Методы точного измерения времени выполнения гнезд циклов при анализе *JavaMPI*-программ в среде *ParJava*

А.И. Аветисян, М.С. Акопян, С.С. Гайсарян [1],
arut@ispras.ru, manuk@ispras.ru, ssg@ispras.ru

Аннотация. В работе рассматриваются методы оценки времени выполнения модели параллельной программы на инструментальном компьютере, которые позволяют достаточно точного предсказывать время реального выполнения параллельной программы на заданном параллельном вычислительном комплексе. Модель разработана для параллельных SPMD программ с явным обменом сообщениями, написанных на языке Java с обращениями к библиотеке MPI, и включена в состав среды ParJava. В модели выделяются определенные виды циклов (однородные, редуцируемые) и производится их оценка на узле целевой вычислительной системы (высокопроизводительного кластера). Это позволяет не только уменьшить погрешность предсказания, но и ускорить время интерпретации модели на инструментальном компьютере.

Ключевые слова: параллельные вычисления; моделирование параллельной по данным программы; оценка времени выполнения; оценка масштабируемости; многоядерность.

1. Введение

Среда *ParJava* [1] предоставляет прикладному программисту набор инструментов, поддерживающих разработку прикладных параллельных программ для вычислительных систем с распределенной памятью (высокопроизводительных кластеров) на языке *Java*, расширенном стандартной библиотекой передачи сообщений *MPI* [2, 3]. Многие инструменты среды *ParJava* могут выполняться на инструментальном компьютере, используя вместо разрабатываемой параллельной программы ее модель [4], которая может интерпретироваться как на целевой вычислительной системе, так и на инструментальном компьютере. Модель параллельной программы адекватно отражает ее поведение на кластере,

1 НИР поддержана грантами РФФИ 09-07-00382-а и ФЦП «Научные и научно-педагогические кадры инновационной России на 2009-2013 годы», ГК №П728 от 12 августа 2009 года

позволяя исследовать динамические характеристики (в частности, время выполнения) разрабатываемой программы, или ее частей.

Модель *SPMD*-программы представляет собой совокупность моделей всех классов, входящих в состав моделируемой программы. Модель каждого класса – это множество моделей всех методов этого класса; кроме того, в модель класса включается модель еще одного дополнительного метода, описывающего поля класса, его статические переменные, а также инициализацию полей и статических переменных. Модель метода (функции) представляется в виде пары ⟨модель потока управления, модель вычислений⟩. Модель потока управления представляет собой модифицированное дерево управления метода, полученное из графа потока управления путем выделения и сворачивания областей. Внутренние вершины модели соответствуют операторам управления моделируемой программы, а листовые представляют собой дескрипторы ее базовых блоков. В качестве базовых блоков рассматриваются не только линейные последовательности вычислений (вычислительные базовые блоки), но также и вызовы библиотечных функций, вызовы пользовательских методов и функций и вызовы коммуникационных функций. Для обеспечения возможности частичной интерпретации модели и интерпретации модели по частям введено понятие редуцированного блока, который представляет значения динамических атрибутов уже проинтерпретированных частей метода. Дескриптор каждого базового блока содержит оценку времени выполнения данного базового блока, определенную до начала интерпретации на одном узле целевой вычислительной системы. Модель вычислений содержит байт-код каждого базового блока моделируемого метода, что позволяет выполнять (интерпретировать) ее на *JavaVM*. По окончании выполнения очередного базового блока модели вычислений вызывается интерпретатор среды *ParJava*, который, с помощью модели потока управления, определяет очередной базовый блок, который следует интерпретировать.

В данной работе рассматриваются проблемы, связанные с прогнозом времени выполнения параллельной прикладной программы и исследованием зависимости времени выполнения программы от числа узлов целевой вычислительной системы (кластера), участвующих в выполнении программы. Оценки времени выполнения программы для различных значений числа улов кластера, позволяют прикладному программисту построить график зависимости оценки времени выполнения его программы от числа задействованных узлов, оценить границы области масштабируемости программы и выбрать оптимальное число узлов кластера. Кроме того, такие оценки позволяют вручную сбалансировать обмены данными между параллельными ветвями программы за счет правильного выбора функций библиотеки *MPI* и определения оптимальных точек вызова этих функций.

Интерпретатор модели параллельной программы, работающий на инструментальном компьютере использует временные профили параллельной программы, полученные заранее на целевой вычислительной системе.

Проблемы, связанные с получением достаточно точных оценок времени выполнения операций обмена по характеристикам коммуникационной сети кластера рассмотрены в работе [5]. В данной работе рассматриваются методы оценки времени выполнения фрагментов *Java*-программы (в частности, базовых блоков), не содержащих операций обмена. Как показано в данной работе, при построении временного профиля не следует ограничиться оценкой времени выполнения базовых блоков, интерпретация может быть существенно ускорена, если оценивать и время выполнения более крупных фрагментов: отдельных итераций циклов, циклов в целом, а в некоторых случаях – и гнезд циклов.

Временной профиль моделируемой программы строится на целевой вычислительной системе и уточняется в процессе интерпретации программы. С помощью операции редукции (свертки), введенной в [4], модель преобразуется к форме, обеспечивающей ее быструю интерпретацию. Методам разбиения модели программы на достаточно крупные фрагменты, для которых возможно получить и использовать достаточно точные оценки их времени выполнения, посвящена данная статья.

В разделе 2 приводится краткое описание модели программы, методов оценки времени выполнения программы и ее фрагментов, реализованных в среде ParJava, и уточняется постановка задачи. В разделе 3 рассматривается схема укрупнения фрагментов программы, для которых возможно построить достаточно точные оценки времени их выполнения. В частности, в данном разделе рассматриваются методы получения оценок для гнезд циклов и разбираются случаи, когда этих оценок достаточно для моделирования поведения программы с требуемой точностью. В разделе 4 обсуждаются методы измерения степени разбалансированности гнезда циклов и методы оценки времени выполнения разбалансированных гнезд циклов в среде ParJava. В разделе 5 рассматриваются методы получения оценок времени выполнения параллельных циклов, требующих синхронизации. В разделе 6 рассмотренные методы применяются для оценки ускорения и границ области масштабируемости некоторых модельных и реальных программ. В частности строятся графики предсказания границ области масштабируемости для программы моделирования интенсивных атмосферных вихрей [6].

