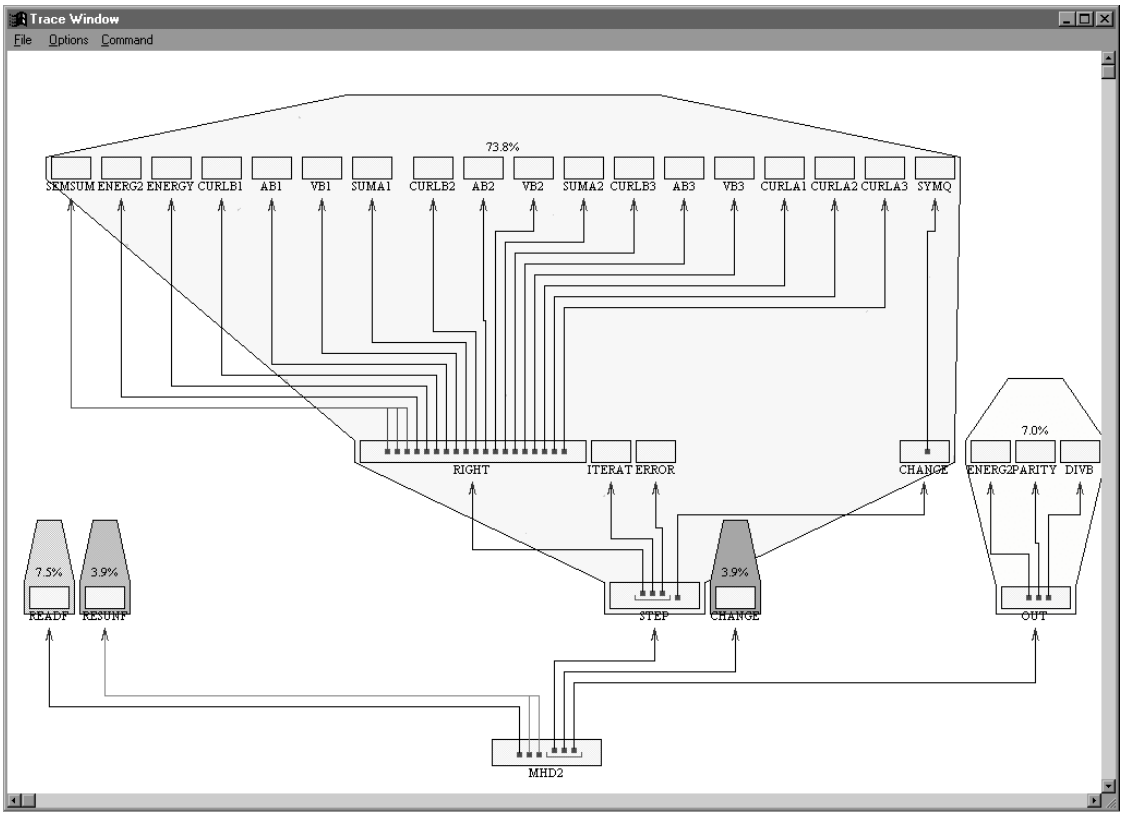
Trace Window
File Options Command
73.8%
SEMSUM ENERG2 ENERGY CURLB1 AB1 VB1 SUMA1 CURLB2 AB2 VB2 SUMA2 CURLB3 AB3 VB3 CURLA1 CURLA2 CURLA3 SYMQ
RIGHT
ITERAT ERROR
CHANGE
7.0%
ENERG2 PARITY DIVB
7.5%
3.9%
READF RESUME
STEP
3.9%
CHANGE
OUT
MHD2

