Библиотека интерфейсов и классов, расширяющих язык Java средствами разработки параллельных программ в модели SPMD*

А.И. Аветисян, В.А. Падарян

Аннотация. В статье описываются языковые средства, а именно набор интерфейсов и реализующих их классов, которые обеспечивают возможность параллельного программирования на однородных и неоднородных сетях JavaVM. Указанные интерфейсы и классы составляют расширение языка Java для параллельного программирования в модели параллелизма по данным (SPMD). Рассматриваемые интерфейсы и реализующие их классы расширяют язык Java низкоуровневыми возможностями реализации параллельных программ на однородных и неоднородных вычислительных системах. В дальнейшем это даст возможность эффективно реализовывать объектные модели параллельного программирования более высокого уровня.

1. В связи с появлением на рынке дешевых и надежных высокопроизводительных коммуникационных систем, позволяющих строить однородные и неоднородные кластеры на базе процессоров массового производства [1], разработка и использование параллельных программ становится особенно актуальной. Возникает потребность в простых и эффективных языковых средствах, поддерживающих разработку параллельных программ. Одним из таких средств является среда ParJava [1], обеспечивающая разработку масштабируемых переносимых параллельных программ в модели SPMD [2] на языке Java.

В среде ParJava параллельные программы реализуются на сети JavaVM (виртуальных Java-машин [3]). Это позволяет использовать параллельные Java-программы на произвольных параллельных вычислительных системах (суперкомпьютерах, однородных и неоднородных кластерах, а также системах, которые могут быть построены из свободных сетевых ресурсов в Internet). Для реализации параллельных программ на сетях JavaVM в среде ParJava используются следующие подходы:

* Работа поддерживалась грантом РФФИ 99-01-00206

- запуск на каждом процессоре распределенной вычислительной системы одной JavaVM, в результате чего получается, вообще говоря, неоднородная сеть Java-исполнителей;
- моделирование однородной сети Java-исполнителей на неоднородной параллельной вычислительной системе путем запуска нескольких JavaVM на части процессоров для балансировки нагрузки.

Моделирование однородной Java-сети может использоваться при разработке параллельных алгоритмов для однородных параллельных вычислительных систем (в частности, суперкомпьютеров и однородных кластеров), для обеспечения переносимости алгоритмов, разработанных для однородных систем, на неоднородные вычислительные системы, а также для исследования свойств параллельных алгоритмов.

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.

Целью данной работы является описание языковых средств, а именно набора интерфейсов и реализующих их классов, которые обеспечивают возможность параллельного программирования на однородных и неоднородных сетях JavaVM. Указанные интерфейсы и классы составляют расширение языка Java для параллельного программирования в модели SPMD. Инструментальные средства среды ParJava рассмотрены в работе [1]. Алгоритмы балансировки нагрузки, позволяющие обеспечить оптимальное распределение данных по узлам сети, в данной работе не рассматриваются. Они будут опубликованы в дальнейшем.

Мы различаем системы параллельного программирования высокого и низкого уровня. В рамках высокоуровневой модели пользователь не занимается передачей сообщений “вручную”, а использует для передачи данных и синхронизации примитивы, описанные в этой модели. Примером системы высокого уровня является JavaParty [4] (хотя, это система распределенного, а не параллельного программирования), а такие средства, как, например, MPI [5] и PVM [6], являются моделями низкого уровня. Каждая система параллельного программирования высокого уровня поддерживается средствами низкого уровня. Так, например, система JavaParty опирается на низкоуровневый интерфейс RMI [7].

Рассматриваемые в статье интерфейсы и реализующие их классы расширяют язык Java низкоуровневыми возможностями реализации параллельных программ на однородных и неоднородных вычислительных системах. В дальнейшем это даст возможность эффективно реализовывать объектные

модели параллельного программирования более высокого уровня, такие как DPJ[8], DVM[9].

В п. 2 описывается иерархия и общая функциональность интерфейсов и классов, расширяющих язык Java низкоуровневыми возможностями реализации параллельных программ.

В п. 3 описываются интерфейсы IJavaNet, IFullGraph, INetProperties. В интерфейсе IJavaNet вводится понятие вычислительного пространства как неструктурированного нумерованного множества Java-исполнителей, которые связаны каждый с каждым. Каждой сети Java-исполнителей можно сопоставить полный взвешенный граф (вес узла – относительная производительность Java-исполнителя, вес ребра – пропускная способность канала связи между соответствующими Java-исполнителями в сети). В интерфейсе INetProperties специфицированы методы сопоставления полного взвешенного графа, отражающего свойства данной сети Java-исполнителей, к этой сети, а также методы, специфицирующие требования программы к структуре вычислительного пространства. Методы, специфицирующие понятие полного взвешенного графа, определены в интерфейсе IFullGraph.