2. Вычисление оценки времени выполнения методов.

Как описано в [4], модель каждого метода *Java*-программы рекурсивно строится из моделей его фрагментов[2] – базовых блоков и их комбинаций (последовательностей, ветвлений, переключателей и циклов). Измерив значение времени выполнения каждого базового блока, можно в процессе интерпретации модели вычислять оценки времени выполнения различных комбинаций базовых блоков и более крупных фрагментов, получая в конце интерпретации метода оценку времени его выполнения. При этом время выполнения последовательности фрагментов всегда равно сумме времен выполнения составляющих фрагментов. Но если в состав фрагмента входит ветвление, то время его выполнения зависит от времени выполнения текущей ветви, т.е. определяется значениями параметров метода. Если время выполнения фрагмента не зависит от параметров метода (например, фрагмент не содержит ветвлений, или время выполнения каждой ветви примерно одинаково[3]), будем называть такой фрагмент *простым*. В терминах [4] каждый простой фрагмент может быть редуцирован.

Легко видеть, что если тело цикла является простым фрагментом, то и сам цикл является простым фрагментом и может быть редуцирован. Мы будем называть такие циклы *простыми*. Подобные соображения применимы и к соответствующим гнездам циклов, которые тоже будут называться *простыми*. Если можно доказать, что гнездо циклов является простым, можно при определении временного профиля метода измерить время выполнения такого гнезда и заменить его на редуцированный блок с соответствующим временем выполнения.

Следует отметить, что гнездо циклов, содержащее операции обмена данными через *MPI*, не может быть простым, хотя внутренние циклы, не содержащие операций обмена, могут быть простыми и допускать редукцию.

Для параллельных *MPI*-программ, написанных на одном из традиционных языков (C/C++, Fortran), модель должна строиться не над исходным кодом программы, а над ее промежуточным представлением с учетом результатов статической оптимизации, так как оптимизаторы современных компиляторов могут существенно изменить структуру программы, а следовательно, и ее модель.

Для параллельных *Java*-программ, использующих *MPI*, структура программы, выполняемой на *Java VM*, практически не отличается от структуры исходной

[2] В данной работе рассматриваются фрагменты, которые в литературе обычно называются областями или регионами, т.е. подграфы графа потока управления метода, имеющие единственный входной базовый блок, доминирующий над всеми остальными блоками фрагмента.

[3] Времена выполнения ветвей различаются на p%, где p – достаточно мало (например, $p = 10$).

программы: статическая оптимизация, как правило, производится только внутри базовых блоков и не влияет на структуру программы. В процессе выполнения *Java*-программы на *Java VM* определяются наиболее часто выполняемые фрагменты программы (*«горячие» методы*) и для них выполняется динамическая компиляция с агрессивной оптимизацией (разработчики динамических компиляторов считают, что такие фрагменты составляют менее 20% всей программы [7]). Таким образом, большая часть методов программы продолжает интерпретироваться на *Java VM*, причем такие методы обычно удовлетворяют условиям редукции и при интерпретации модели вносят некоторый постоянный вклад во время выполнения соответствующих фрагментов (время интерпретации таких методов в среде *ParJava* сравнимо с временем интерпретации одного базового блока).

Что касается «горячих» методов, то для ускорения интерпретации их моделей с сохранением точности оценок времени выполнения необходимо уметь выделять как можно более крупные фрагменты, в пределах которых бывает сосредоточена большая часть структурных изменений кода, связанных с оптимизацией, и при профилировании измерять время выполнения таких фрагментов в целом, по возможности игнорируя их структуру. В параллельных программах такими крупными фрагментами являются гнезда циклов.

3. Измерение времени выполнения сбалансированных гнезд циклов.

Рассмотрим гнездо циклов **for**. Оно может быть представлено в виде дерева, корнем которого является самый внешний цикл **for**, а поддеревьями являются циклы **for** глубины вложенности 1, далее по рекурсии. Если у одного из циклов гнезда нет вложенных в него циклов, то соответствующий ему узел дерева не имеет поддеревьев и является терминальным в дереве, представляющем гнездо циклов.

В *MPI*-программах, как правило, распараллеливается один из внешних циклов гнезда (часто это бывает самый внешний цикл). В дальнейшем будут рассматриваться только распараллеливаемые подгнезда, у которых распараллеливается самый внешний цикл.

Рассмотрим цикл **for**, количество повторений которого равно n. Интерпретация этого цикла на интерпретаторе среды *ParJava* (либо на *JavaVM*) позволит измерить:

(1) время выполнения цикла T_{Loop} ;

(2) время выполнения тела цикла на i-ой итерации $T^{\mathrm{i}}_{\mathrm{LoopBody}}$.

Сравнение $T^{\mathrm{i}}_{\mathrm{LoopBody}}$ со средним временем выполнения тела цикла $T^{\mathrm{mean}}_{\mathrm{LoopBody}} = \frac{T_{\mathrm{Loop}}}{\mathrm{n}}$ позволит определить степень сбалансированности цикла $D = \max_i \left| T^{\mathrm{i}}_{\mathrm{LoopBody}} - T^{mean}_{\mathrm{LoopBody}} \right|$. Будем говорить, что рассматриваемый цикл *сбалансирован*, если его степень сбалансированности D не превышает $\alpha \cdot T^{\mathrm{mean}}_{\mathrm{LoopBody}}$. (поскольку допустимая погрешность оценки времени выполнения фрагментов программы порядка 10% [1], можно принять $\alpha = 0{,}1$). В противном случае цикл будет считаться *разбалансированным*.

Поскольку оптимизирующие преобразования цикла (такие, как вынесение из цикла вычислений, инвариантных относительно цикла, или минимизация числа индуктивных переменных) не влияют на его степень сбалансированности D, измерение D можно выполнять для исходного кода программы при его интерпретации на интерпретаторе среды *ParJava* , либо на *JavaVM*.