Рис. 3

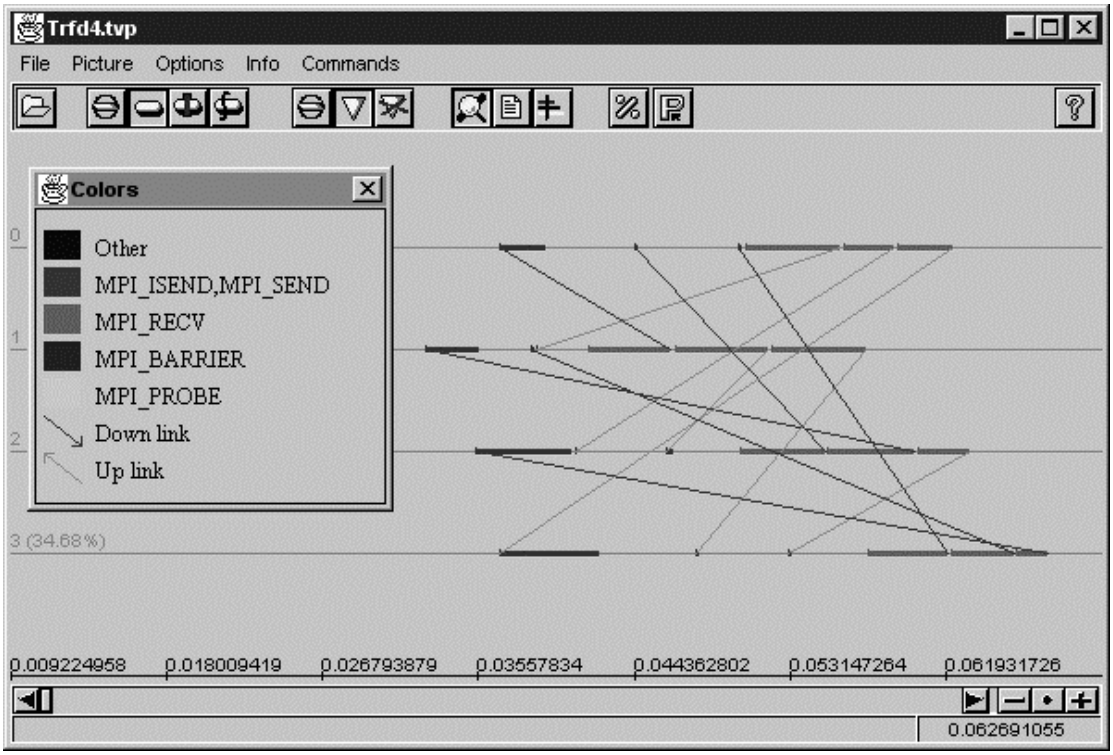
Trfd4.tvp
File Picture Options Info Commands
Colors
Other
MPI_ISEND,MPI_SEND
MPI_RECV
MPI_BARRIER
MPI_PROBE
Down link
Up link
0
1
2
3 (34.68%)
0.009224958
0.018009419
0.026793879
0.03557834
0.044362802
0.053147264
0.061931726
0.062691055

