Среда ParJava для разработки SPMD-программ для однородных и неоднородных сетей JavaVM*

А.И. Аветисян, И.В. Арапов, С.С. Гайсарян, В.А. Падарян

Аннотация. В статье дается общее описание среды 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, 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.

1. Мы изучаем возможности разработки и реализации параллельных программ для многопроцессорных вычислительных систем с распределенной памятью в модели SPMD [1]. 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,

* Работа поддерживалась грантом РФФИ 99-01-00206

- иногда – 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 [2], SR8000 Series фирмы Hitachi [3] 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 [4]. 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 [5] и OpenMP [6] – международные стандарты интерфейсов передачи сообщений соответственно на распределенной и общей памяти.

В настоящее время в связи с появлением стандартного высокопроизводительного коммуникационного оборудования (Fast Ethernet [7], Myrinet [8] и др.) широкое распространение получают вычислительные кластеры с распределенной памятью. Их основное достоинство – их аппаратное обеспечение состоит из стандартных процессоров и коммуникационного оборудования массового производства, что позволяет использовать программное обеспечение, разработанное для локальных сетей (например, ОС Linux [9], 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. Сведения о вычислительных кластерах приведены в таблице 1, заимствованной из [10].

Таблица 1. Вычислительные кластеры и их операционные системы.

Организация/проект	N	Конфигурация узла	CPU	Сеть	ОС
LANL/Avalon	140	Alpha 21164/533MHz, 256 MB RAM, 3.2 GB HDD	140	Fast Ethernet + Gigabit Ethernet (в центре).	Linux
НИВЦ МГУ	12	Dual Pentium III/500MHz, 256 MB RAM, 3.2 GB HDD	24	SCI+ Fast Ethernet.	Red Hat Linux 6.1
ИВВиБД, "Паритет"	4	2 x Pentium II/450MHz, 512MB RAM, 10GB HDD	8	Myrinet + Fast Ethernet	Linux
ИСОИ РАН	4	2 x Pentium II/400MHz, 512MB RAM, 9 GB HDD	8	Fast Ethernet, коммутатор Cisco Catalyst 2912	Linux
OSC	32	Сервер SGI 1400L: 4 x Pentium III Xeon/500	128	N/A	Linux
CTC "AC3 Velocity"	64	DELL PowerEdge – 4 x Pentium III Xeon/500MHz, 4GB RAM, 54GB HDD.	256	Clan	Windows NT/2000
Indiana Uni.	32	Compaq ProLiant – 2xPentium II/400MHz	64	Fast Ethernet и Gigabit Ethernet (параллельно)	Windows NT + Linux
NCSA/NT Supercluster	64	2-процессорные раб. станции HP Kayak XU, процессоры Pentium III/450MHz и Pentium II/300MHz	128	Myrinet	Windows NT
AHPCC/ Roadrunner	64	2xPentium II/450MHz, 512MB RAM, 6.4GB HDD	128	Myrinet	Linux
Paderborn Uni./PSC1	32	SNI Celsius 2000E – 2xPentium II/300MHz	64	SCI(Dolphin)	Linux
Paderborn Uni./PSC	96	SNI Primergy – 2 x Pentium II/450MHz, 512 MB RAM, 4 GB HDD.	192	SCI (Dolphin)	Solaris 2.6
LANL/ Pathforward	128	Alpha EV6/500MHz, 1GB RAM	256	Onet (Quadrics)	Tru64 UNIX
Sandia/Cplant	400	Digital Personal Workstation 500a – Alpha/500MHz, 192 MB RAM, без диска	400	Myrinet(для каждого подкластера по 16 узлов)	Linux

Linux-кластер, состоящий из 12 двухпроцессорных серверов Eximer с процессорами Pentium III/500 Мгц, имеет пиковую производительность 12 GFLOPS. Он введен в эксплуатацию в НИВЦ МГУ. Вычислительные узлы объединяются в двумерную решетку с помощью высокоскоростной сети SCI [11], 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 [12]. Кластер работает под управлением операционной системы Linux Red Hat 6.1 [13]. Производительность на тесте при решении системы линейных уравнений с матрицей 16000×16000 составляет 5.7 GFLOPS. На тестах NPB 2.3 [14] производительность сравнима с показателями суперкомпьютеров семейства CRAY T3E [15].