Гнездо циклов будем называть сбалансированным, если каждый цикл, входящий в его состав, сбалансирован. Анализ сбалансированного гнезда циклов выполняется, начиная с самых внутренних («листовых») циклов. Для каждого такого цикла измеряется время его выполнения, после чего указанный цикл редуцируется. Поэтому на каждом этапе анализа рассматривается цикл, у которого нет вложенных циклов (после редукции каждый вложенный цикл заменяется «эквивалентным» редуцированным базовым блоком). Если сбалансированы как сам цикл, так и все подциклы, входящие в его тело, то при снятии временного профиля получается оценка времени выполнения всего цикла, а оценка времени выполнения отдельной итерации цикла определяется делением оценки времени выполнения цикла на число его итераций.

Рассмотрим случай сбалансированного цикла, число итераций которого определяется во время выполнения программы (например, в объемлющем цикле). В этом случае нужно получить оценку времени выполнения цикла для достаточно большого числа итераций, потом получить оценку времени выполнения одной итерации цикла, причем последняя будет использоваться для вычисления оценки времени выполнения рассматриваемого цикла на каждой итерации объемлющего цикла. Аналогичный метод применим и к вычислению оценок времени выполнения циклов, распределяемых среди узлов вычислительной системы для параллельного выполнения. Результаты численных расчетов подтвердили правомерность описанного подхода к оценке времени выполнения сбалансированных циклов.

4. Измерение степени разбалансированности гнезда циклов и времени выполнения разбалансированных гнезд циклов в среде ParJava.

Рассмотрим цикл вида:

```
for(i = 0; i ≤ n; i++) {
      {fr1};
      if(c(i)) {fr2};
      else {fr3};
      {fr4}
}
```

Интерпретация фрагментов **{fr1}**, **{fr2}**, **{fr3}** и **{fr4}** и условия **c(i)** на интерпретаторе среды *ParJava* (либо на *JavaVM*) позволяет получить предварительные оценки времени их выполнения: T_{fr1}, T_{fr2}, T_{fr3}, T_{fr4}, $T_{c(i)}$. При этом оценка времени выполнения тела цикла может быть вычислена по формуле $T_{LoopBody} = T_{fr1} + T_{c(i)} + T_{Cond} + T_{fr4}$, где T_{Cond} определяется следующим образом: на интерпретаторе оцениваются частоты выполнения ветвей условного оператора F_{fr2} и F_{fr3}, после чего T_{Cond} определяется по формуле

$$T_{Cond} = \frac{F_{fr2} \cdot T_{fr2} + F_{fr3} \cdot T_{fr3}}{F_{fr2} + F_{fr3}}$$

Если фрагменты **{fr1}**, **{fr2}**, **{fr3}** и **{fr4}** содержат ветвления, оценки времена их выполнения должны быть получены заранее, с помощью описанного метода.

Для определения степени разбалансированности цикла (или гнезда циклов) интерпретатор модели запускается не менее чем в P потоках (P – достаточно большое число, например $P \geq 16$), и время интерпретации гнезда циклов измеряется в каждом потоке. Пусть T^p_{Cond} – оценка T_{Cond} в p-ом потоке ($1 \leq p \leq P$). Тогда степень разбалансированности вычисляется по формуле $D = \max_p(T^p_{Cond}) - \min_p(T^p_{Cond})$. Легко видеть, что рассмотренный метод применим для оценки разбалансированности циклов общего вида.

Ввиду того, что при вычислении времени выполнения цикла и его отдельных итераций измерение времени производится лишь в начале и конце итерации, оценка получается правильной, несмотря на то, что структура выполняемого цикла после динамической компиляции может быть оптимизирована, в результате чего она может существенно отличаться от структуры исходного цикла. Может измениться даже число итераций цикла из-за его частичной раскрутки. Интерактивный сценарий устранения обнаруженной разбалансированности гнезда циклов состоит в распределении по потокам

групп итераций цикла таким образом, чтобы времена выполнения указанных групп итераций стали близки (различались на достаточно малую величину). В случае недостаточно низкой (с точки зрения пользователя) степени разбалансированности (это выясняется с помощью интерпретатора) пользователь вручную меняет распределение и снова с помощью интерпретатора выясняет степень разбалансированности, после чего либо процесс заканчивается, либо проверяется новое распределение. Если цикл разбалансирован, оценка времени его выполнения получается как сумма оценок времен выполнения всех его итераций. При этом время измеряется только в начале и в конце итерации: интерпретации итераций не требуется, так что оптимизация не влияет на оценку.

5. Оценка времени выполнения циклов, требующих синхронизации.

В предыдущих разделах рассматривались циклы, целиком выполняемые на одном из узлов кластера, т.е. циклы, которые во время своего выполнения не требуют периодических обменов данными между узлами кластера (и связанной с такими обменами синхронизации вычислений на разных узлах). В данном разделе рассматриваются оценки, необходимые для правильного (оптимального) размещения вызовов функций библиотеки *MPI* в теле цикла, выполняемого параллельно на нескольких узлах кластера. Указанные вызовы должны быть размещены таким образом, чтобы (1) передача данных велась параллельно с расчетами, и (2) передача данных заканчивалась прежде, чем потребуется начать их обработку [8]. Таким образом, оптимизация (ее обычно называют настройкой параллельного цикла) состоит в нахождении такого взаимного расположения фрагментов тела цикла, при котором выполняются условия (1) и (2), и для ее решения необходимо знать оценки времени выполнения каждого из рассматриваемых фрагментов тела цикла. Пример такой оптимизации рассмотрен в [4].

Следует отметить, что методы, содержащие параллельные циклы, входят в число наиболее часто выполняемых методов программы и, следовательно, оптимизируются во время динамической компиляции. Значит для оптимизации (настройки) такого метода необходимо уточнить его модель, отразив в ней оптимизирующие преобразования, выполненные при его динамической компиляции. Основная трудность в том, что не все динамические компиляторы языка *Java* (в частности, [9]) предоставляют доступ к коду оптимизированной программы. Тем не менее, один из наиболее современных динамических компиляторов [7], разработанный в IBM, обеспечивает такой доступ, а тем самым и принципиальную возможность уточнения модели рассматриваемого метода после его динамической компиляции. Анализируя оптимизированный код можно построить его граф потока управления, а затем и дерево управления, лежащее в основе модели параллельной *Java*-программы среды *ParJava*. Используя эту возможность, в

среде *ParJava* строится уточненная модель тела параллельного цикла, позволяющая оценить время выполнения каждого его фрагмента и тем самым выбрать оптимальные точки расположения вызовов функций обмена данными. Такой подход является достаточно трудоемким, но он позволяет достаточно точно оценить задержки, вызванные асинхронностью выполнения частей циклов на узлах кластера, и добиться их минимизации.