В п. 4 приводится спецификация текущей версии Java-интерфейса к пакету MPI, который используется в качестве прототипа среды коммуникации.

В п. 5 описывается интерфейс IHeterogenNet, в котором специфицируются методы позволяющие моделировать оптимальную неоднородную сеть Java-исполнителей на данной неоднородной параллельной вычислительной системе и реализовывать эффективные низкоуровневые параллельные программы на ней.

В п. 6 описывается интерфейс IHomogenNet, в котором специфицируются методы, позволяющие моделировать однородную сеть Java-исполнитлей на данной неоднородной параллельной вычислительной системе и реализовывать эффективные низкоуровневые параллельные программы на ней.

В п. 7 разбирается пример параллельной Java-программы, демонстрирующий возможности рассмотренных интерфейсов. В заключительном п. 8 содержатся выводы и направления дальнейшего развития описанных интерфейсов.

Библиотека классов, реализующих интерфейсы, описанные в данной работе, а также более подробное описание среды ParJava доступны по Internet на сайте www.ispras.ru/~javap.

2. 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.

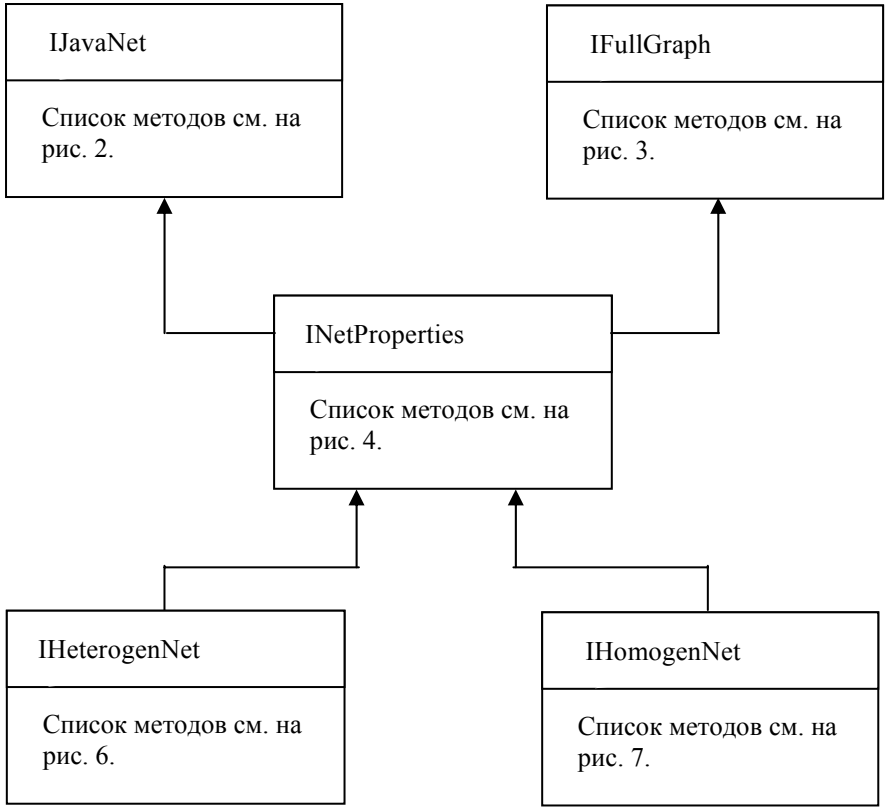

Рис. 1. Иерархия интерфейсов, расширяющих язык Java средствами параллельного программирования в модели SPMD.

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

3. В интерфейсе `IJavaNet` (рис. 2) специфицируются операции (методы) порождения сетей Java-исполнителей и их подсетей, а также операции их преобразования. 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).

```
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 сети
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.

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 сети, 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 сети;
- 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 `complement` 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.

Отметим, что вычислительное пространство (Java-сеть) можно представить в виде полного взвешенного графа, в котором вес ребра соответствует скорости передачи данных между Java-исполнителями, 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.

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

```
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.

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-испонителей, 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.

4. В качестве прототипа среды коммуникации используется библиотека Java-классов, реализующих объектно-ориентированный интерфейс MPI [6], который был реализован для языка C++ [10]. Современное состояние Java-интерфейса к пакету MPI показано на рис. 5. Этот интерфейс может использоваться в Java-программах отдельно от всей системы.

В настоящее время в состав среды ParJava включена прототипная реализация интерфейса JavaMPI. Соответствующие Java-классы реализованы посредством Java Native Interface [3] с помощью вызовов стандартных функций языка C++, реализующих соответствующую функциональность MPI (используется свободно распространяемая версия MPI lam6.3 [10]). В дальнейшем этот интерфейс будет реализован на Java.