Кластер AliCE (Alpha Linux Cluster Engine) компании Compaq состоит из 128 процессоров Alpha 21264 [16], установленных в станциях DS10 [17], которые объединены в сеть Myrinet SAN (пропускная способность 1.28 Гбит/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 [18]). Распределенная оперативная память – 16 Гбайт, дисковая – 1 Тбайт. Кластер работает под управлением операционной системы Suse Linux 6.3 [19].

В настоящее время используются также "вычислительные фермы" (compute farm) – кластеры, все узлы которых размещены в одном корпусе. В качестве примеров вычислительных ферм можно упомянуть:

- ферму 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;
- ферму на базе плат TPE3 [20]. Плата представляет собой вычислительный узел с процессором G4(400 Мгц), содержащий до 512 Mbytes оперативной памяти. Система включает до 16 таких узлов с общей пиковой производительностью до 45 GFLOPS. Платы расположены на общей PCI шине и, кроме того, объединены в сеть Myrinet.

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.

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.

Отметим, что программа, переносимая на однородных системах, может быть не переносима на НВС даже в случае, если у нее однородны вычислительная и коммуникационная среды (т.е. различны только производительности узлов).

```
MPI.Init(args);
int size=MPI.COMM_WORLD.Size();
int rank=MPI.COMM_WORLD.Rank();
nt local_size=S/size+(S%size)/(rank+1);
Request req;
if (rank==0)
{
        matrx=new int[S*S];
        for (int i=0,shift=0;i<size;i++)
```

```
        {
                MPI.COMM_WORLD.Isend(matrx,S*shift,S*(S/size
                        +(S%size)/(i+1)),MPI.INT,i,99);
                shift+=S/size+(S%size)/(i+1);
        }
}
wm=new int[local_size*S];
req=MPI.COMM_WORLD.Irecv(wm,0,local_size*S,MPI.INT,0,99);
req.Wait();
res=new int[local_size];
for (int i=0;i<local_size;i++)
        for (int j=0;j<S;j++)
                res[i]+=wm[S*i+j];
MPI.COMM_WORLD.Isend(res,0,local_size,MPI.INT,0,99);
if (rank==0)
{
        R=new int[S];
        for (int i=0,shift=0;i<size;i++)
        {
                req=MPI.COMM_WORLD.Irecv(R,shift,S/size+(S%
                        size)/(i+1),MPI.INT,i,99);
                req.Wait();
                shift+=S/size+(S%size)/(i+1);
        }
}
MPI.Finalize();
```

Рис. 1. Параллельная программа, не переносимая с однородных систем на НВС.

Рассмотрим параллельную программу (рис.1), которая может выполняться как на однородной системе, так и на НВС, имеющих однородную вычислительную среду. Но она не является переносимой на НВС в смысле нашего определения, т.к. ее перенос на НВС связан с потерей масштабируемости

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 дает возможность пользователям, ранее не имевшим доступа к параллельным вычислительным ресурсам, широко использовать параллельные высокопроизводительные вычислительные комплексы, доступные через Интернет.

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 зарекомендавшие 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.

Сравнение производительностей на некоторых бенчмарках для различных реализаций JavaVM и оптимизированного кода C приведено в таблице 2.

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

Таблица 2. Сравнение производительностей 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

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. 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. 2 показана часть главного окна системы во время редактирования параллельной программы, разрабатываемой в среде ParJava.