Для сбалансированного цикла, различные итерации которого требуют примерно одинаковых объемов вычислений, планы вычислений на различных узлах кластера одинаковы. Время t, требуемое на обмен данными между такими узлами, складывается из времени t_{dt}, необходимого на передачу требуемого объема данных через канал связи, и *времени синхронизации* t_s, которое идет на компенсацию асинхронности работы узлов кластера (и ядер указанных узлов). Зная параметры коммуникационной сети кластера можно достаточно точно определить t_{dt}. Для времени t_s возможны лишь вероятностные оценки, так как в это время входит ожидание в случае, когда к моменту обращения к данным, оказывается, что обмен еще не завершен. Исследование детального профиля сбалансированного цикла позволяет настроить цикл, обеспечив приемлемые значения ускорения и масштабируемости.

В случае разбалансированного цикла, время вычисления различных итераций которого может отличаться на большие значения, настройка программы существенно усложняется. Но и в этом случае детальный профиль тела цикла может помочь найти методом проб приемлемое распределение данных по узлам кластера.

6. Результаты численных расчетов.

Реализация рассмотренных методов оценки времени выполнения *Java*-программ и их фрагментов позволила применять среду *ParJava* при разработке достаточно сложных параллельных программ. В данном разделе приведены оценки времени выполнения для двух программ-примеров, использовавшихся при реализации среды *ParJava*. Эти примеры позволили выявить некоторые особенности кластеров с многоядерными узлами. Чтобы показать применимость среды *ParJava* при разработке реальных прикладных программ приведены результаты ее применения при разработке прикладной программы численного решения системы уравнений, моделирующей процессы и условия генерации интенсивных атмосферных вихрей (ИАВ) в трехмерной сжимаемой атмосфере, исходя из теории мезомасштабных вихрей В. Н. Николаевского [10]. На приведенных графиках «Действительное ускорение» представляет собой ускорение программы, измеренное при выполнении программы на целевой платформе, «Прогнозированное ускорение» - ускорение, предсказанное интерпретатором ParJava на инструментальной машине, «Разность ускорений» - модуль разности между предсказанным и действительным ускорением.

Пример 1. Программа умножения матриц. На рисунке 1 представлены графики зависимости ускорения от числа процессов для программы умножения квадратных плотных матриц A и B порядка n. Программа выполнялась на кластере ИСП РАН, содержащем 12 узлов. Каждый узел этого кластера состоит из двух четырехъядерных процессоров Intel® Xeon® 5355, так что всего на узле 8 ядер. Каждое ядро имеет собственный кэш *L1* размером 32Kb и кэш *L2* размером 4Mb, разделяемый с соседним ядром. Каждый процесс обрабатывал n/P строк матрицы A (P – число процессов) и всю матрицу B, вычисляя соответствующие n/P строк матрицы-произведения C. На графиках представлены *действительное ускорение*, полученное при выполнении программы, и *прогнозированное ускорение*, полученное при интерпретации программы в среде *ParJava*.

Сначала каждом узле кластера было запускалось 8 процессов (по 1 процессу на каждое ядро). Оказалось, что в этом случае ускорение (нижняя пара кривых на рисунке 1) примерно в два раза меньше максимального, определяемого по формуле Амдаля (пунктирная кривая на рисунке 1). При исследовании этого вопроса выяснилось, что неудовлетворительное ускорение связано с особенностями архитектуры используемых узлов: ресурсы узлов делятся между их ядрами, и при полной загрузке борьба ядер за ресурсы приводит к разбалансированию вычислений и падению ускорения. И действительно, расчеты, в которых на каждом узле кластера запускалось по 4 процесса (половина максимально возможного количества), показали ускорение, достаточно близкое к максимальному (верхняя пара кривых на рисунке 1). Отметим, что именно такое ускорение ожидалось для такой простой программы. Оказалось, что для рассматриваемого примера прогнозиреванное ускорение в обоих случаях было настолько близко к реально полученному, что соответствующие пары кривых на графике почти слились. Поэтому на нижнем графике рисунка 1 приведены графики модуля разности между прогнозированным и реальным значениями (абсолютная погрешность прогноза). Высокая точность прогноза обусловлена тем, что исследуемое гнездо циклов является хорошо сбалансированным, а обмен данными между процессами во время выполнения программы не производится. Последовательная часть программы составляет всего 0,6%.

В рассматриваемом примере порядок матриц n был равен 5000, а число процессов P в первом случае принимало значения 8 (вычисления проводились на 1 узле), 16 (на 2 узлах), …, 48 (на 6 узлах), т.е. на каждом ядре каждого узла был запущен *MPI*-процесс. Во втором случае вычисления проводились таким образом, что на каждом узле кластера запускалось 4 процесса, т.е. число процессов P принимало значения 4 (вычисления проводились на 1 узле), 8 (на 2 узлах), …, 48 (на всех 12 узлах).