5. 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.

```
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.

Для этого интерфейс `INetProperties` расширен интерфейсом `IHeterogenNet` (рис. 6), в котором специфицируются следующие методы:

- метод `relativePerformanceHost` вычисляет производительность узла с запрашиваемым номером;
- метод `myrelativePerformance` вычисляет производительность текущего узла;
- метод `netpower` вычисляет сумму относительных мощностей всех узлов сети;
- метод `rectstaken` вычисляет сумму 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, несоответствующие 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.

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 [11]. 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 производительности узлов неоднородной сети на рассматриваемом классе задач.

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 с временем выполнения ее параллельных частей, этого не достаточно. В этом случае на этапе разработки и отладки программы необходимо оценить затраты времени на выполнение последовательных частей, что даст возможность учесть время выполнения последовательных частей при вычислении 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 [1].

```
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.

6. 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.

7. В качестве примера использования 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, и 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 1×1 уравнение принимает вид:

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

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

Начальные и граничные условия соответственно

$$u\big|_{t=0}=\varphi(x,y),\quad 0\le x\le 1{,}0\le y\le 1$$
$$u\big|_{x=0}=\psi_1(y,t),\ u\big|_{x=1}=\psi_2(y,t),\ 0\le y\le 1{,}0\le t,$$
$$u\big|_{y=0}=\psi_3(x,t),\ u\big|_{y=1}=\psi_4(x,t),\ 0\le x\le 1{,}0\le t.$$

В качестве 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).

Решение поставленной задачи производится методом сеток. Двумерная квадратная область разбивается на квадратики со стороной h (шаг разностной сетки). Значения функций в узлах разностной сетки рассматриваются как

разностные функции, приближающие коэффициенты уравнения, начальные данные и решение уравнения. В результате исходное уравнение превратится в разностное уравнение, которое представляет собой систему линейных алгебраических уравнений с квадратной матрицей A размера $n\times n$, где $n=1/h$.

Распараллеливание происходит за счёт 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. 8 приведены тексты программ, реализующие этот метод для следующих трёх конфигураций вычислительной сети:

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. 8 это соответствует не закомментированным строкам текста.

Моделировалась однородная оптимальная сеть Java-исполнителей; данные по ветвям параллельной программы были распределены равномерно. Использовался класс `HomogenNet`, реализующий интерфейс `IHomogenNet`. На рис. 8 это соответствует закомментированным “/**” строкам текста.

Моделировалась неоднородная сеть JavaVM, данные были распределены пропорционально *относительной производительности* узлов системы. Использовался класс `HeterogenNet`, реализующий интерфейс `IHeterogenNet`. На рис. 8 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. 8, 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. 8, 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 системы запущена одна JavaVM, исключены узлы, не соответствующие условию масштабируемости (рис. 8, строка 7). Класс `HeterogenNet` предоставляет средства для определения и эффективного использования ресурсов параллельной системы.

```
1.  import ru.ispras.ParJava.*;
2.  import java.util.*;
3.  class HeatTransport {
4.  public static void main(String args[])
5.  {     JavaNet      net = new JavaNet();
6.    /** HomogenNet   net = new HomogenNet();
7.     /* HeterogenNet net = new HeterogenNet(args);
8.        int pid =    net.rank();   //номер текущего узла
9.        int P =      net.size();   //количество узлов сети
10.    int PROCQ =  P;
11. /* int PROCQ =  net.netpower();
    /* метод netpower() вычисляет сумму относительных
       производительностей всех узлов сети*/
13.    int gridFrequency = 50;    //количество подобластей,
                                  // на которые разбивается
                                  // обсчитываемая область
14.    int M, K, D, S;
15. /* D – количество столбцов матрицы, размещаемых на узле с
       единичной производительностью */
16.    D=(gridFrequency-2)/PROCQ;
17. // M – количество столбцов матрицы, размещаемых на
     //     текущем узле
18.    M=D+2;
19. /* M=D*relativePerformanceHost(pid)+2;
20.    K=D*PROCQ+2;       //приведенное количество
                          //подобластей, на которые
                          //разбивается обсчитываемая область
21.    S = D*pid;
22. /* S = D*net.rectstaken();
```



```
    /* метод rectstaken() вычисляет сумму относительных
       производительностей узлов неоднородной сети с первого до
       текущего узла включительно */
24. //      [ Инициализация остальных переменных. ]
25. //      [ Вычисление начальных условий.]
26. for(m = 0; m < M; m++ )
27. { for(k = 0; k < K; k++)
28.   {
29.      xx1 = (S + m) * h1 - a/2;
30.      xx1 = xx1 * xx1;
31.      yy1 = k * h2 - b/2;
32.      yy1 = yy1 * yy1;
33.      // продолжение вычисления начальных условий
34.   }
35. }
36. while (allContin != 0) {
37. //  [Основной цикл параллельной части программы]
38. }
39. // Обработка полученных результатов
40. }
```

рис. 8. 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. 8, строка 10). В случае неоднородного вычислительного пространства суммарная относительная производительность всех узлов сети определяется при помощи метода, реализованного в классе HeterogenNet (рис. 8, строка 11).

