УДК 681.3.06

# ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ С РАСПРЕДЕЛЕНИЕМ ПО ДАННЫМ В СИСТЕМЕ PARJAVA

**А. И. Аветисян[1], И. В. Арапов[1], С. С. Гайсарян[1], В. А. Падарян[1]**

В статье дается общее описание среды ParJava [1], которая является расширением среды Java средствами разработки масштабируемых, эффективных, переносимых, объектно-ориентированных параллельных программ как для однородных, так и для неоднородных параллельных вычислительных систем с распределенной памятью. При этом инструментальная вычислительная система, на которой разрабатывается программа, может быть как однородной, так и неоднородной. Среда позволяет использовать алгоритмы, разработанные для однородных систем, на неоднородных системах без потери масштабируемости, т.е. делает их переносимыми. В состав среды включены низкоуровневые средства (библиотека Java-классов), обеспечивающие возможность разработки, реализации и выполнения параллельных программ в модели параллелизма по данным (SPMD) на однородных и неоднородных вычислительных системах. В дальнейшем эти средства позволят эффективно реализовывать объектные модели параллельного программирования более высокого уровня.

**Ключевые слова:** параллельное программирование, объектно-ориентированные программы, инструментальные вычислительные системы, параллельные вычислительные системы, модели параллелизма.

**1**. Мы изучаем возможности разработки и реализации параллельных программ для многопроцессорных вычислительных систем с распределенной памятью в модели SPMD [2]. Целью является реализация простых в использовании и эффективных средств разработки переносимых масштабируемых параллельных программ для таких систем. Для поддержки параллельности в системах с распределенной памятью обычно используется механизм процессов и передачи сообщений. Прикладные параллельные программы для суперкомпьютеров с массивно-параллельной архитектурой разрабатываются, как правило, на последовательных языках, расширенных библиотеками передачи сообщений. Массивно-параллельные системы состоят из одинаковых *вычислительных узлов*, включающих:

— один или несколько центральных процессоров (обычно RISC),

— локальную память (прямой доступ к памяти других узлов невозможен),

— коммуникационный процессор или сетевой адаптер,

— жесткие диски и/или другие устройства ввода/вывода.

К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.). Существуют различные проекты таких суперкомпьютеров: Option Red фирмы Intel [3], SR8000 Series фирмы Hitachi [4] др.; они различаются по типу используемых процессоров, однако основное их различие — подход к организации коммуникационных подсистем. Сложность используемой коммуникационной подсистемы определяет и стоимость проекта.

Самым производительным суперкомпьютером на данный момент является массивно-параллельная система ASCI White корпорации IBM [5]. Система объединяет 8192 процессора на медной основе с общей пиковой производительностью в 12,3 TFLOPS (триллионов операций в секунду). Она включает в общей сложности 8 TB (миллионов мегабайт) оперативной памяти, распределенной по 16-процессорным SMP-узлам, и 160 TB дисковой памяти. Все узлы системы работают под управлением ОС AIX — варианта UNIX от IBM. Среда программирования для ASCI White включает реализации интерфейсов MPI [6] и OpenMP [7] — международные стандарты интерфейсов передачи сообщений соответственно на распределенной и общей памяти.

В настоящее время в связи с появлением стандартного высокопроизводительного коммуникационного оборудования (Fast Ethernet [8], Myrinet [9] и др.) широкое распространение получают вычислительные

[1] Институт системного программирования РАН, ул. Б. Коммунистическая, 25, 109004, Москва; e-mail: arut@ispras.ru, ssg@ispras.ru

кластеры с распределенной памятью. Основное достоинство таких кластеров — их аппаратное обеспечение состоит из стандартных процессоров и коммуникационного оборудования массового производства, что позволяет использовать программное обеспечение, разработанное для локальных сетей (например, ОС Linux [10], различные реализации интерфейса MPI и др., не только коммерческие, но и свободно распространяемые). Следствием этого является высокое соотношение производительность/стоимость, возможность модификации путем замены микропроцессоров, коммуникационных сред и версий программного обеспечения. Примерами таких систем являются: Linux-кластер на основе двухпроцессорных серверов Eximer, кластер AliCE компании Compaq и др.

Linux-кластер, состоящий из 12 двухпроцессорных серверов Eximer с процессорами Pentium III/500, имеет пиковую производительность 12 GFLOPS. Кластер введен в эксплуатацию в НИВЦ МГУ [11]. Вычислительные узлы объединяются в двумерную решетку с помощью высокоскоростной сети SCI [12], а в качестве служебной сети используется Fast Ethernet. Максимальная скорость обмена данными по сети в приложениях пользователя составляет 80 Мбайт/с, а время задержки не превышает 5,6 мкс. В качестве основной среды параллельного программирования в кластере применяется MPI (Message Passing Interface) версии ScanMPI 1.9.1 [13]. Кластер работает под управлением операционной системы Linux Red Hat 6.1 [14]. Производительность на тесте при решении системы линейных уравнений с матрицей 16000 на 16000 элементов составляет 5.7 GFLOPS. На тестах NPB 2.3 [15] производительность сравнима с показателями суперкомпьютеров семейства CRAY T3E [16].

Кластер AliCE (Alpha Linux Cluster Engine) компании Compaq состоит из 128 процессоров Alpha 21264 [17], установленных в станциях DS10 [18], которые объединены в сеть Myrinet SAN (пропускная способность 1,28 Мбит/с при работе в дуплексном режиме). Параллельные приложения используют Myrinet для внутренних обменов, а для рабочих пересылок, управления и поддержки связи с Internet по протоколу TCP/IP применяются два канала Fast Ethernet. Пиковая производительность составляет 154 GFLOPS (80 GFLOPS по тестам Linpack [19]). Распределенная оперативная память — 16 Мбайт, дисковая — 1 Тбайт. Кластер работает под управлением операционной системы Suse Linux 6.3 [20].

В настоящее время используются также “вычислительные фермы” (“compute farms”), т.е. кластеры, все узлы которых размещены в одном корпусе. Примером вычислительной фермы является ферма на базе рабочих станций HP VISUALIZE J6000. Эта рабочая станция включает до двух процессоров PA-8600 и до 16 GB оперативной памяти. Система включает до 20 таких рабочих станций с общей пиковой производительностью до 88 GFLOPS и суммарной оперативной памятью до 320 GB. Рабочие станции объединяются с помощью коммутатора Cisco Catalyst 3524 XL.

Все вышеперечисленные кластеры используются в качестве дешевого варианта массивно-параллельного компьютера. Они представляют собой однородные вычислительные сети: все процессоры, расположенные в узлах сети, одинаковы и, следовательно, имеют одинаковую производительность, все каналы связи между процессорами имеют одинаковую пропускную способность. Производительность таких кластеров и ферм обычно имеет тот же порядок, что и производительность суперкомпьютеров предыдущего поколения. Стоимость кластеров имеет тот же порядок, что и теперешняя стоимость указанных суперкомпьютеров. Но кластеры требуют существенно меньших расходов на сопровождение, чем суперкомпьютеры, так как не предъявляют повышенных требований к квалификации персонала. Кроме того, кластеры имеют значительно более дешевое системное программное обеспечение.

При объединении в кластер компьютеров разной производительности или разной архитектуры говорят о гетерогенных (неоднородных) кластерах. Во избежание путаницы мы будем называть такие системы неоднородными вычислительными сетями (НВС). Такая ситуация возникает

— в случае постепенного расширения и модернизации кластеров или ферм;

— в случае использования локальных компьютерных сетей (в их состав входят рабочие станции, персональные компьютеры и т.п.) как кластеров.

В будущем в качестве параллельных систем с распределенной памятью, вероятно, можно будет рассматривать также системы, строящиеся из свободных ресурсов в Internet.

В суперкомпьютерах и кластерах мощности процессоров согласованы с пропускной способностью каналов связи. Это обстоятельство необходимо учитывать при построении НВС. Это означает, что не любая локальная компьютерная сеть может рассматриваться как НВС: необходимо, чтобы значения производительностей узлов и пропускных способностей каналов связи различались в допустимых пределах (см. ниже) и чтобы средняя производительность узла была согласована со средней пропускной способностью канала связи.