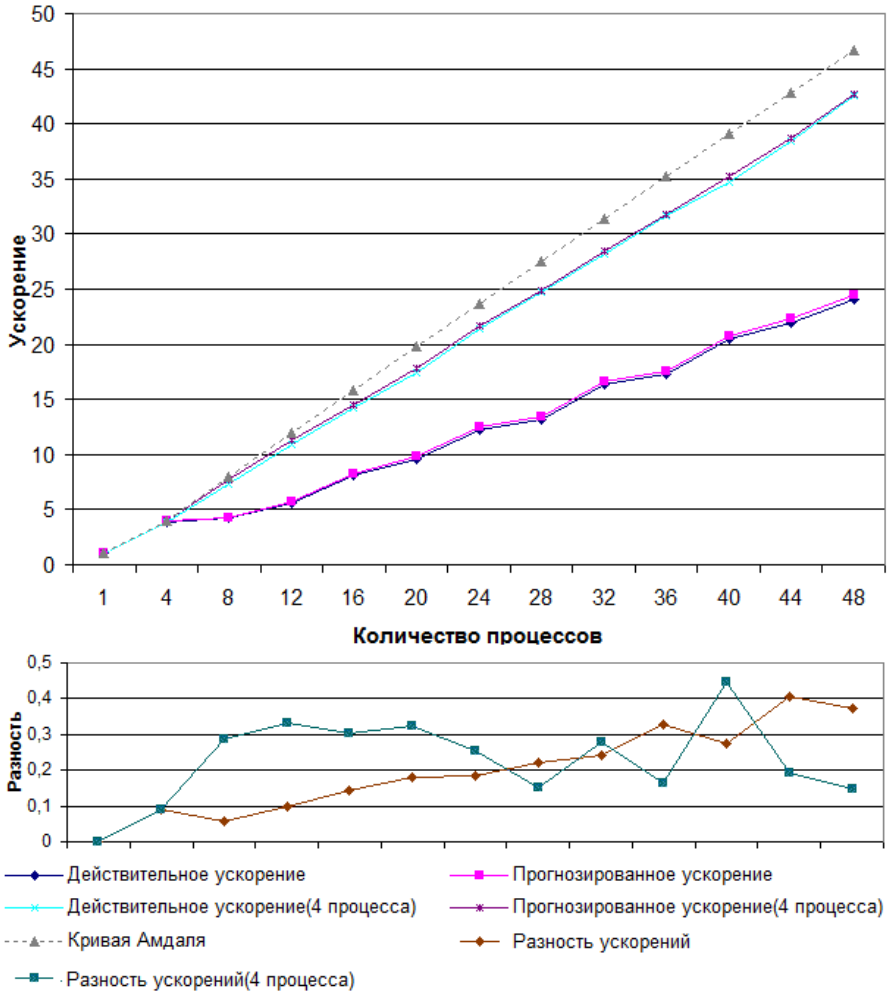

Рис. 1. Зависимость ускорения программы от числа используемых узлов кластера.

Использоваласъ 32-разрядная версия среды Java, так что каждый элемент матрицы занимал 8 байтов (тип `double`), а строка (столбец) матрицы – 40000 байтов. Запуск процессов (не потоков[4]) на каждом ядре процессора связан с

[4] В данной статье термин *поток* соответствует понятию «легковесный процесс» (англ. *thread*).

существенными накладными расходами, в частности, для интерпретации параллельной *Java*-программы на каждом ядре должна быть запущена своя *JavaVM*. В случае потоков это не так: в среде *Java* реализован системный класс *java.util.concurrent.ExecutorService*, обеспечивающий запуск требуемого числа потоков на одной *JavaVM*. Если число потоков не превышает числа ядер на узле, то каждый поток запускается на отдельном ядре.

На рисунке 2 представлены графики, на которых сравниваются ускорения параллельной *Java*-программы, запущенной на ядрах одного процессора, сначала с использованием *MPI*-процессов, а затем - *Java*-потоков. Графики показывают, что на одном узле использование потоков позволяет получить большее ускорение, чем в случае процессов.

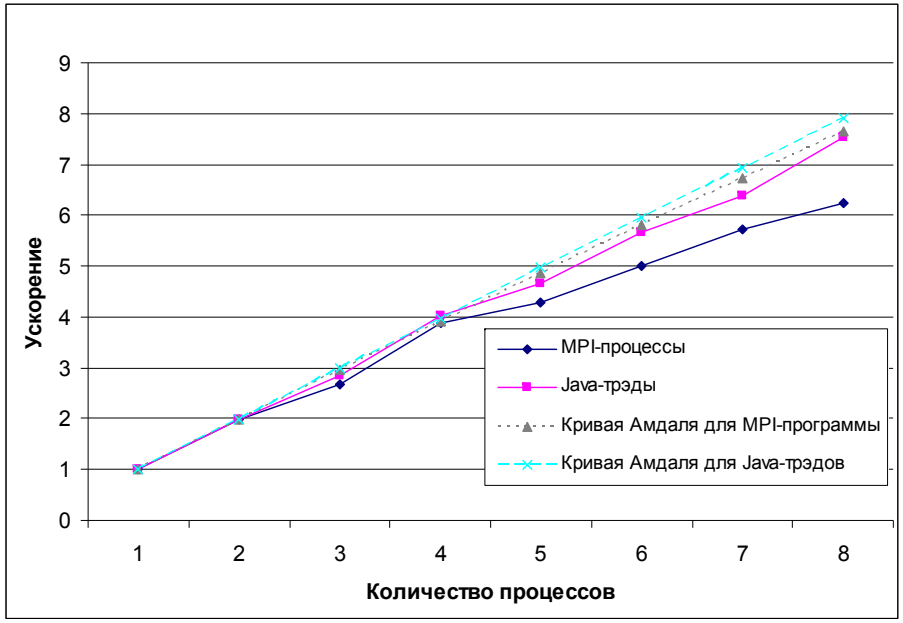

Рис. 2. Сравнение ускорения программы умножения матриц при ее выполнении на многоядерном узле кластера с использованием MPI-процессов или Java-потоков.

При выполнении MPI-программ на кластерах с многоядерными узлами оказалось, что хорошая масштабируемость получается в тех случаях, когда число процессов (или потоков), запущенных на одном узле кластера, меньше числа ядер на соответствующем узле. На рисунке 3 это явление иллюстрируется на примере программы умножения матриц (в данном примере порядок матриц n равен 5500). Видно, что с увеличением числа процессов, запущенных на узле, масштабируемость уменьшается сначала незначительно,

когда число запущенных процессов намного меньше числа ядер, а потом весьма существенно. Это явление можно объяснить резким возрастанием накладных расходов на синхронизацию обращения к данным параллельных процессов (потоков), запущенных на узле. Этот график был получен на кластере МСЦ РАН МВС-100К, который имеет узлы примерно такие же, что и кластер ИСП РАН (на каждом узле два 4-ядерных процессора Intel® Xeon® CPU X5365 с частотой 3.00GHz и с интерфейсной платой HP Mezzanine Infiniband DDR), но узлов у него гораздо больше (1410).