Рис. 4

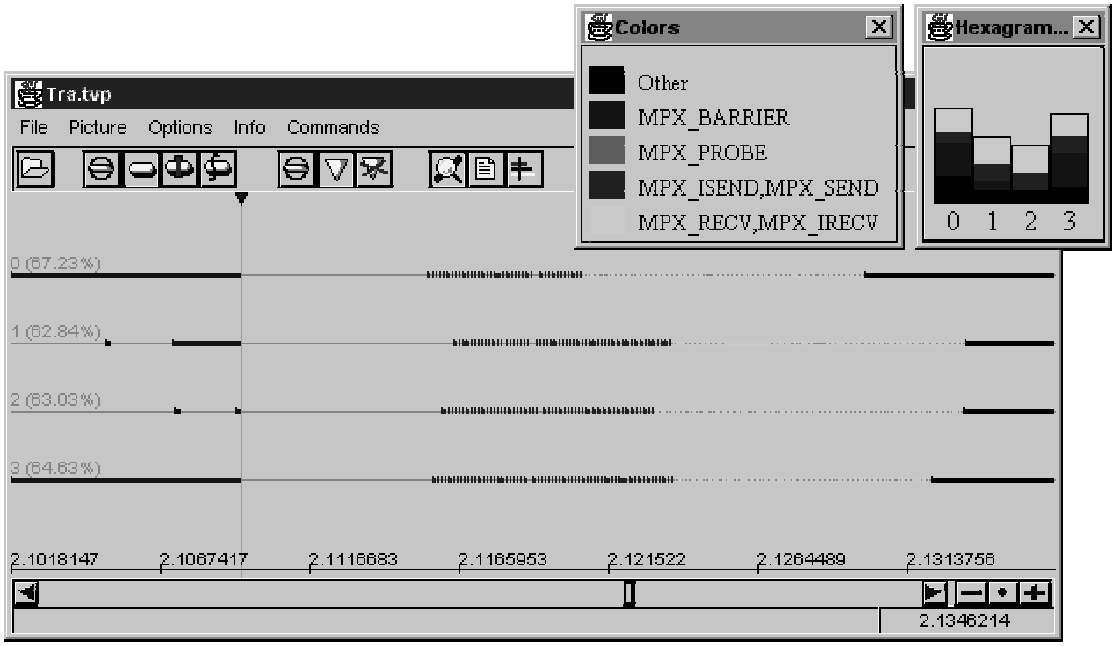

Рис. 5

Из этих изображений пользователю становится ясно, что 0-ой и 3-ий процесс (рис.5) достаточно долго простаивали на барьере, а на рис.4 можно увидеть структуру передачи сообщений между процессами. Развитые средства трассировки и визуализации позволяют получить достаточно подробную информацию о событиях - всегда можно определить какое событие какому оператору соответствует, по какому пути этот оператор вызывался, каков объем передаваемой информации и т.д.

Имея такие средства, пользователь получает возможность исследования тонких свойств программ. Например, объединив трассы, собранные во время нескольких запусков одной и той же программы, в одну метатрассу, можно изучить зависимость поведения параллельной программы от различных наборов входных данных. В частности, сложная структура программ, зачастую и недетерминированность поведения параллельных программ могут привести к миграции «узких» мест.[2]

Понимание структуры программы, обнаружение «узких» мест или определение наиболее «весомых», с точки зрения времени выполнения, подпрограмм (процедур) и фрагментов – это всего лишь первый шаг к разработке высокоэффективных программ. Дальше следует исследовать структуру отдельных фрагментов и приспособить их к архитектуре целевого компьютера. Система V-Ray предоставляет и такую возможность – возможность исследования структуры отдельных процедур. С помощью статического анализа мы можем определить такие характеристики, как циклическая структура процедуры, независимость итераций циклов, информационная зависимость отдельных операторов и линейных участков и многое другое.

На рис.6 и 7 приведены графы управления фрагмента и циклический профиль процедуры. Вершины с метками соответствуют циклическим конструкциям (вложенные циклы), метка означает максимальный уровень вложенности циклов. Остальным вершинам соответствуют отдельные операторы и линейные участки. Внизу изображена структура циклов. На рис.7 вершина более темного цвета с меткой 2 соответствует циклу, итерации которого независимы и, соответственно, его можно выполнить в параллельном режиме. Для исследования структуры алгоритмов на микроуровне, т.е. на уровне отдельных операторов используются графы зависимостей [5].