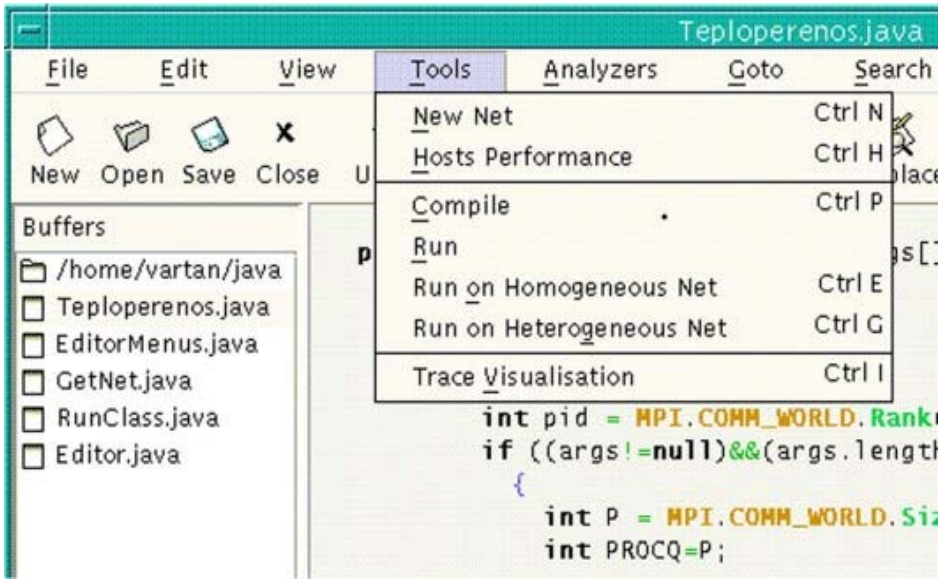

Рис. 2. Часть главного окна системы ParJava во время редактирования параллельной программы.

В тексте редактируемой программы все вызовы параллельных функций выделяются отдельным цветом (на рис. 2 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. Некоторые из этих средств содержатся в меню Tools (на рис. 2 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.
- 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 выдаче.
- 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 TraceVizualization, пункта меню Tools. Пример такой визуализации показан на рис. 3.