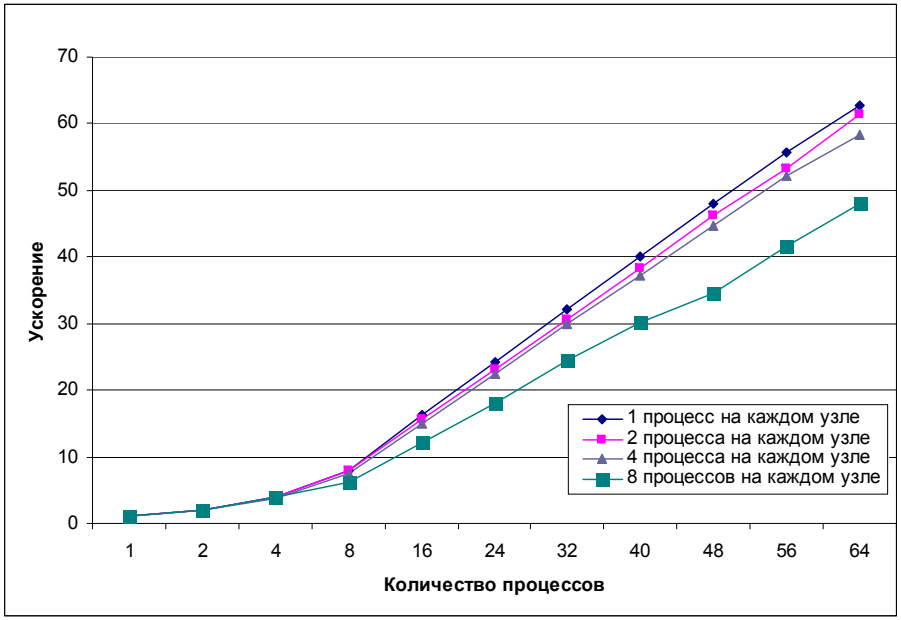

Рис. 3. Деградация масштабируемости параллельной программы с увеличением числа процессов (потоков), запущенных на многоядерном узле кластера.

Пример 2. Программа решения уравнения теплопроводности. Было рассмотрено линейное уравнение теплопроводности:

$$\frac{\partial}{\partial x}\left(k\frac{\partial u}{\partial x}\right)+F(x,t)=c\rho\frac{\partial u}{\partial t},$$

где $u(x, t)$ – температура в точке n-мерного пространства x в момент времени t, k – коэффициент теплопроводности, c – удельная теплоемкость, ρ - плотность (вообще говоря, параметры k, c и ρ могут зависеть от (x, t)).

В модельном примере рассматривался случай неоднородной плоской квадратной пластины ($n = 2$), т.е. рассматривалось уравнение

$$\frac{\partial}{\partial x}\left(k\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial u}{\partial y}\right)+F(x,y,t)=c\rho\frac{\partial u}{\partial t},$$

$$u(x,y,t),\quad 0<x<1,0<y<1,0<t.$$

в области $0 < x < 1$, $0 < y < 1$, $t > 0$. Начальные и граничные условия задавались следующим образом:

$$u\big|_{t=0}=\varphi(x,y),\quad 0\le x\le 1,0\le y\le 1,$$

$$u\big|_{x=0}=\psi_1(y,t),\ u\big|_{x=1}=\psi_2(y,t),\ 0\le y\le 1,0\le t,$$

$$u\big|_{y=0}=\psi_3(x,t),\ u\big|_{y=1}=\psi_4(x,t),\ 0\le x\le 1,0\le t.$$

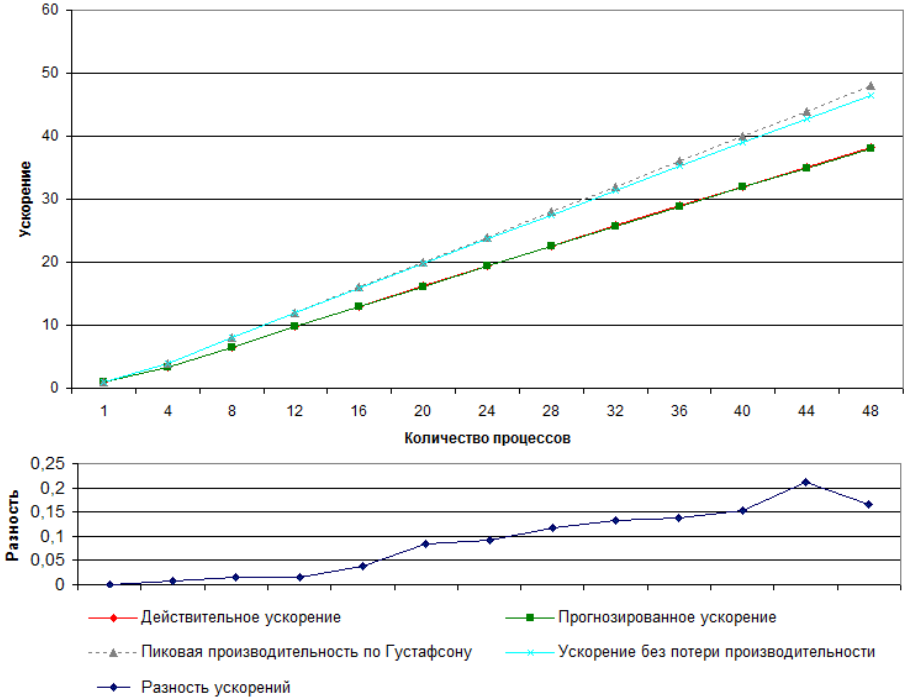

Рис. 4. Сравнение предсказанного и фактически полученного ускорения программы решения линейного уравнения теплопроводности на кластере ИСП РАН (n=6000).

В качестве граничных функций φ и ψ были выбраны $e^{-(x_1+y_1)}$ и $100*e^{-(x_1+y_1)}$ где $x_1=(x-½)^2$, $y_1=(y-½)^2$ (при соответствующих границе значений переменных). С помощью неявной разностной схемы задача свелась к системе линейных алгебраических уравнений, которая решалась итерационно с помощью одного из обобщений метода Зейделя.