Для разработки программ для параллельных систем с распределенной памятью необходимы специальные языковые средства параллельного программирования. В настоящее время одним из наиболее

часто используемых языковых средств является международный стандарт интерфейса передачи сообщений MPI. Большинство параллельных программ как для суперкомпьютеров, так и для кластеров разрабатываются на языках Fortran+MPI, C+MPI. Сейчас происходит постепенный переход к объектно-ориентированной технологии разработки параллельных программ. В основном используется язык C++, расширенный библиотеками классов, реализующих функциональность MPI.

Набор функций MPI вобрал в себя лучшие черты предшествующих систем передачи сообщений p4 [21], PVM [22] и др. MPI поддерживает несколько режимов передачи данных, важнейшие из которых: синхронная передача, не требующая выделения промежуточных буферов для данных и обеспечивающая надежную передачу данных сколь угодно большого размера, и передача с буферизацией, при которой посылающий сообщение процесс не ждет начала приема, что позволяет эффективно передавать короткие сообщения. MPI позволяет передавать данные не только от одного процесса другому, но и поддерживает коллективные операции: широковещательную передачу, разборку-сборку (scatter-gather), операции редукции.

При разработке и реализации параллельных программ для НВС возникают специфические проблемы, связанные с неоднородностью вычислительной сети. Возникает потребность в средствах управления ресурсами, выделяемыми для выполнения параллельной программы (в частности, возникает необходимость динамического изменения ресурсов). Интерфейс MPI разработан для однородных систем, и в нем такие возможности не предусмотрены. Следовательно, НВС требуются дополнительные языковые средства для управления ресурсами системы.

Целью настоящей работы является общее описание среды ParJava, предназначенной для разработки параллельных Java-программ, которые могли бы выполняться как на однородных параллельных вычислительных системах (суперкомпьютерах и кластерах), так и на НВС. Среда ParJava включает в себя:

— языковые средства в виде набора интерфейсов и классов, расширяющие язык Java возможностями для разработки параллельных программ;

— инструментальные средства, помогающие в разработке и реализации эффективных масштабируемых параллельных программ.

В п. 2 вводится терминология и формулируется цель разработки среды ParJava. В п. 3 проводится сравнение среды Java с другими языковыми средами разработки программ и обосновывается возможность разработки эффективных параллельных программ в среде Java. В п. 4 приводится общее описание среды разработки параллельных программ ParJava. В п. 5 разбирается пример использования среды ParJava для разработки параллельной программы для однородной сети JavaVM и для НВС с балансировкой нагрузки: однородная сеть JavaVM моделировалась на неоднородной вычислительной системе. В заключительном п. 6 содержатся выводы и направления дальнейшего развития системы.

**2**. Перейдем к строгой постановке задачи разработки среды ParJava. Для этого нам понадобятся следующие определения.

*Однородной* будем называть вычислительную систему, в которой:

— однородны узлы;

— однородна коммуникационная среда, т.е. скорость передачи данных для всех существующих в системе каналов связи одинакова.

Узлы будем считать однородными, если у них:

— одинаковая аппаратная реализация и как следствие одинаковая производительность;

— одинаковая вычислительная среда, т.е. одни и те же:

- операционная система;
- компилятор;
- редактор связей;
- символьный отладчик;
- инструментальная среда программирования;
- разделяемые библиотеки;
- библиотеки поддержки времени выполнения;
- программные средства коммуникационной среды;
- программные средства организации параллельных вычислений PVM, MPI и др.

В противном случае систему будем называть *неоднородной*.

Следовательно, неоднородность распределенной вычислительной системы может быть трех видов:

— неоднородность вычислительной среды;

— неоднородность по производительности узлов;

— неоднородность среды коммуникации.

Производительность каждого узла параллельной вычислительной системы (а также производительность всей вычислительной системы) измеряется с помощью специальных программ, называемых бенчмарками. Это связано с тем, что производительность системы может существенно различаться на разных классах задач. Поэтому для каждого класса задач используются свои бенчмарки. В среде Java разработаны бенчмарки для многих классов задач [23]. Набор этих бенчмарков постоянно пополняется. Бенчмарки для однородных и неоднородных сетей Java-исполнителей могут быть разработаны на основе бенчмарков на языках Fortran+MPI, C+MPI для параллельных вычислений в модели SPMD. Бенчмарки дают возможность измерить относительные производительности узлов неоднородной сети на рассматриваемом классе задач.

При выполнении программы, кроме чистого времени выполнения программы, возникают также накладные расходы. Они возникают в связи с работой операционной системы, библиотек поддержки (например, MPI), JavaVM и т.д. Если накладные расходы ограничены и достаточно малы, то программу будем называть *эффективной*.

Параллельная вычислительная система называется *масштабируемой*, если производительность системы пропорциональна сумме производительностей всех ее узлов. Для однородной системы масштабируемость можно определить и как пропорциональность ее производительности числу ее узлов. Для каждой масштабируемой вычислительной системы существует максимальное число узлов (оно определяется архитектурой системы) такое, что при большем числе узлов система перестает быть масштабируемой. *Масштабируемость* параллельной программы — это линейная зависимость скорости ее выполнения от производительности масштабируемой вычислительной системы.

Если система неоднородна по производительностям узлов, то ее масштабируемость зависит не только от количества ее узлов, но и от их *относительной производительности* (отношения производительности данного узла к суммарной производительности системы). Вычислительные эксперименты показывают, что когда относительная производительность одного или нескольких узлов системы меньше некоторого числа $p$, система перестает быть масштабируемой. Число $p$ для каждой системы может быть определено с помощью вычислительных экспериментов на бенчмарках. Мы будем называть его *допустимой степенью неоднородности системы*.

Под *переносимостью* последовательной программы мы будем понимать возможность ее выполнения на любой платформе без каких-либо модификаций или преобразований, а также без потери эффективности.

Под *переносимостью* параллельной программы мы будем понимать возможность ее выполнения на любой масштабируемой вычислительной системе без каких-либо модификаций или преобразований, с сохранением условия масштабируемости. Параллельные программы могут быть переносимы:

— на однородных вычислительных системах;

— на неоднородных вычислительных системах;

— на произвольных вычислительных системах.

Для реализации параллельных программ на неоднородных системах можно использовать следующие подходы:

— балансировка нагрузки на неоднородной сети;

— моделирование однородной сети на НВС.

Очевидно, что в первом случае накладные расходы меньше. Однако моделирование однородной сети снимает проблему переносимости параллельных алгоритмов, разработанных для однородных систем.

В дальнейшем, употребляя введенные термины, мы будем вкладывать в них только тот смысл, который был определен выше.

Таким образом, ставится задача: предоставить прикладному программисту простые, удобные и эффективные средства, позволяющие разрабатывать *масштабируемые*, *эффективные*, *переносимые*, *объектно-ориентированные* параллельные программы, которые могли бы выполняться как на однородных параллельных вычислительных системах (суперкомпьютерах и кластерах), так и на НВС.

**3**. Для решения поставленной задачи в качестве среды реализации выбрана среда Java [24]. Основным достоинством Java является полная переносимость программ — программа, написанная с использованием спецификаций системы Java и откомпилированная на одной из платформ, работает на любой

программно-аппаратной платформе, на которой установлена система Java, без каких-либо модификаций или преобразований.

Последовательную С-программу можно сделать переносимой только внутри фиксированного множества платформ, которое должно задаваться разработчиком с помощью директив псевдокомпиляции, использование которых позволяет разрабатывать программу, выполнимую на всех платформах из указанного множества. Такая же техника может обеспечить переносимость параллельных С-программ в однородных и неоднородных средах из указанного множества платформ, если реализована соответствующая коммуникационная среда (например, библиотека MPI, которая осуществляет передачу данных между всеми этими платформами). Использование языка Java позволяет разрабатывать последовательные и параллельные программы, переносимые между любыми однородными и неоднородными параллельными системами, на которых установлена JavaVM.

К положительным свойствам системы Java в аспекте разработки параллельных программ также относятся:

— простота объектной модели, которая позволяет легко разбираться в исходной программе и производить изменения, дополнения и документирование программы;

— детерминированность получаемого кода — свойство языка, не позволяющее программисту написать "неправильную" программу, т.е. программу, в которой возможны несанкционированные действия типа обращения к объектам с помощью "висячих" указателей, преобразования случайного участка памяти в экземпляр объекта в результате ошибки в адресной арифметике и т.п.

