Модел. и анализ информ. систем. Т. **21**, № **2** (2014) 15–25

УДК 519.681.5: 519.682

О поддержке рекурсивно-параллельного программирования в .NET Framework

Васильчиков В. В.

Ярославский государственный университет им. П. Г. Демидова
150000 Россия, г. Ярославль, ул. Советская, 14

e-mail: vasilch@uniyar.ac.ru

получена 26 января 2014

Ключевые слова: параллельные вычисления, рекурсия, .NET

Рассматриваются программные компоненты для поддержки рекурсивно-параллельного программирования в .NET Framework. Они представляют собой динамически подключаемые библиотеки, предоставляющие необходимую функциональность для разработки и отладки приложений, предназначенных для параллельного выполнения на локальной сети. Библиотечные классы коммуникационного модуля обеспечивают удобные средства для установления соединения компьютеров в сети по принципу "каждый с каждым" и надежную асинхронную передачу сериализуемых объектов. Классы модуля поддержки рекурсивно-параллельного программирования обеспечивают возможность оформления параллельных ветвей вычислений как мигрирующих процессов, их распределение по сети, передачу параметров и возврат результатов с необходимой синхронизацией, динамическое перераспределение работы для балансировки загрузки, а также работу с общими данными. Приводятся несколько вариантов рекурсивного распараллеливания решения задачи о поиске максимальной клики в неориентированном графе и результаты тестирования рассматриваемых компонентов на примере этой задачи.

Введение

Основные принципы организации рекурсивно-параллельных (РП) вычислений изложены в [1], там же описаны основные алгоритмы и механизмы поддержки этого стиля программирования, а также первая версия соответствующих программных средств его поддержки. В качестве основного языка программирования в этой версии использовался модифицированный язык C. Впоследствии было разработано еще несколько версий библиотек с целью добавления возможности создавать РП-приложения на C++ для выполнения в Win32 [2, 3]. Все эти библиотеки требовали создания специальных проектов и соблюдения при разработке ряда не всегда очевидных правил.

С появлением .NET Framework появилась возможность упростить процесс создания конечного РП-приложения, причем предоставить пользователю возможность использовать любой язык программирования, поддерживаемый этой средой исполнения, и любой тип пользовательского интерфейса (консольный, WPF или основанный на Windows Forms). К минимуму сводятся и ограничения на код клиентского приложения, они вызваны только особенностями работы с общими данными, однако этот момент имеет место, по-видимому, для любых параллельных программ.

Для разработчика собственно библиотек поддержки гибкость предоставляемых средств несколько уменьшилась по сравнению с Win32: меньше возможностей ручного управления потоками, отсутствие поддержки волокон. Однако классы .NET Framework позволяют обеспечивать надежность и устойчивость кода в большей степени, а это представляется автору более важным моментом при разработке таких сложных программных средств и в известной степени компенсирует недостаток гибкости управления.

Автором были разработаны два основных компонента [4, 5] для организации рекурсивно-параллельных вычислений в локальной сети с поддержкой .NET Framework версии 4.0 или более новой, а именно: коммуникационный модуль и собственно библиотека поддержки РП-программирования. Каждый из них представляет собой динамически подключаемую библиотеку, которая может использоваться в программах для .NET Framework версии 4.0 с любым типом интерфейса. Пользователю они предоставляются в виде DLL-файлов, которые он должен включить в свой проект. Предполагается, что для создания и отладки программы используется среда Visual Studio 2010 или более новая. Разумеется, в этом случае программист может воспользоваться всеми возможностями среды разработки, включая IntelliSense. Кроме того, для обоих компонентов дополнительно предоставляется развитый файл контекстной справки в формате CHM.

1. Коммуникационный модуль

Модуль предоставляется в виде библиотечного файла CommModule.dll (файл справки CommModule.chm). Основными функциональными возможностями компонента являются следующие.

При запуске пользователю (независимо от типа основного интерфейса приложения) предоставляется удобный WPF-интерфейс для связи по протоколу TCP приложений, запущенных на разных компьютерах локальной сети, по принципу "каждый с каждым".

При установлении связи не требуется вводить ни имена, ни IP-адреса — на первом этапе соединения используется широковещание по протоколу UDP. Поскольку упомянутый протокол не гарантирует доставку сообщений, предусмотрены средства для посылки дополнительных сообщений, побуждающих участников процесса соединения выполнить необходимые действия. После завершения этапа первоначального установления связи каждый компьютер сети связан с каждым по протоколу TCP, каждому присвоен номер для его идентификации, и управление передается приложению-клиенту.

Для удобства отладки на одном компьютере предусмотрена возможность запускать несколько (в настоящей версии до четырех — автор полагает, что этого доста-

точно) экземпляров приложения, они будут связываться друг с другом так же, как если бы работали на разных рабочих станциях. Клиент может даже автоматически удобно расположить на экране окна разных копий приложения, реализующих выбранный разработчиком пользовательский интерфейс (в том числе и консольные окна).

Классы коммуникационного модуля предоставляют разработчику удобные методы для надежной передачи на удаленный компьютер любого сериализуемого объекта и обеспечивают восстановление там его состояния. Они также предлагают средства для пересылки по сети пользовательских сообщений и организации их обработки по мере приема. При этом все необходимые действия по синхронизации, преобразованию сложных объектов в поток байтов и восстановлению их на удаленном компьютере осуществляются автоматически.

Область применения предложенного коммуникационного модуля не ограничивается его использованием со стороны библиотеки поддержки рекурсивно-параллельного стиля программирования, хотя изначально он предназначался именно для этого. Компонент может быть полезен в любых сетевых программах для .NET Framework, поскольку позволяет очень легко установить полносвязное соединение и передавать объекты любого уровня сложности, не заботясь о множестве технических деталей.