На рисунке 4 представлены графики, подтверждающие высокую точность прогноза производительности программы при использовании заданного числа узлов с помощью соответствующего инструмента среды *ParJava*. Следует отметить, что максимально достижимое ускорение для данного числа узлов зависит от доли последовательных вычислений при выполнении программы и в случае фиксированного объема обрабатываемых данных определяется законом Амдаля [11]. Согласно закону Амдаля максимально достижимое ускорение $S_A(p)$ на p процессах определяется по формуле $S_A(p)=\frac{p}{1+(p-1)\cdot f}$, где f - доля не распараллеливаемой (последовательной) части программы. Для рассматриваемого примера f = 0,3%. Пунктирная кривая на рисунке – это график функции $S_A(p)$.

В работе Дж. Густафсона [12] было показано, что если объем обрабатываемых данных (в данном случае – это порядок матрицы n) меняется таким образом, чтобы объем обрабатываемых данных в каждом процессе был постоянным, то максимально достижимое ускорение определяется по формуле Густафсона $S_G(p)=p-(p-1)\cdot f$. Основное отличие функций Амдаля и Густафсона в их асимптотике: $\lim\limits_{p\to\infty} S_A(p)=\frac{1}{f}$, тогда как $\lim\limits_{p\to\infty} S_G(p)=\infty$.

На рисунке 5 представлены графики предсказанного и реально полученного ускорения для программы решения линейного уравнения теплопроводности на кластере ИСП РАН для случая, когда объем обрабатываемых данных в каждом процессе поддерживался постоянным (по Густафсону): в каждом процессе обрабатывалась матрица размеров $n \times k$ (n = *6000, k= 6000*). График показывает, что и в этом случае ускорение прогнозируется достаточно точно: относительная погрешность прогноза не превышает 1%. Для сравнения пунктиром показан график функции $S_G(p)$.

Пример 3. Прикладная параллельная программа численного решения системы уравнений, моделирующей процессы и условия генерации интенсивных атмосферных вихрей (ИАВ) в трехмерной сжимаемой атмосфере, исходя из теории мезомасштабных вихрей В.Н. Николаевского.

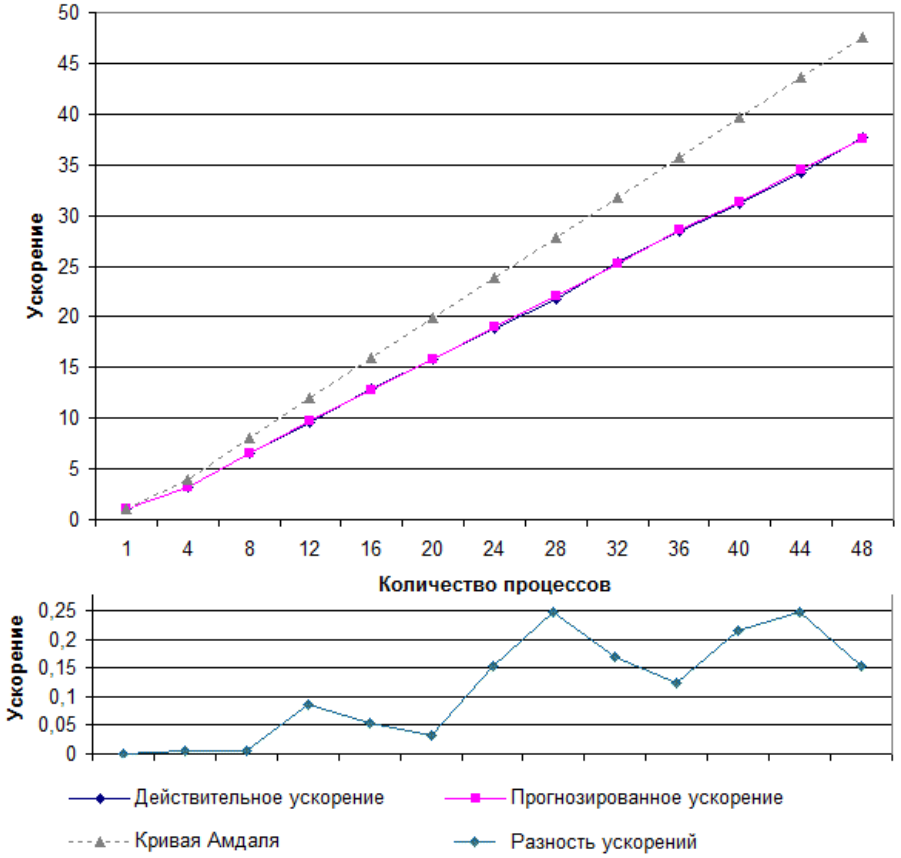

Рис. 5. Сравнение предсказанного и фактически полученного ускорения программы решения линейного уравнения теплопроводности по Густафсону.

Математическая модель описывается сильно нелинейной многокомпонентной трехмерной системой уравнений в частных производных смешанного типа, которая была выведена в работе [13]; аналитические решения ее неизвестны, а численное решение впервые было получено в работе [14]. Соответствующая параллельная программа была разработана и реализована в среде ParJava. В процессе разработки инструменты ParJava использовались для определения области масштабируемости и оптимального числа параллельных процессов, а также для настройки параллельной программы с целью ее оптимизации.