Система программирования Java отвечает современным требованиям: поддержка исключительных ситуаций, наличие легковесных процессов (трэдов), потоковый ввод/вывод, поддержка средств сетевого программирования (библиотека сокетов), графический интерфейс пользователя, системы обеспечения безопасности при пересылке данных и байт-кода по сети и т.д. Кроме того, возможность написания параллельных программ на Java дают возможность пользователям, ранее не имевшим доступа к параллельным вычислительным ресурсам, широко использовать параллельные высокопроизводительные вычислительные комплексы, доступные через Internet.

Основной недостаток среды Java — сравнительно низкая производительность виртуального процессора. Однако использование JIT-компиляторов (или компиляторов реального времени), интегрированных в JavaVM, позволяет получать программы, работающие с такой же скоростью, что и Java-программы, откомпилированные в код компьютера, и примерно с такой же скоростью, что и программы на языке C/C++ (как уже было отмечено, проигрыш в производительности составляет всего 1,5 раза). При этом в JIT-компиляторах используются результаты профилирования программы, что позволяет определить ее узкие места и существенно повысить качество оптимизации. Кроме того, существует большое количество машинно-зависимых компиляторов (native compilers), осуществляющих компиляцию Java-программ в выполняемый двоичный код.

Такие компиляторы входят в состав следующих сред разработки:

— Code Warrior Professional 2.0 фирмы Metrowerks;

— JBuilder Client/Server Suite фирмы Borland;

— Java Workshop фирмы Sun;

— Power/J Enterprise 2.x фирмы Sybase;

— SuperCede for Java 2.0 Professional Edition фирмы SuperCede;

— Visual Café for Java 2.x (Professional Edition или Database Development Edition) фирмы Symantec.

Следует отметить, что даже прямые статические компиляторы для Java не позволяют достичь такого же уровня производительности, который возможен в среде C/C++. Это объясняется тем, что Java-программа остается объектно-ориентированной и во время выполнения (в отличие от программ C++). Это обеспечивает большую гибкость Java-программ по сравнению с программами среды C/C++ (например, из-за возможности использования рефлексии), но это является причиной того, что технологии статической оптимизации, хорошо зарекомендовавшие себя в компиляторах традиционных языков программирования, далеко не всегда могут столь же эффективно применяться при компиляции Java-программ. Это можно рассматривать как плату за преимущества использования объектно-ориентированной технологии, которые могут оказаться важнее различий в производительности, в том числе для параллельных программ.

Сравнительные результаты производительности на некоторых бенчмарках для различных реализаций JavaVM и оптимизированного кода C приведены в табл. 1.

В среде Java параллельные программы реализуются на сети, состоящей из виртуальных машин Java. Это позволяет использовать параллельные Java-программы на произвольных параллельных системах (однородных, неоднородных и даже системах, которые могут быть построены из свободных ресурсов

Таблица 1

Сравнение результатов производительности на бенчмарках

| | sieve | loop | memory | method | float point | linpack |
|---|---|---|---|---|---|---|
| Optimized C Code | 121 | 830 | 378 | 11029 | 276 | 21730 |
| Java 1.0.2 | 13 | 26 | 7 | 55 | 60 | 641 |
| Java 1.1.6 | 20 | 45 | 10 | 109 | 91 | 990 |
| Java 1.2 beta 4 | 189 | 203 | 109 | 135 | 65 | 10630 |
| Java 1.2.1 | 211 | 412 | 205 | 141 | 107 | 13895 |

в Internet). Единственным условием для этого является наличие на каждом узле системы виртуальной машины Java.

Таким образом, для реализации параллельных программ на сетях JavaVM можно использовать следующие подходы:

— запуск на каждом узле системы по одной JavaVM и использование полученной неоднородной сети JavaVM;

— моделирование однородной сети Java-исполнителей на неоднородной параллельной вычислительной системе.

**4**. В среде Java запрещены любые изменения спецификаций как языка Java, так и его интерпретатора JavaVM. Это правильное решение, так как такие изменения сделали бы практически невозможным выполнение Java-программ в произвольной сети и повлекли бы за собой серьезные ограничения в переносимости программ. Поэтому в работе рассматривается функциональное расширение языка Java с помощью библиотек системных классов, добавляющих возможности низкоуровневого параллельного программирования с распараллеливанием по данным для систем с распределенной памятью, без изменения спецификации самого языка.

Иерархия интерфейсов и классов, расширяющих язык Java низкоуровневыми возможностями реализации параллельных программ на однородных и неоднородных вычислительных системах, приведена на рис. 1.

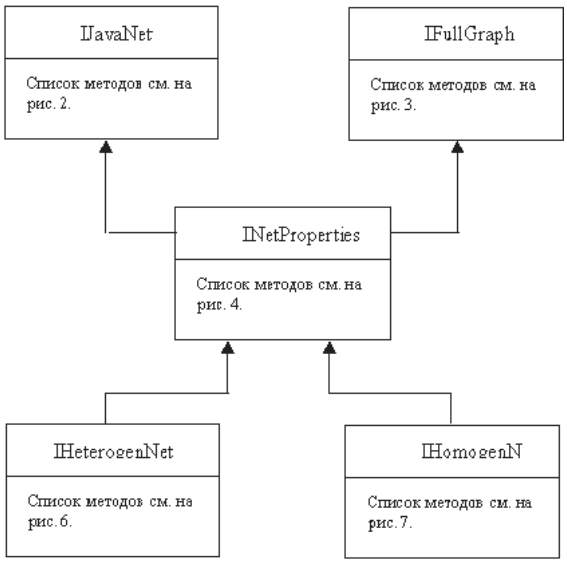


Рис. 1. Иерархия интерфейсов, расширяющих язык Java средствами параллельного программирования в модели SPMD

В этих интерфейсах вводится понятие вычислительного пространства как Java-сети, на которой выполняется параллельная программа. Каждой сети ставится в соответствие взвешенный граф, который отражает свойства этой сети (узлы — производительности Java-исполнителей, дуги — пропускные спо-

собности каналов связи). Определены интерфейсы и классы, позволяющие моделировать однородную сеть Java-исполнителей на неоднородной параллельной вычислительной системе, а также реализовывать параллельные программы на неоднородных сетях Java-исполнителей. При этом в обоих случаях производится балансировка нагрузки с учетом производительностей процессоров и пропускных способностей каналов связи. В интерфейсе IJavaNet (рис. 2) специфицируются операции (методы) порождения сетей Java-исполнителей и их подсетей, а также операции их преобразования.

```
public interface IJavaNet{
public IJavaNet createNet(int[ ] executor); // создать подсеть
public IJavaNet createNet(int first, int last, int stride); // создать подсеть
public IJavaNet cloneNet(); // построить сеть совпадающую с данной
public int size(); //получить количество узлов в сети
public int rank(); //получить порядковый номер узла в сети
public boolean isIncl(); //
public IJavaNet union(IJavaNet net); //получить объединение net и данной сети
public IJavaNet intersection(IJavaNet net); //получить пересечение net и данной сетей
public IJavaNet complement(); //получить дополнение данной сети до родительской
public IJavaNet getParent(); //получить родительскую сеть
public IJavaNet getRoot(); //получить корневую (root) сеть в иерархии
}
```

Рис. 2. Интерфейс IJavaNet

Под *сетью Java-исполнителей* понимается неструктурированное пронумерованное множество Java-исполнителей, которые могут обмениваться сообщениями. Никаких предположений относительно однородности сети не делается (для задания однородности или неоднородности сети используются интерфейсы, специфицируемые ниже).