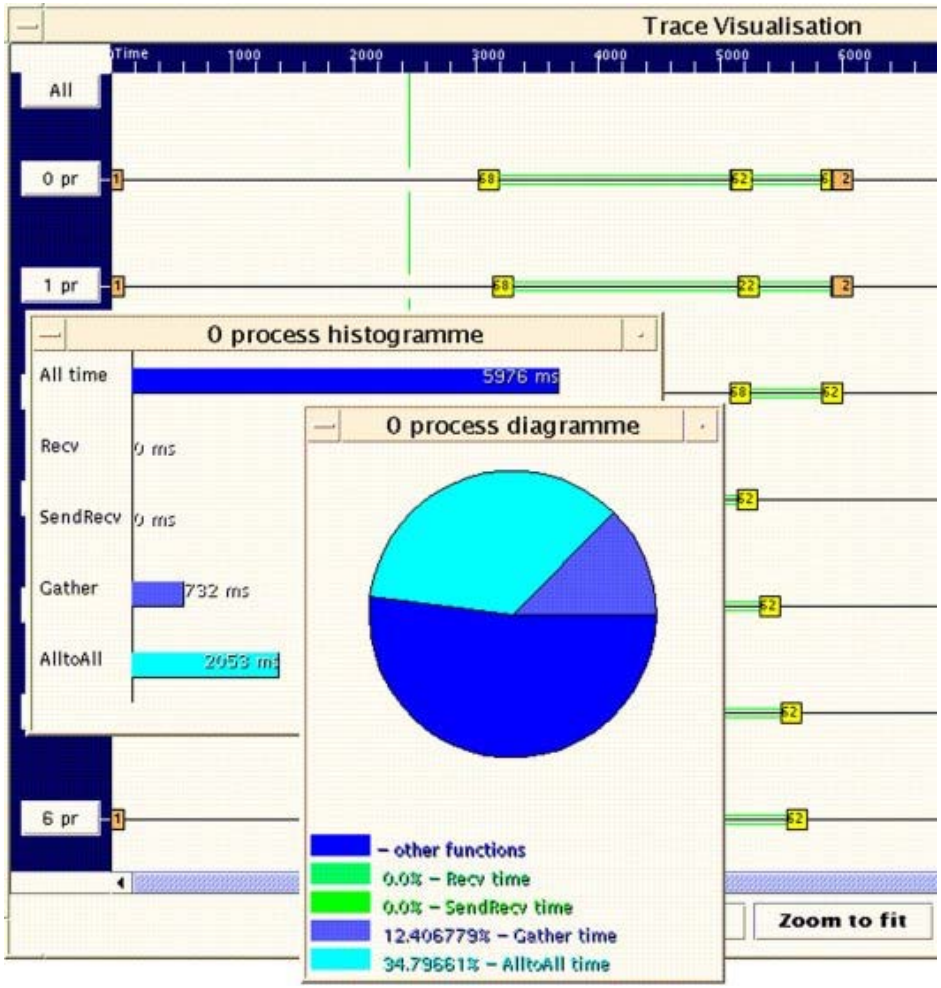

Рис. 3. Пример визуализации процессов

На приведенном рисунке можно увидеть, как программа TraceVisualization демонстрирует работу семи процессов. Здесь каждая линия – это отдельный процесс параллельной программы. Каждый квадратик на линии – вызов одной

из функций MPI. Внизу показана временная линейка. При помощи нее можно определить (в миллисекундах), когда произошел вызов той или иной функции.

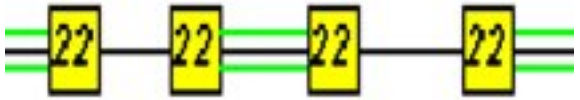

Рис. 4. Ожидание данных.

Функции на схеме, ради экономии места и удобства представления отображаются не названиями, а 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) данной функцией. На рис. 4 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.

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 двумя красным

границами (рис. 5.) 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.