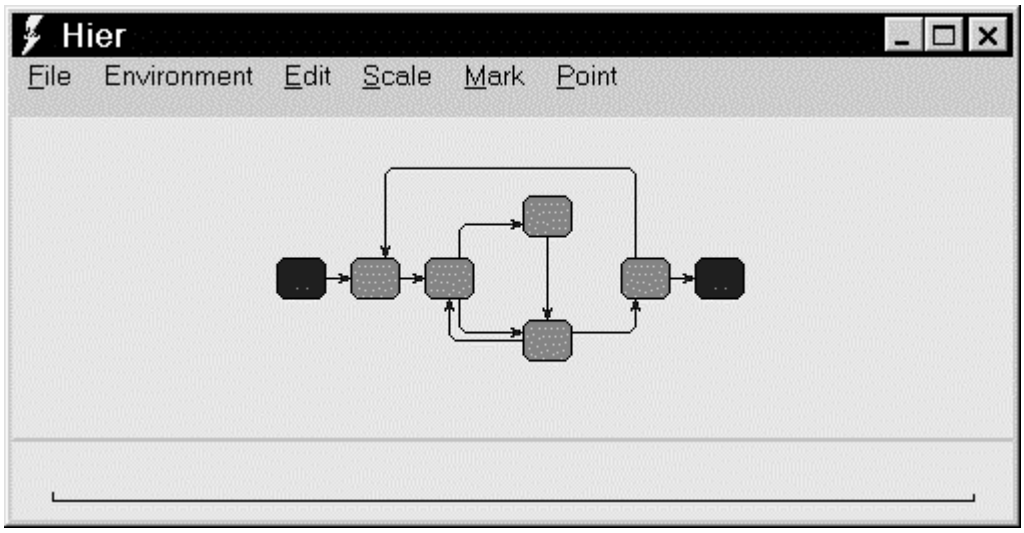

Рис. 6

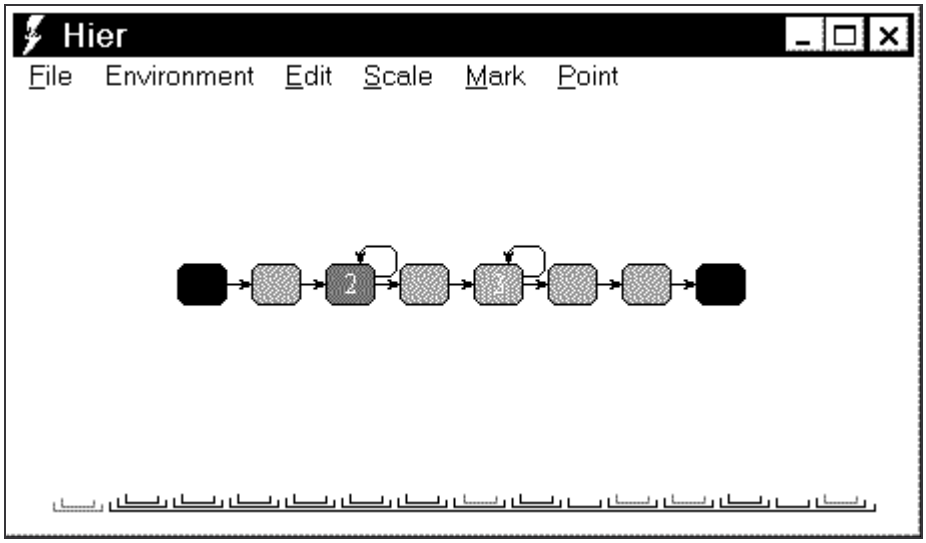

Рис. 7

В данной статье мы не можем и не имели целью описать всю функциональность системы V-Ray и коснулись лишь малой части ее возможностей. Основная мысль, которую нам хотелось довести до читателя заключается в том, что несмотря на всю сложность и многогранность задачи согласования структуры программ с особенностями архитектуры параллельных вычислительных систем, она может быть решена и, более того, она успешно решается на практике. Решение далеко не всегда тривиально и может не лежать “на поверхности”, но совместные усилия коллективов программистов и прикладных специалистов всегда приведут к желаемому результату.

Более подробную информацию о системе V-Ray, а также многочисленную информацию по проблемам параллельных вычислений можно найти в Интернете на сайте Информационно-аналитического центра по параллельным вычислениям http://www.parallel.ru.

Л И Т Е Р А Т У Р А

1. *Суперкомпьютерные вычислительные и информационные технологии в физических и химических исследованиях*. Сборник лекций Первой Всероссийской молодежной школы (1-2 ноября 1999, Черноголовка).
2. Вл. В. Воеводин, М.П. Филамофитский, П.А. Церетели. *Комплексный подход к анализу динамических характеристик параллельный программ*. Фундаментальные и прикладные аспекты разработки больших распределенных программных комплексов: Тезисы докладов Всероссийской научной конференции (21-26 сентября 1998 г., Новороссийск). –М.: Изд-во МГУ, 1998, с.123-126.
3. Vl.V. Voevodin. *Theory and Practice of Parallelism Detection in Sequential Programs*. Programming and computer software, 1992.
4. V.V. Voevodin, Vl.V. Voevodin. *V-Ray Technology: a New Approach to the Old Problems.* Optimization of the TRFD Perfect Club Benchmark to CRAY Y-MP and CRAY T3D Supercomputers. Proc. of the High Performance Computing Symposium'95, Phoenix, Arizona, USA, 1995.- pp.380-385.
5. В.В. Воеводин. *Информационная структура алгоритмов*. М.: Изд-во МГУ, 1997.