Каждая подсеть Java-исполнителей также является сетью, имеет родительскую сеть (подсеть) и может иметь произвольное количество дочерних подсетей. Внутри одной сети (подсети) отсутствуют ограничения на обмен данными между узлами. Сеть Java-исполнителей, у которой нет родительской, назовем корневой. Интерфейс IJavaNet включает в себя следующие методы:

— методы createNet позволяют создать сеть (подсеть) Java-исполнителей из узлов текущей сети; параметром первого метода createNet является массив, содержащий номера узлов текущей сети, которые требуется включить в формируемую подсеть; второй метод createNet имеет три параметра, которые задают соответственно номер первого узла текущей сети, номер ее последнего узла и целое число, равное постоянному приращению номеров узлов;

— метод cloneNet создает подсеть, совпадающую с данной сетью;

— метод size позволяет получить количество узлов сети;

— метод rank позволяет получить порядковый номер узла в сети;

— метод getParent позволяет получить родительскую сеть для данной;

— метод getRoot позволяет получить корневую сеть в иерархии;

— операции над сетями:

- метод union создает новую сеть как объединение указанной и данной сетей;
- метод intersection создает новую сеть как пересечение указанной и данной сетей;
- метод complitment создает новую сеть, которая является дополнением данной сети.

При реализации программ с использованием класса JavaNet, реализующего интерфейс IJavaNet, вычислительное пространство создается запуском на каждом узле параллельной вычислительной системы одного Java-исполнителя. В интерфейсе IJavaNet не специфицированы возможности определения неоднородности вычислительного пространства. Предполагается, что параллельная программа выполняется на однородной вычислительной системе, в которой все узлы связаны со всеми.

```
public interface IFullGraph {
public int getGraphSize();
public void setNode(int node,int value);
public void setNodes(int[ ] values);
```

```
public void setNodes(int first,int stride,int[ ] values);
public int getNode(int node);
public int[ ]getNodes();
public int[ ]getNodes(int first,int stride);
public void setEdge(int from,int to,int value);
public void setEdges(int[ ][ ] values);
public void setEdges(int first_from,int first_to,int
stride_from,int stride_to,int[ ][ ] values);
public int getEdge(int from,int to);
public int[ ][ ] getEdges();
public int[ ][ ] getEdges(int first_from,int first_to,int stride_from,int stride_to);
public void setEdgesFrom(int from,int[ ] values);
public void setEdgesTo(int to,int[ ] values);
}
```

Рис. 3. Интерфейс IFullGraph

Отметим, что вычислительное пространство (Java-сеть) можно представить в виде полного взвешенного графа, в котором вес ребра соответствует скорости передачи данных между Java-исполнителями, а вес вершины — относительной производительности Java-исполнителя. Масштабируемые параллельные программы (SPMD) могут выполняться на сетях с различным числом узлов, поэтому программа не должна содержать описание сети, на которой она будет выполняться. В ней могут содержаться лишь требования к локальной структуре (топологии) сети. Структура требуемой сети, вообще говоря, определяется тем, между какими узлами происходит обмен сообщениями, и может быть получена из кода программы, либо может специфицироваться пользователем. При этом сеть считается однородной: вес каждого узла равен 1, пропускная способность каждого канала равна 1.

Методы интерфейса IFullGraph (рис. 3) специфицируют понятие полного взвешенного графа.

Интерфейс INetProperties (рис. 4) расширяет интерфейсы IJavaNet и IFullGraph. В нем специфицируются методы, позволяющие определять свойства Java-сети, в случае если она неоднородна. Эти свойства бывают необходимы как для моделирования оптимальной однородной сети, так и для моделирования оптимальной неоднородной сети Java-исполнителей на данной неоднородной Java-сети. Кроме того, в интерфейс INetProperties входят методы, определяющие требования программы к топологии вычислительного пространства (линейка, двумерная решетка, звезда и др.). Интерфейс IJavaNet включает в себя следующие методы:

— метод createFullGraph строит полный взвешенный граф, соответствующий данной Java-сети;

— метод getP позволяет получить значение коэффициента неоднородности [1] для данной сети;

— метод excludeNodes позволяет получить из данной неоднородной сети Java-исполнителей новую сеть, в которой исключены все узлы, не соответствующие условию масштабируемости;

— методы net_1dGrid, net_2dGrid, метод net_star дают возможность задать топологию формируемой сети (линейка Java-исполнителей, двумерная решетка, или звезда). В дальнейшем список таких методов будет расширен.

```
public interface INetProperties extends IJavaNet,IFullGraph {
public void createFullGraph();
public int getP();
public NetProperties excludeNodes();
public void net_1dGrid();
public void net_2dGrid();
public void net_star();
}
```

Рис. 4. Интерфейс INetProperties

**5**. В качестве прототипа среды коммуникации используется библиотека Java-классов, реализующих объектно-ориентированный интерфейс MPI, который был создан для языка C++ [25]. Современное соотношение Java-интерфейса и пакета MPI показано на рис. 5. Этот интерфейс может использоваться в Java-программах отдельно от всей системы.

```
public class Cartcomm extends Intracomm;
```

```
public class CartParms;
public class Comm;
public class Datatype;
public class Errhandler;
public class Graphcomm extends Intracomm;
public class GraphParms;
public class Group;
public class Info;
public class Intercomm extends Comm;
public class Intracomm extends Comm;
private class Maxlock extends User_function;
private class Minlock extends User_function;
public class MPI;
public class MPIException;
public class Op;
public class Prequest extends Request;
public class Request;
public class ShiftParms;
public class Status;
public class User_function;
```

Рис. 5. Java-интерфейс к пакету MPI

В настоящее время в состав среды ParJava включена прототипная реализация интерфейса JavaMPI. Соответствующие Java-классы реализованы посредством Java Native Interface [25] с помощью вызовов стандартных функций языка C++, реализующих соответствующую функциональность MPI (используется свободно распространяемая версия MPI lam6.3 [26]). В дальнейшем этот интерфейс будет реализован на Java.

**6**. При реализации параллельных программ на неоднородных сетях Java-исполнителей возникает необходимость неравномерного распределения данных с целью эффективного использования ресурсов системы. Отметим, что в отличие от высокоуровневых моделей параллельного программирования, где такое распределение автоматически делается системой поддержки, а способ распределения задается примитивами модели, в низкоуровневых моделях этим необходимо заниматься вручную. Хотелось бы предоставить пользователю простые и эффективные инструменты. Для этого интерфейс INetProperties расширен интерфейсом IHeterogenNet (рис. 6), в котором специфицируются следующие методы:

— метод relativePerformanceHost вычисляет относительную производительность узла с запрашиваемым номером;

— метод myrelativePerformance вычисляет относительную производительность текущего узла;

— метод netpower вычисляет сумму относительных мощностей всех узлов сети;

— метод rectstaken вычисляет сумму относительных производительностей узлов неоднородной сети с первого до текущего узла включительно;

— метод createHeterogenNet создает неоднородную сеть, в которой на каждом физическом узле системы запущена одна JavaVM, исключены узлы, не соответствующие условию масштабируемости, и которая оптимизирована в соответствии с требованием программы к топологии сети;

— метод heterogenGather собирает в приемный буфер данного узла передающие буферы остальных узлов (позволяет задавать разное количество отправляемых данных в разных узлах-отправителях);

— метод heterogenScatter рассылает части передающего буфера неодинаковой длины в приемные буферы неодинаковой длины.

После того как оптимальная сеть Java-исполнителей будет создана (при ее создании учитываются производительности узлов, измеренные на бенчмарках, и требования программы к топологии сети, моделируемой на данной параллельной вычислительной системе), можно достичь более эффективного использования ресурсов системы, распределяя данные пропорционально производительности ее узлов. Однако в случае, когда время выполнения последовательных частей параллельной программы (SPMD) сравнимо со временем выполнения ее параллельных частей, этого не достаточно. В этом случае на этапе ее разработки и отладки программы необходимо оценить затраты времени на выполнение последовательных частей, что даст возможность учесть это время при вычислении относительных производительностей узлов и тем самым обеспечить более высокую производительность сети при выполнении программы. Для

этого можно воспользоваться инструментами Instrumentate и Profile меню Analyzers среды ParJava.

```
public interface IHeterogenNet extends INetProperties{
public void relativePerformanceHost(int hostnumber);
public void myrelativePerformance();
public void createHeterogenNet();
public void heterogenGather(Object sendbuf, int sendcount, Datatype sendtype, Object recvbuf, int recvcounts[
], int displs[ ], Datatype recvtype, int root);
public void heterogenScatter(Object sendbuf, int sendcounts[ ], int displs[ ],
Datatype sendtype, Object recvbuf, int recvcount, Datatype recvtype, int root);
}
```

Рис. 6. Интерфейс IHeterogenNet

**7**. Интерфейс IHomogenNet (рис. 7) расширяет интерфейс INetProperties. В нем определены методы, позволяющие на данной неоднородной сети создавать однородную сеть Java исполнителей:

— метод getMinPermissibleNumberHosts вычисляет минимальное допустимое количество узлов в однородной сети, моделируемой на данной неоднородной сети;

— метод getMaxPermissibleNumberHosts вычисляет максимальное допустимое количество узлов в однородной сети, моделируемой на данной неоднородной сети;

— метод createHomogenNet с параметром hostsnumber создает однородную сеть с заданным количеством узлов на данной неоднородной сети;

— метод createHomogenNet создает оптимальную однородную сеть на данной неоднородной сети.