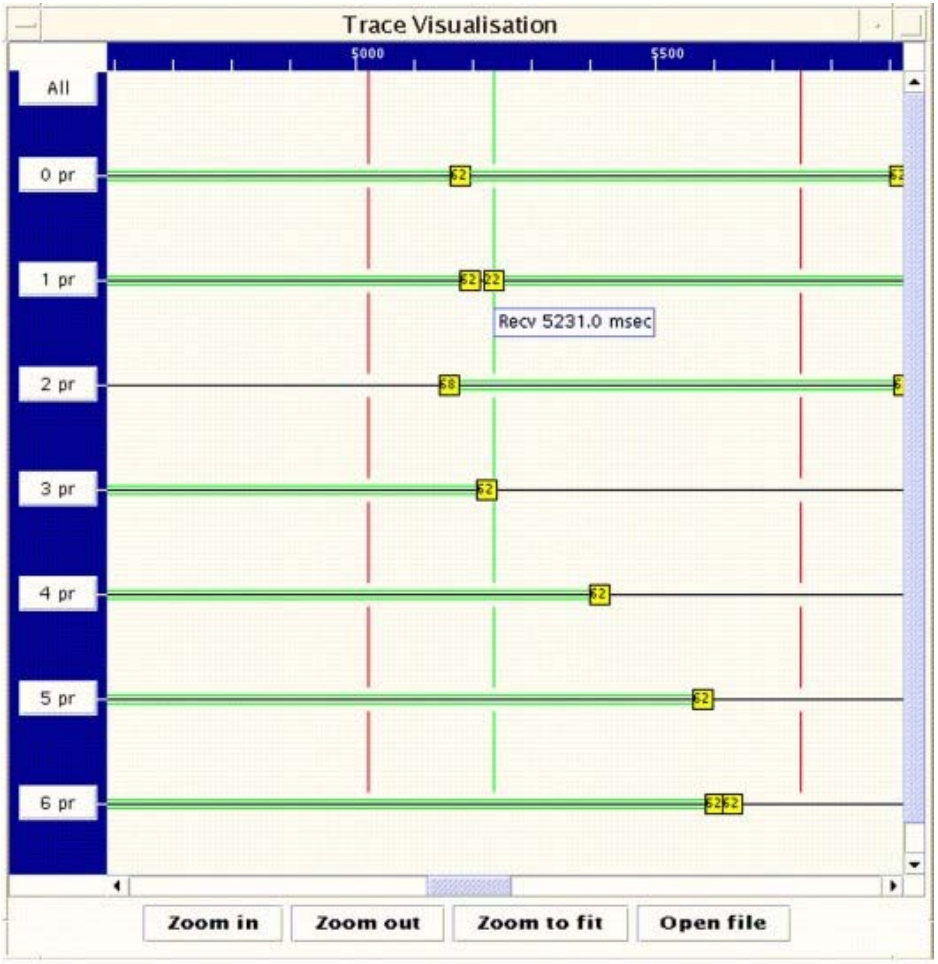

Рис. 5. Выделение участка для просмотра

5. В качестве примера использования среды ParJava рассмотрен параллельный алгоритм, разработанный в рамках курсовой работы на кафедре математической физики 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).

Программа была перенесена с C на Java практически без изменений. Распараллеливание происходит за счёт распределения узлов сетки между разными процессорами – она «нарезается» на прямоугольники вдоль оси *y*. Каждый процессор обсчитывает свою матрицу (часть общей матрицы) до достижения требуемой точности. Когда на всех процессорах будет достигнута требуемая точность, выполнение параллельной программы завершается. Тестирование проходило при двух выбранных наборах параметров. В первом случае время расчётов в итерации было значительно больше времени пересылок данных; во втором случае эти времена были соизмеримы. В каждом случае программа работала в трёх конфигурациях:

1. На каждом узле системы запускалось по одной JavaVM, данные были равномерно распределены по ветвям параллельной программы.
2. Моделировалась однородная оптимальная сеть Java-исполнителей; данные по ветвям параллельной программы были распределены равномерно.
3. Моделировалась неоднородная сеть JavaVM, данные были распределены пропорционально *относительной производительности* узлов системы.

Абсолютное время выполнения программы для каждой из перечисленных конфигураций приведены на рис. 6 и 7 (соответственно для первого и второго случая).

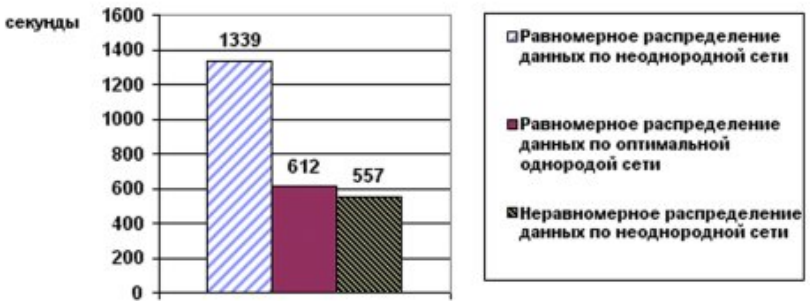

Рис. 6. Абсолютное время выполнения программы в случае, когда время расчётов в итерации было значительно больше времени пересылок данных

Первая серия тестов (сетка – квадрат со стороной 350 узлов, временной шаг 0,001 сек). На расчёт одной итерации затрачивалось примерно 300 мсек., на пересылки данных за итерацию уходило не более 5 мсек., что дало общее время расчета 612 секунд. Во второй конфигурации на измерение производительностей узлов и моделирование однородной сети потребовалось 35 сек. (5,72%).