Исследования показали, что при выбранной схеме вычислений доля последовательных вычислений f = 0,83%. Разработанная программа показала достаточно высокую производительность и хорошую масштабируемость. На

рисунках приведены графики зависимости ускорения программы моделирования ИАВ для кластера ИСП РАН (рисунок 6) и МСЦ РАН (рисунок 7). Графики показывают достаточно хорошее совпадение предсказанных и реальных значений производительности программы. Моделировалась 64-разрядная версия программы на Java 1.6. Как видно из графика, начиная со 120 процессов, реальное ускорение программы идет на спад. Интерпретатор также фиксирует, что в данной точке производительность

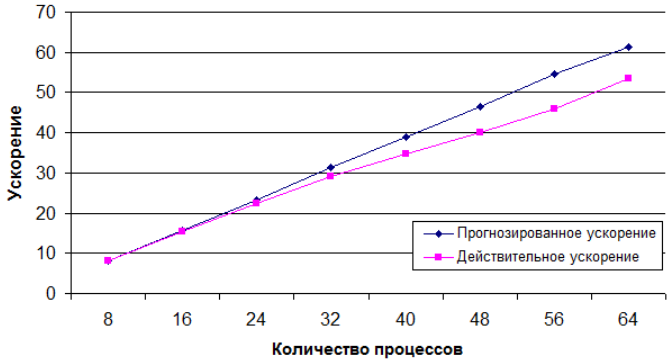

Рис. 6. Ускорение программы моделирования ИАВ на кластере ИСП РАН.

программы падает. Отметим, что такое падение производительности связано с асимптотикой формулы Амдала ($\lim_{p \to \Gamma} S_A(p) = \frac{1}{f}$). В самом деле, 1/0,0083 ≈ 120.

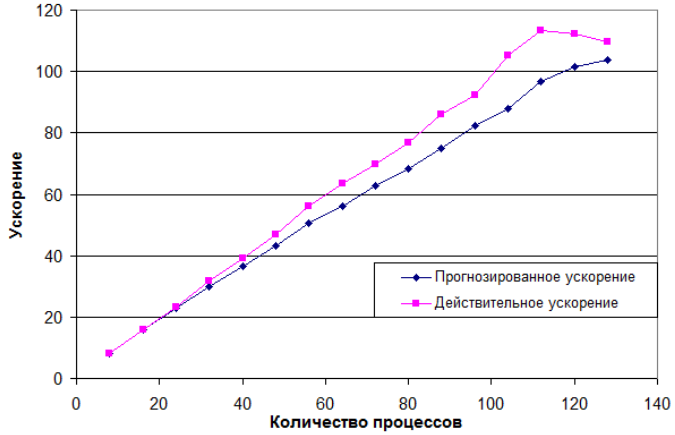

Рис. 7. Ускорение программы моделирования ИАВ на кластере МСЦ РАН.

7. Заключение.

Предложенные методы оценки времени выполнения гнезд циклов реализованы и интегрированы в инструментальную среду разработки параллельных программ ParJava [4]. Предложенный в данной статье подход позволяет существенно сократить степень участия прикладного программиста при получении указанных оценок и в то же время уменьшить погрешность предсказания времени выполнения. Эти методы применимы к реальным прикладным параллельным программам. Рассмотренные методы позволили не только получить приемлемую относительную погрешность оценки (погрешность прогноза не превышает 15%), но и ускорило время интерпретации модели.

Новые методы оценки времени выполнения гнезд циклов особенно актуальны при использовании кластеров с многоядерными узлами, так как позволяют учесть эффект потери производительности при чрезмерной загрузке узлов кластера и определить оптимальное число процессов, запускаемых на узлах.

Список литературы

[1] В.П. Иванников, А.И. Аветисян, С.С. Гайсарян, В.А. Падарян. Оценка динамических характеристик параллельной программы на модели. «Программирование» 2006, №4, с. 21–37

[2] Mark Baker, Bryan Carpenter, and Aamir Shafi. MPJ Express: Towards Thread Safe Java HPC, Submitted to the IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Spain, 25-28 September, 2006.

[3] Markus Bornemann , Rob V. Van Nieuwpoort , Thilo Kielmann. MPJ/Ibis: A Flexible and Efficient Message Passing Platform for Java. Euro PVM/MPI 2005, volume 3666

[4] Иванников В.П., Аветисян А.И., Гайсарян С.С., Акопян М.С. Особенности реализации интерпретатора параллельных программ в среде ParJava. «Программирование» 2009, №1, с. 10-25

[5] В.П. Иванников, А.И. Аветисян, С.С. Гайсарян, В.А. Падарян. Прогнозирование произ-водительности MPI-программ на основе моделей. «Автоматика и телемеханика», 2007, №5, с. 8-17

[6] А.И. Аветисян, В.В. Бабкова и А.Ю. Губарь. Возникновение торнадо: трехмерная численная модель в мезомасштабной теории турбулентности по В.Н. Николаевскому// ДАН/Геофизика, т. 419, №4, с. 547-552. Москва 2008.

[7] Alpern A.B, S. Augart, S.M. Blackburn, M. Butrico, A. Cocchi, P. Cheng, J. Dolby, S. Fink, D. Grove, M. Hind, K.S. Mckinley, M. Mergen, J.E.B. Moss, T. Ngo, V. Sarkar. The Jikes Research Virtual Machine project: IBM Systems Journal, Vol. 44, No 2, 2005

[8] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jaffrey D. Ullman. Compilers: principles, techniques, and tools. – 2nd ed., Pearson Education Inc., 2007, с.836.

[9] M. Paleczny, C. Vick, and C. Click. The Java HotSpotTM server compiler. In Proceedings of the Java Virtual Machine Research and Technology Symposium, pages 1–12, 2001.

[10] V.P. Ivannikov, A.I. Avetisyan, V.V. Babkova, A.Yu. Gubar “Tornado arising modeling using high performance cluster systems” Sixth International Conference on Computer Science and Information Technologies (CSIT’2007), 24-28 September, Yerevan, Armenia

[11] Amdahl G.M. Validity of single-processor approach to achieving large-scale computing capability, Proceedings of AFIPS Conference, Reston, VA. 1967. pp. 483-485
[12] Gustafson J.L., Reevaluating Amdahl's Law, CACM, 31(5), 1988. pp. 532-533.
[13] Аветисян А.И., Бабкова В., Гайсарян С.С., Губарь А.Ю.. Рождение торнадо в теории мезомасштабной турбулентности по Николаевскому. Трехмерная численная модель в ParJava. Журнал «Математическое моделирование», том 20, №8, с. 28-40, 2008
[14] Аветисян А.И., Бабкова В.В., Губарь А.Ю. «Моделирование интенсивных атмосферных вихрей в среде ParJava.» Всероссийская научная конференция «Научный сервис в сети Интернет: технологии параллельного программирования», г. Новороссийск, 2006. с. 109-112.