Однородные сети моделируются с учетом требований программы к топологии сети.

```
public interface IHomogenNet extends INetProperties {
public int getMinPermissibleNumberHosts();
public int getMaxPermissibleNumberHosts();
public IHomogenNet createHomogenNet();
public IHomogenNet createHomogenNet(int hostsnumber);
}
```

Рис. 7. Интерфейс IHomogenNet

**8**. Реализована среда ParJava, позволяющая редактировать, отлаживать и выполнять параллельные программы на однородных и неоднородных сетях JavaVM, а также моделировать однородные сети JavaVM на неоднородных вычислительных системах. Среда запускается на одном из узлов параллельной вычислительной системы (в дальнейшем мы будем называть этот узел *корневым*; при отображении списка доступных узлов этот узел помечается словом root). На рис. 8 показана часть главного окна системы во время редактирования параллельной программы, разрабатываемой в среде ParJava.

В тексте редактируемой программы все вызовы параллельных функций выделяются отдельным цветом (на рис. 8 они выглядят более бледными). Кроме стандартных возможностей по редактированию последовательной программы (окна File, Edit, View, Goto, Help), в среде ParJava имеются средства, поддерживающие разработку и выполнение параллельных программ на параллельной вычислительной системе (в частности, на локальной сети), на которой установлена среда ParJava. Часть из этих средств содержится в меню Tools (на рис. 8 показано выпадающее окно с этим меню). Меню Tools содержит следующие инструменты:

— New Net: выделить подсеть; при вызове этого пункта на экране появится диалоговое окно со списком идентификаторов доступных узлов параллельной вычислительной системы; пользователь отмечает узлы, из которых он хочет сформировать сеть; на этой сети и будет запускаться параллельная программа;

— Hosts Performance: определить относительную производительность узлов выделенной сети; при определении относительной производительности узлов учитываются параметры программы, определенные при анализе программы (меню Analyzers);

— Compile: скомпилировать SPMD-программу; запускается компилятор Java из среды JDK 1.2 [25];

— Run: выполнить программу в последовательном режиме на корневом узле;

— Run on Homogeneous Net: запустить SPMD-программу на однородной сети JavaVM, моделируемой на текущей подсети с оптимальным или заданным пользователем количеством узлов;

— Run on Heterogeneous Net: запустить SPMD-программу на неоднородной сети JavaVM; на каждом узле системы запускается одна JavaVM; узлы, не соответствующие условию масштабируемости, исключаются;

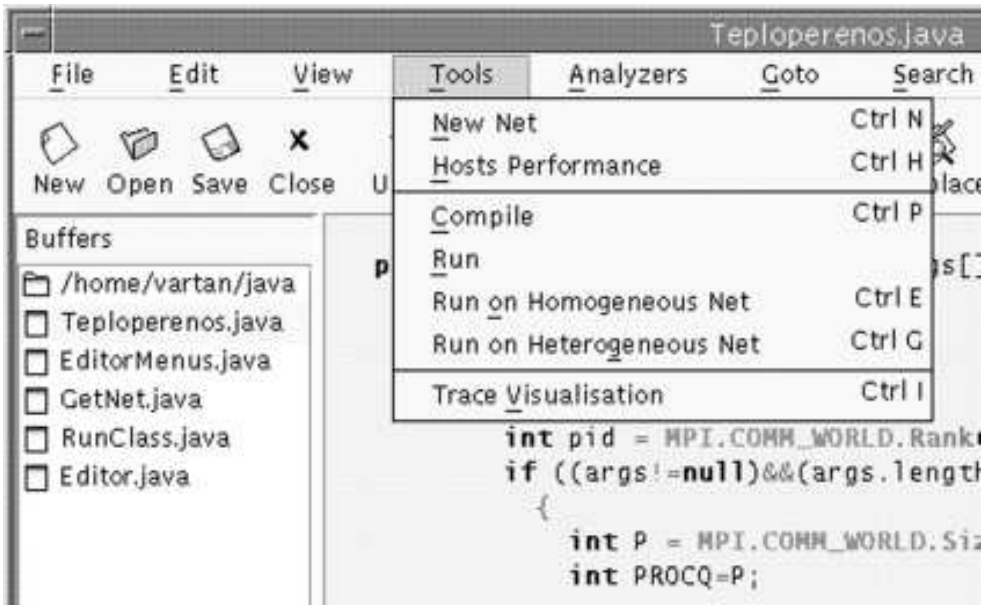


Рис. 8. Часть главного окна системы ParJava во время редактирования параллельной программы

— Trace Visualization (см. ниже).

Основным требованием к параллельной программе является ее масштабируемость. При выполнении параллельной программы на неоднородной вычислительной сети возникают вопросы, связанные с эффективным использованием ресурсов системы. Для обеспечения масштабируемости программы разработчику полезно знать ее профили. Для эффективного распределения программы по узлам неоднородной сети необходимо знать значения параметров программы, определяющих фактическую скорость ее выполнения на каждом узле сети.

Для определения этих свойств программы в среду ParJava (меню Analyzers) включены следующие инструменты:

— Instrumentate: инструментирует программу, т.е. вставляет в нее отладочные обращения к таймеру и выдачи;

— Profile: составляет динамический профиль программы;

— Test mode: переводит параллельную программу в тестовый режим, в котором она будет выполняться (окно Tools) с использованием специальной отладочной библиотеки.

Средства Instrumentate и Profile позволяют оценить затраты времени на выполнение последовательной части программы на этапе ее разработки и отладки. Это позволяет учитывать вес последовательной части при вычислении производительности узлов, обеспечивая большую производительность сети при выполнении программы.

В режиме Test mode каждая ветвь параллельной программы накапливает историю параллельного исполнения в специальном файле. В этом файле отражаются все события, связанные с параллельным исполнением, а также абсолютное время выполнения как ветви в целом, так и время между параллельными событиями в этой ветви. К таким событиям, например, относятся все вызовы коммуникационных функций. Специальный файл содержит также дополнительную информацию о событии (размер сообщений, от кого кому, номер строки в исходной Java программе и т.д.). Действия по построению этого файла осуществляются автоматически при помощи отладочной библиотеки и не требуют дополнительных усилий со стороны пользователя. Визуализировать накопленную во время выполнения параллельной программы историю можно из среды, выбрав подпункт Trace Vizualization, пункта меню Tools. Пример такой визуализации показан на рис. 9.

На приведенном рисунке можно увидеть, как программа Trace Visualization демонстрирует работу семи процессов. Здесь каждая линия — это отдельный процесс параллельной программы. Каждый квадратик на линии — вызов одной из функций MPI. Снизу показана временная линейка. При помощи нее можно определить (в миллисекундах), когда произошел вызов той или иной функции. Видно, например, что первый процесс закончился раньше, а последний — позже всех.