Количество столбцов матрицы, размещаемых на узле с единичной производительностью, является значением переменной *D* (рис. 8, 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. 8, строка 18). В неоднородном случае число загружаемых столбцов пропорционально относительной производительности узла (рис. 8, строка 19).

Абсолютный номер первого из *D* столбцов, размещенных на текущем узле, определяется по номеру узла (`pid`) в однородном случае (рис. 8, строка 21) и с

помощью метода `rectstaken()` в неоднородном случае (рис. 8, строка 22). В строках 26-35 вычисляются начальные условия для текущего узла.

Последующий текст программы не зависит от однородности вычислительного пространства (рис. 8, строки 36 и далее до конца программы). На рис. 9 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. 9. Использование функций коммуникационной библиотеки.

8. В статье описано языковое расширение среды Java библиотеками интерфейсов и классов, позволяющими реализовывать масштабируемые, эффективные, переносимые, объектно-ориентированные параллельные программы, как для однородных, так и для неоднородных параллельных вычислительных систем с распределенной памятью. Это расширение является частью среды ParJava [1].

Отметим, что разработанные библиотеки интерфейсов и классов являются низкоуровневым расширением языка Java для разработки параллельных программ. Эти интерфейсы и классы позволяют эффективно реализовывать объектные модели параллельного программирования более высокого уровня DPJ, DVM и др., что существенно облегчит разработку параллельных программ, так как оптимальное распределение данных по узлам параллельных вычислительных систем будет производиться автоматически.

Основным преимуществом использования описанных интерфейсов и классов является то, что параллельная программа, разработанная при помощи них,

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 уступают по производительности параллельным программам на языке C. Однако это различие не очень существенно (например, как показали расчеты на компьютере для теста Linpack [12] производительность JavaVM из среды JDK 1.2 [3] в 1,5 раза меньше C/C++). Его можно рассматривать, как плату за преимущества использования объектно-ориентированной технологии разработки и абсолютной переносимости разработанных параллельных программ.

Литература:

1. А.И. Аветисян, И.В. Арапов, С.С. Гайсарян, В.А. Падарян. “Среда ParJava для разработки SPMD-программ для однородных и неоднородных сетей JavaVM.” Труды Института Системного Программирования Российской Академии Наук. Том 2, 2000.
2. Rajkumar Buyya (ed.) "High Performance Cluster Computing": Programming and Applications. Vol. 2. Prentice Hall PTR, New Jersey, 1999.
3. JAVA 2 SDK, Standard Edition http://java.sun.com/products/jdk/1.2/
4. M. Philippsen, M. Zender, “JavaParty – Transparent Remote Objects in Java”, In Proceedings of ACM PPoPP Workshop on Java for Science and Engineering Computation, 1997
5. MPI: Message Passing Interface Standard, Message Passing Interface Forum, 2000; http://www.mcs.anl.gov/mpi/index.html
6. PVM: Parallel Virtual Machine, home page. http://www.epm.ornl.gov/pvm/pvm_home.html
7. Java Remote Method Invocation (RMI) http://java.sun.com/products/jdk/1.2/docs/guide/rmi/index.html
8. С.С. Гайсарян, М.В. Домрачёв, В.Ф. Еч, О.И.Самоваров, А.И. Аветисян. “Параллельное программирование в среде Java для систем с распределенной памятью. Объектные модели параллельного выполнения.” Труды Института Системного Программирования Российской Академии Наук. Том 1. 1999.
9. В.П. Иванников, С.С. Гайсарян, М.В. Домрачев, О.И. Самоваров. “Объектная модель DVM и ее реализация на языке Java.” Вопросы Кибернетики. Приложения системного программирования. Выпуск 4. 1998.
10. LAM/MPI Parallel Computing. LAM Team/UND. 1996-2000. http://mpi.nd.edu/lam/index.html
11. The industry standard Java benchmark. Pendragon Software. http://www.webfayre.com/pendragon/cm3/
12. Linpack Benchmark -- Java Version http://netlib2.cs.utk.edu/benchmark/linpackjava /