Вторая серия тестов (сетка – квадрат со стороной 50 узлов, временной шаг 0,0001 сек.) На расчёт одной итерации затрачивалось примерно 15 мсек., на пересылки данных за итерацию уходило не более 1 мсек., что дало общее время расчета 118 секунд. Во второй конфигурации на измерение производительностей узлов и моделирование однородной сети потребовалось 37 сек. (31%).

6. 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, 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.

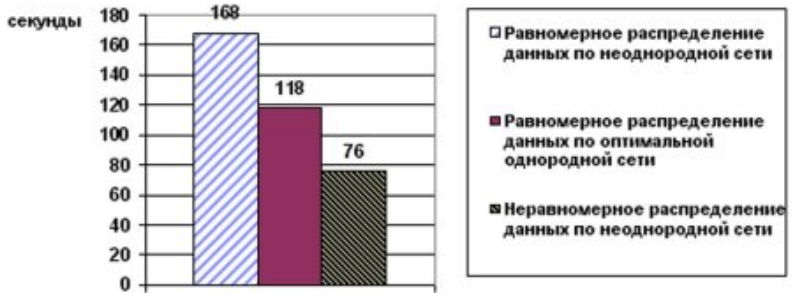

Рис. 7. Абсолютное время выполнения программы в случае, когда время расчётов в итерации соизмеримо со временем пересылок данных.

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[26], DVM[27] 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.

Литература:

1. Rajkumar Buyya (ed.) "High Performance Cluster Computing": Programming and Applications. Vol. 2. Prentice Hall PTR, New Jersey, 1999.
2. An Overview of the Intel TFLOPS Supercomputer http://developer.intel.com/technology/itj/q11998/articles/art_1d.htm

3. HITACHI SR8000 Series Super Technical Server
http://www.hitachi.co.jp/Prod/comp/hpc/eng/sr81e.html
4. ASCI White
http://www.rs6000.ibm.com/hardware/largescale/supercomputers/asciwhite/
5. MPI: Message Passing Interface Standard, Message Passing Interface Forum, 2000;
http://www.mcs.anl.gov/mpi/index.html
6. OpenMP: Simple, Portable, Scalable SMP Programming
http://www.openmp.org/
7. 100 Mbps Fast Ethernet
http://wwwhost.ots.utexas.edu/ethernet/100mbps.html
8. Myrinet Index Page
http://www.myri.com/myrinet/index.html
9. Linux Online
http://www.linux.org/
10. Сервер НИВЦ МГУ
http://www.parallel.ru/
11. What is the Scalable Coherent Interface
http://sunrise.scu.edu/WhatIsSCI.html
12. Hybrid MPI/OpenMP programming for the SDSC teraflop system
http://www.npaci.edu/online/v3.14/SCAN.html
13. On-line Resources for Red Hat Linux 6.1
http://www.redhat.com/support/docs/rhl61.html
14. The NAS Parallel Benchmarks
http://www.nas.nasa.gov/Software/NPB/
15. Performance of the Cray T3ETM Multiprocessor
http://www.cray.com/products/systems/crayt3e/paper1.html
16. The Alpha 21264 microprocessor
http://www.microprocessor.sscc.ru/alpha-21264/
17. Compaq AlphaServer DS10, Compaq AlphaStation DS10
http://www5.compaq.com/products/quickspecs/10398_na/10398_na.html
18. Linpack Benchmark -- Java Version
http://netlib2.cs.utk.edu/benchmark/linpackjava/
19. SuSE Linux 7.0
http://www.suse.com/en/index.html
20. PowerPC 750/7400 PCI Module
http://www.transtech-dsp.com/powerpc/tpe3.htm
21. The p4 parallel programming system. 2000.
http://www-fp.mcs.anl.gov/~lusk/p4/index.html
22. PVM: Parallel Virtual Machine, home page.
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 & training. 2000.
http://developer.java.sun.com/developer/infodocs/index.shtml
25. JAVA 2 SDK, Standard Edition
http://java.sun.com/products/jdk/1.2/

26. 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.
27. 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