Функции на схеме, ради экономии места и удобства представления, отображаются не названиями, а

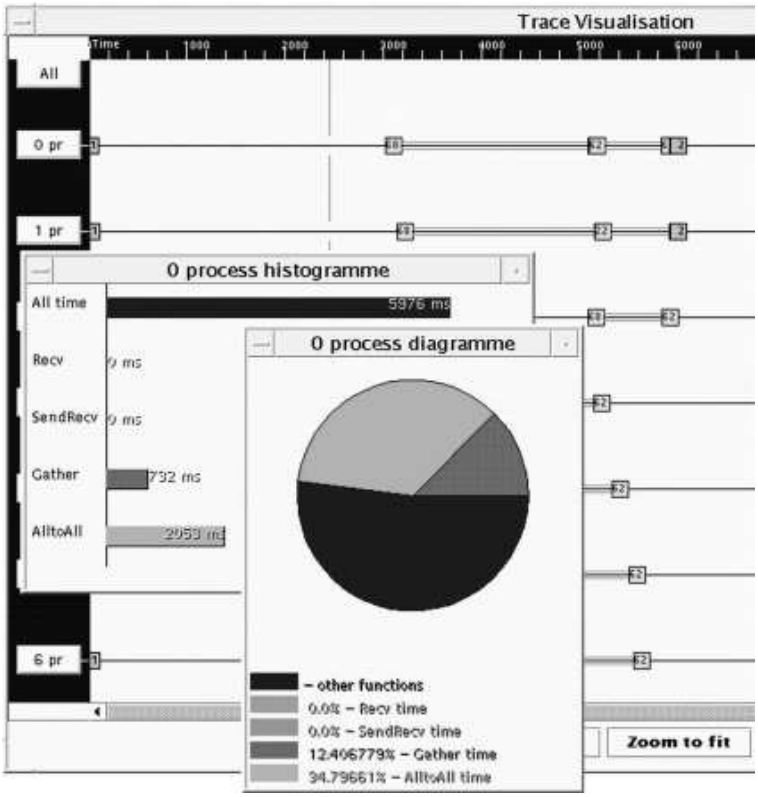


Рис. 9. Пример визуализации процессов

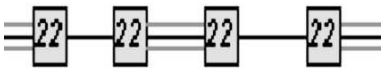


Рис. 10. Ожидание данных

номерами (например, функциям Init() и Finalize() сопоставлены коды 1 и 2 соответственно). При наводке зеленого курсора на тот или иной квадрат с кодом функции появляется подсказка с именем функции и относительным временем ее вызова. Работа параллельной программы начинается с запуска функции Init() (код 1), а заканчивается функцией Finalize() (код 2). На схеме эти обрамляющие функции выделены оранжевым цветом.

На схеме некоторые квадратики соединены зелеными линиями; длина зеленой линии пропорциональна времени ожидания внешнего события (например, ожидание данных). На рис. 10 показано, как процесс вызвал функцию получения массива данных Recv (код 22), подождал их и вышел из стадии ожидания, получив нужную порцию данных. Время ожидания выделено зеленым цветом.

Из рис. 9 видно, что процесс номер 5 несколько раз вызывает функцию получения данных и почти половину времени своего выполнения тратит на их ожидание.

Процент времени простоя каждого процесса будет вычислен и отображен (в процентах) на диаграмме любого из процессов, если нажать на соответствующую кнопку слева от него. Простои процесса в миллисекундах отобразятся на гистограмме. Чтобы получить диаграмму всех процессов, надо нажать кнопку All.

Отображения запуска программ можно сохранять неоднократно — это делается автоматически. Имена файлов данных отличаются двумя последними цифрами, первая из них — номер запускаемой программы, последняя — номер процесса.

При загрузке файла данных работы любого процесса программы автоматически загружаются остальные процессы этой программы. При начальном отображении схемы отрезки времени между вызовами функций масштабируются так, чтобы вся картина работы программы целиком отобразилась на экране. В дальнейшем масштаб изображения может быть изменен. Пользователь может выделить участок трассы

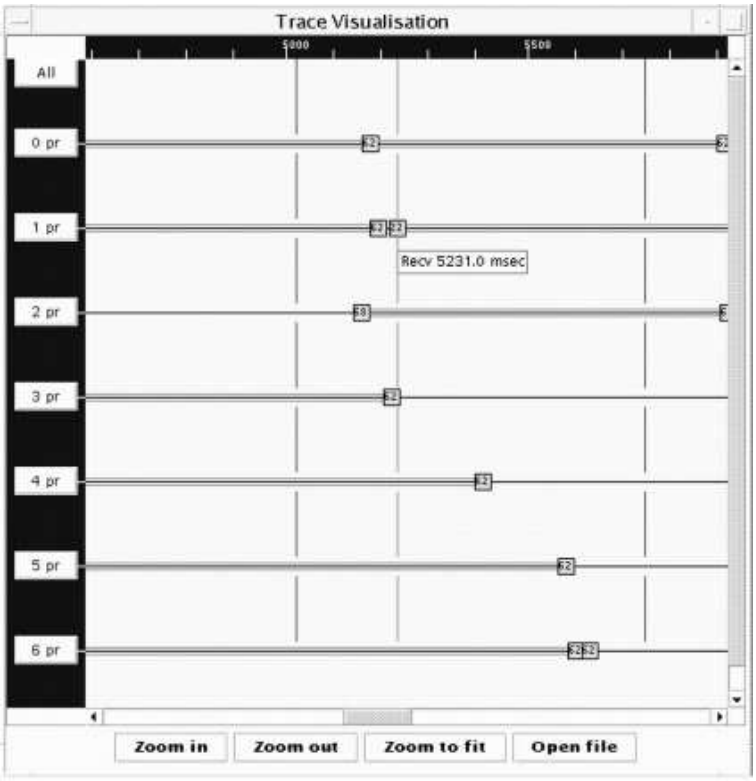


Рис. 11. Выделение участка для просмотра

двумя красным границами (рис. 11) и нажать кнопку Zoom In. Тогда этот кусок растянется на весь экран. При помощи Zoom Out таким же путем можно увеличить размеры просматриваемого отрезка времени. Передвигаться по схеме можно при помощи полосок скроллинга.

**9**. В качестве примера использования среды ParJava и описанной библиотеки интерфейсов и классов рассмотрим параллельный алгоритм, разработанный в рамках курсовой работы на кафедре математической физики факультета ВМиК МГУ. В ней рассматривалось линейное уравнение теплопроводности, для которого была составлена параллельная масштабируемая программа на языке C для компьютера Parsytec GC. Линейное уравнение теплопроводности имеет вид

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

где $u = u(x,t)$ и $x$ — точка $n$-мерного пространства. В двумерной квадратной области $1\times1$ уравнение принимает вид

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

где $u = u(x,y,t)$ и $0 < x < 1$, $0 < y < 1$, $0 < t$. Начальные и граничные условия имеют вид

$$\begin{aligned}
&u|_{t=0} = \varphi(x,y), \quad 0 \leqslant x \leqslant 1, \quad 0 \leqslant y \leqslant 1,\\
&u|_{x=0} = \psi_1(y,t), \quad u|_{x=1} = \psi_2(y,t), \quad 0 \leqslant y \leqslant 1, \quad 0 \leqslant t,\\
&u|_{y=0} = \psi_3(x,t), \quad u|_{y=1} = \psi_4(x,t), \quad 0 \leqslant x \leqslant 1, \quad 0 \leqslant t.
\end{aligned}$$

В качестве функций $\varphi$ и $\psi$ были выбраны $e^{-(x_1+y_1)}$ и $100 * e^{-(x_1+y_1)}$, где $x_1 = (x - 1/2)^2, y_1 = (y - 1/2)^2$ (при соответствующих границе значений переменных).

Решение поставленной задачи выполнялось методом сеток. Двумерная квадратная область разбивалась на квадратики со стороной $h$ (шаг разностной сетки). Значения функций в узлах разностной сетки рассматривались как разностные функции, приближающие коэффициенты уравнения, начальные данные и решение уравнения. В результате исходное уравнение было заменено разностным, решение которого свелось к решению системы линейных алгебраических уравнений с квадратной матрицей $A$ порядка $n$, где $n = 1/h$.

Распараллеливание происходит за счет распределения столбцов матрицы $A$ между разными процессорами. Каждый процессор обсчитывает свою матрицу (часть матрицы $A$) до достижения требуемой точности. Когда на всех процессорах будет достигнута требуемая точность, выполнение параллельной программы завершается.

В случае однородной сети на каждый процессор рассылается одинаковое число столбцов матрицы $A$ (обозначим его через $D$). Поскольку для вычисления конечных разностей необходимы соседние узлы, имеет смысл загрузить на каждый процессор по дополнительному столбцу справа и слева (дублирование данных оправдывается сокращением количества пересылок). Поэтому фактическое число столбцов матрицы $A$ на процессоре равно $M = D + 2$.

В случае неоднородной сети ей сопоставляется однородная сеть (производительность каждого узла неоднородной сети кратна производительности узлов однородной сети), на каждый процессор которой рассылается одинаковое число ($D$) столбцов матрицы $A$. Следовательно, на каждый процессор неоднородной сети будет загружено $M = D * K + 2$ столбцов матрицы $A$.

На рис. 12 приведены фрагменты текстов программ, реализующие этот метод для следующих трех конфигураций вычислительной сети.

1. На каждом узле системы запускалось по одной JavaVM, данные были равномерно распределены по ветвям параллельной программы. Программа была перенесена с C на Java практически без изменений (использовался Java-интерфейс к MPI). На рис. 12 это соответствует не закомментированным строкам текста.

2. Моделировалась однородная оптимальная сеть Java-исполнителей; данные по ветвям параллельной программы были распределены равномерно. Для реализации использовался класс HomogenNet, реализующий интерфейс IHomogenNet. На рис. 12 это соответствует закомментированным “/**” строкам текста.

3. Моделировалась неоднородная сеть JavaVM, данные были распределены пропорционально *относительной производительности* узлов системы. Для реализации использовался класс HeterogenNet, реализующий интерфейс IHeterogenNet. На рис. 12 это соответствует закомментированным “/*” строкам текста.

В первом случае создается вычислительное пространство — Java-сеть, в которой на каждом узле параллельной вычислительной системы с распределенной памятью будет запущена одна JavaVM (рис. 12, строка 5); на полученной сети JavaVM выполняется параллельная программа. Программа реализована из предположения, что параллельная система однородна (класс JavaNet не предоставляет никаких возможностей по управлению ресурсами системы).

Во втором случае на данной системе моделируется однородная сеть JavaVM, которая и является вычислительным пространством для параллельной программы (рис. 12, строка 6). Это позволяет в случае неоднородности вычислительной системы балансировать нагрузку в сети. Становится возможным выполнение программ, разработанных для однородных вычислительных систем на неоднородных вычислительных системах, а также разрабатывать программы для однородных систем на имеющейся неоднородной вычислительной системе (например, на локальной сети).

В третьем случае создается вычислительное пространство — Java-сеть (объект класса HeterogenNet), в которой на каждом физическом узле запущена одна JavaVM; исключены узлы, не соответствующие условию масштабируемости; полученная сеть JavaVM может быть также оптимизирована в соответствии с требованием программы к топологии сети (рис. 12, строка 7). Класс HeterogenNet предоставляет средства для определения и эффективного использования ресурсов параллельной системы.

```
import ru.ispras.ParJava.*;
import java.util.*;
class HeatTransport {
public static void main(String args[ ])
{ JavaNet net = new JavaNet();
/** HomogenNet net = new HomogenNet();
/* HeterogenNet net = new HeterogenNet(args);
int pid = net.rank(); //номер текущего узла
int P = net.size(); //количество узлов сети
int PROCQ = P;
/* int PROCQ = net.netpower();
/* метод netpower() вычисляет сумму относительных производительностей всех узлов сети*/
int gridFrequency = 50; //количество подобластей,
// на которые разбивается
```

```
// обсчитываемая область
    int M, K, D, S;
    /* D – количество столбцов матрицы, размещаемых на узле с
единичной производительностью */
    D=(gridFrequency-2)/PROCQ;
    // M – количество столбцов матрицы, размещаемых на
// текущем узле
    M=D+2;
    /* M=D*relativePerformanceHost(pid)+2;
    K=D*PROCQ+2; //приведенное количество
//подобластей, на которые
//разбивается обсчитываемая область
    S = D*pid;
    /* S = D*net.rectstaken();
    /* метод rectstaken() вычисляет сумму относительных производительностей узлов неоднородной сети с
первого до текущего узла включительно */
    // [ Инициализация остальных переменных. ]
    // [ Вычисление начальных условий.]
    for(m = 0; m < M; m++ )
    { for(k = 0; k < K; k++)
    {
    xx1 = (S + m) * h1 - a/2;
    xx1 = xx1 * xx1;
    yy1 = k * h2 - b/2;
    yy1 = yy1 * yy1;
    // продолжение вычисления начальных условий
    }
    }
    while (allContin != 0) {
    // [Основной цикл параллельной части программы]
    }
    // Обработка полученных результатов
    }
```

Рис. 12. Фрагмент SPMD-программы решения уравнения теплопроводности

В программе вводится переменная PROCQ, которая содержит суммарную относительную производительность всех узлов сети. В случае однородного вычислительного пространства она равна количеству узлов в сети (рис. 12, строка 10). В случае неоднородного вычислительного пространства суммарная относительная производительность всех узлов сети определяется при помощи метода, реализованного в классе HeterogenNet (рис. 12, строка 11).

Количество столбцов матрицы, размещаемых на узле с единичной производительностью, является значением переменной $D$ (рис. 12, строка 16). Поскольку в однородном случае абсолютная производительность всех узлов одинакова, то относительная производительность каждого узла равна 1 и на каждый узел необходимо распределить $D$ столбцов. Но, как было отмечено выше, вычисление конечных разностей связано с использованием соседних столбцов матрицы. Для всех $D$ столбцов, загруженных на рассматриваемый узел (кроме первого и последнего), соседние столбцы находятся на этом же узле. Поэтому, чтобы сделать цикл по столбцам однородным, имеет смысл загрузить на рассматриваемый узел по дополнительному столбцу справа и слева (как в однородном, так и неоднородном случае). Поэтому общее число узлов, загружаемых на узел в однородном случае, равно не $D$, а $M = D + 2$ (рис. 12, строка 18). В неоднородном случае число загружаемых столбцов пропорционально относительной производительности узла (рис. 12, строка 19).

Абсолютный номер первого из $D$ столбцов, размещенных на текущем узле, определяется по номеру узла (pid) в однородном случае (рис. 12, строка 21) и с помощью метода rectstaken() в неоднородном случае (рис. 12, строка 22). В строках 26 – 35 вычисляются начальные условия для текущего узла.

Последующий текст программы не зависит от однородности вычислительного пространства (рис. 12, строки 36 и далее до конца программы). На рис. 13 представлены фрагменты программы, в которых

производится прием и передача правых столбцов матрицы.

```
// Ожидание данных в текущей ветви параллельной программы
// от предыдущей ветви параллельной программы необходимых
// для следующей итерации(правый столбец участка матрицы
// предыдущей ветви параллельной программы)
// wm — буфер приема; M*K — смещение в буфере приема
// начиная с которого помещаются принимаемые данные; K+1 —
// количество (элементов в столбце) принимаемых данных;
// net.DOUBLE — тип принимаемых данных; pid-1 — номер узла
// от которого ожидаются данные
    net.Recv(wm, M*K,K+1,net.DOUBLE,pid-1,200);

// Посылка текущей ветвью параллельной программы правого
// столбца участка матрицы к последующей ветви параллельной
// программы // wm — буфер передачи; M*K+(M-2)*K — смещение в буфере
// передачи начиная с которого отправляются данные; K+1 –
// количество (элементов в столбце) передаваемых данных;
// net.DOUBLE — тип передаваемых данных; pid+1 — номер узла
// которому посылаются данные
    net.Send(wm, M*K+(M-2)*K,K+1,net.DOUBLE,pid+1,200);
```

Рис. 13. Использование функций коммуникационной библиотеки

Во всех трех конфигурациях тестирование проходило при двух выбранных наборах параметров. В первом случае время расчетов в итерации было значительно больше времени пересылок данных; во втором случае эти времена были соизмеримы.

Абсолютное время выполнения программы для каждой из перечисленных конфигураций приведены на рис. 14 и 15 (соответственно для первого и второго случая).

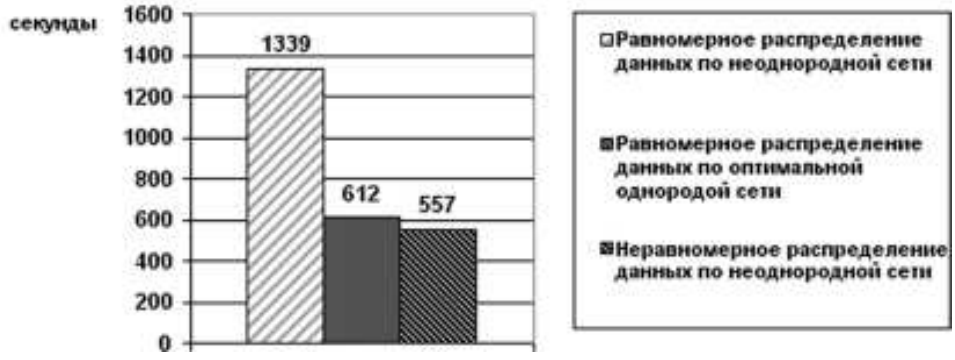


Рис. 14. Абсолютное время выполнения программы в случае, когда время расчетов в итерации было значительно больше времени пересылок данных

Первая серия тестов (сетка – квадрат со стороной 350 узлов, временной шаг 0,001 сек). На расчет одной итерации затрачивалось примерно 300 мсек, на пересылки данных за итерацию уходило не более 5 мсек, что дало общее время расчета 612 сек. Во второй конфигурации на измерение производительностей узлов и моделирование однородной сети ушло примерно 35 сек.

Вторая серия тестов (сетка — квадрат со стороной 50 узлов, временной шаг 0,0001 сек.) На расчет одной итерации затрачивалось примерно 15 мсек, на пересылки данных за итерацию уходило не более 1 мсек, что дало общее время расчета 118 сек. Во второй конфигурации на измерение производительностей узлов и моделирование однородной сети ушло примерно 37 сек.

**10**. В статье описана среда ParJava, которая является расширением среды Java средствами разработки масштабируемых, эффективных, переносимых, объектно-ориентированных параллельных программ как для однородных, так и для неоднородных параллельных вычислительных систем с распределенной памятью. При этом инструментальная вычислительная система, на которой разрабатывается программа,

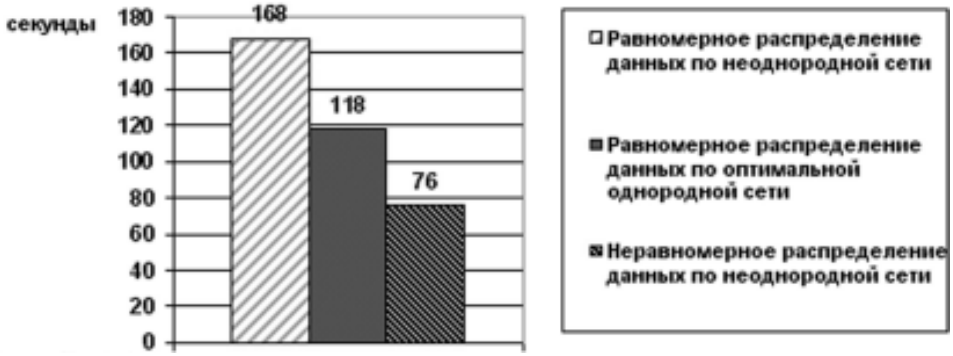


Рис. 15. Абсолютное время выполнения программы в случае, когда время расчетов в итерации соизмеримо со временем пересылок данных

может быть как однородной, так и неоднородной. Среда также позволяет использовать алгоритмы, разработанные для однородных систем, на неоднородных системах без потери масштабируемости, т.е. делает их переносимыми.

Среда ParJava находится в состоянии развития. Разрабатывается полноценный отладчик параллельных программ, использующий в своей работе отладочную библиотеку и средства инструментирования программ и с их помощью поддерживающий процесс разработки масштабируемых параллельных программ. Разрабатываются новые анализаторы.

Расширение языка Java низкоуровневыми возможностями параллельного программирования дает возможность эффективно реализовывать объектные модели параллельного программирования более высокого уровня DPJ [27], DVM [28] и др. Включение этих моделей в среду ParJava позволит существенно облегчить разработку параллельных программ, так как оптимальное распределение данных по узлам параллельных вычислительных систем будет производиться автоматически.

Основным достоинством среды ParJava является то, что параллельная программа, разработанная при помощи этой среды, может выполняться на любой масштабируемой вычислительной системе без каких-либо модификаций или преобразований с сохранением условия масштабируемости. Это снимает многие проблемы по распространению параллельных программ.

Параллельные Java-программы немного уступают по производительности параллельным программам на языке C. Это можно рассматривать как плату за преимущества использования объектно-ориентированной технологии разработки и переносимости разработанных параллельных программ.

Работа выполнена при поддержке РФФИ (проект № 99–01–00206).

СПИСОК ЛИТЕРАТУРЫ

1. *Аветисян А.И., Арапов И.В., Гайсарян С.С., Падарян В.А.* Среда разработки параллельных Java-программ для однородных и неоднородных сетей JavaVM // Труды Всеросс. научн. конф. “Высокопроизводительные вычисления и их приложения”. М.: Изд-во Моск. ун-та, 2000. 46–50.
2. *Buyya R.* (Ed.) High Performance Cluster Computing: Programming and Applications. Vol. 2. Englewood Cliffs: Prentice-Hall, 1999.
3. An Overview of the Intel TFLOPS Supercomputer (http://developer.intel.com/technology/itj/q11998/articles).
4. HITACHI SR8000 Series Super Technical Server (http://www.hitachi.co.jp/Prod/comp/hpc/eng/sr81e.html).
5. ASCI White (http://www.rs6000.ibm.com/hardware/largescale/supercomputers/asciwhite).
6. MPI: Message Passing Interface Standard. Message Passing Interface Forum 2000 (http://www.mcs.anl.gov/mpi).
7. OpenMP: Simple, Portable, Scalable SMP Programming (http://www.openmp.org).
8. 100 Mbps Fast Ethernet (http://wwwhost.ots.utexas.edu/ethernet/100mbps.html).
9. Myrinet Index Page (http://www.myri.com/myrinet/index.html).
10. Linux Online (http://www.linux.org).
11. Учебно-научный центр МГУ по высокопроизводительным вычислениям (http://www.parallel.ru).
12. What is the Scalable Coherent Interface (http://sunrise.scu.edu/WhatIsSCI.html).
13. Hybrid MPI/OpenMP Programming for the SDSC Teraflop System (http://www.npaci.edu/online/v3.14).
14. On-line Resources for Red Hat Linux 6.1 (http://www.redhat.com/support/docs/rhl61.html).
15. The NAS Parallel Benchmarks (http://www.nas.nasa.gov/Software/NPB).

16. Performance of the Cray T3ETM Multiprocessor (http://www.cray.com/products/systems/crayt3e/paper1.html).
17. The Alpha 21264 Microprocessor (http://www.microprocessor.sscc.ru/alpha-21264).
18. Compaq AlphaServer DS10, Compaq AlphaStation DS10 (http://www5.compaq.com/products/quickspecs).
19. Linpack Benchmark — Java Version (http://netlib2.cs.utk.edu/benchmark/linpackjava).
20. SuSE Linux 7.0 (http://www.suse.com/en/index.html).
21. The p4 Parallel Programming System (http://www-fp.mcs.anl.gov/~lusk/p4/index.html).
22. PVM: Parallel Virtual Machine (http://www.epm.ornl.gov/pvm/pvm_home.html).
23. The Industry Standard Java Benchmark. Pendragon Software (http://www.webfayre.com/pendragon/cm3).
24. Java Documentation and Training (http://developer.java.sun.com/developer/infodocs/index.shtml).
25. JAVA 2 SDK. Standard Edition (http://java.sun.com/products/jdk/1.2).
26. LAM/MPI Parallel Computing. LAM Team/UND (http://mpi.nd.edu/lam/index.html).
27. *Гайсарян С.С., Домрачев М.В., Еч Ф.В., Самоваров О.И., Аветисян А.И.* Параллельное программирование в среде Java для систем с распределенной памятью. Объектные модели параллельного выполнения // Труды Ин-та системного программирования РАН. 1999. **1**. 23–34.
28. *Иванников В.П., Гайсарян С.С., Домрачев М.В., Самоваров О.И.* Объектная модель DVM и ее реализация на языке Java // Вопросы кибернетики. Приложения системного программирования. 1998. Вып. 4. 100–118.

Поступила в редакцию
22.03.2001