2. Библиотека поддержки рекурсивно-параллельного программирования

Компонент предоставляется в виде библиотечного файла RPM_ParLib.dll (файл справки RPM_ParLib.chm). В своей работе он использует коммуникационный модуль CommModule.dll, который должен находиться в той же директории. Ниже описываются основные функциональные возможности библиотеки. Эти возможности, а также механизмы их реализации практически не отличаются от описанных в [1], однако в предлагаемой версии библиотеки в качестве языка программирования выступает не C, а C# (точнее, любой язык программирования для .NET Framework). Поэтому способ доступа к предлагаемым методам основан на принципах объектно-ориентированного программирования (ООП) и типичен для любых ООП-языков.

При запуске приложения инициируется установление сетевого соединения приложений, запущенных на разных компьютерах локальной сети, по принципу "каждый с каждым". Для сетевого взаимодействия библиотека использует упомянутый выше коммуникационный модуль.

После установления начального соединения пользователь может выбрать "главный" процессорный модуль (ПМ), в качестве которого может выступать один из запущенных экземпляров приложения (неважно, на разных рабочих станциях они запущены или на одной). "Главным" он называется только потому, что первым начинает вычисления, управление всей последующей работой носит децентрализованный характер.

Весь процесс вычислений должен быть оформлен как рекурсивный метод либо несколько методов, допускающих рекурсивный вызов. Поскольку в C# нет глобальных методов, в качестве "обертки" для своего рекурсивного метода разработчик

должен объявить класс-наследник библиотечного класса $ParMethodBase$ и разместить необходимый код в переписанном (*override*) методе $ParMethod()$. Последний принимает в качестве единственного параметра экземпляр класса-наследника библиотечного класса $ParamBase$, который служит и для передачи рекурсивному методу необходимой информации, и для возврата результатов. Собственно, все требования к оформлению кода сводятся к использованию упомянутых двух библиотечных классов. Вся предлагаемая разработчику функциональность реализуется через вызов их методов.

Библиотека предоставляет удобные методы для разбиения процесса вычислений на параллельные ветви с использованием рекурсии, автоматическое начальное распределение работы по системе, динамическую балансировку загрузки в процессе вычислений, возврат результатов с удаленных модулей, простую (по крайней мере, для разработчика прикладной программы) синхронизацию. Разумеется, при написании кода необходимо понимать основные особенности поведения параллельной программы, в частности, важность правильной синхронизации отдельных ветвей и правильной организации работы с общими данными. Для более тонкой настройки механизмов распределения работы предусмотрены специальные методы и свойства, однако, на начальных этапах работы вполне подойдут и их значения по умолчанию.

Так же, как и в [1], в рассматриваемой версии библиотеки предусмотрены два класса памяти для работы с общими данными: данные, копия которых присутствует на каждом ПМ, а также общие данные, которые распределены по отдельным модулям системы, однако могут потребоваться любому ПМ. Эти данные размещаются во время работы программы динамически, предусмотрено несколько методов для доступа к ним как по чтению, так и по записи с различной семантикой запроса. Однако для эффективной работы параллельной программы следует тщательно продумать способ размещения данных и способ доступа к ним, поскольку передача данных по сети сопряжена со значительными временными задержками.

3. Основные преимущества предлагаемых компонентов

Для использования предлагаемых программных средств не требуется создание проекта какого-либо специального типа, годится практически любой шаблон Visual Studio для создания Windows-приложения .NET Framework. Прикладной программист только должен оформить основную рекурсивную процедуру и ее параметры описанным выше образом, то есть разместив их в классах-наследниках соответствующих библиотечных классов.

Разработчик конечного приложения не должен заботиться о способе и механизме распределения работы, при создании исходного кода даже не требуется знать количество и характеристики компьютеров, образующих сеть для вычислений, он инвариантен к этим характеристикам. От программиста требуется только породить достаточное количество активаций параллельной процедуры — об остальном позаботится библиотека. При этом не исключается возможность ручного разбиения и распределения работы по системе. Правда, программист должен все-таки правиль-

но понимать, как организовать это разбиение и что такое "достаточное количество". Этот вопрос, в частности, рассматривается ниже.

Вопросы синхронизации вычислений и использования общих данных при разработке параллельных приложений очень непросты (в особенности для начинающих), а поиск и устранение ошибок, вызванных неправильной синхронизацией, весьма нетривиален даже с использованием современных средств отладки. Парадигма РП-программирования предлагает очень простую модель синхронизации, понятную даже новичку. При этом механизмы обеспечения правильной работы этой "простой модели" вовсе не так просты, но это уже не забота прикладного программиста.

Не всегда при разработке программы имеется возможность делать это на реальной сети, особенно на начальных этапах написания кода и отладки. Как уже было сказано выше, в библиотеках реализована возможность имитации работы в сети путем запуска нескольких экземпляров приложения на одном компьютере. Это существенно облегчает отладку, поскольку отладчик Microsoft Visual Studio можно "на лету" подцепить к любому выполняющемуся процессу.

4. Тестовая задача и алгоритм ее решения

В качестве тестовой задачи использовалась задача о клике. Напомним, что кликой называется любой полный подграф неориентированного графа (т.е. подграф, в котором каждая вершина соединена с каждой). Задача состоит в нахождении клики максимального размера.

Как известно, задача о клике относится к так называемым NP-полным задачам, для которых на данный момент не известно алгоритма решения с полиномиальной трудоемкостью. Так как более быстрого варианта нахождения решения нет, то остается лишь один способ – перебор, разумеется, не полный, а оптимизированный. Одним из лучших на сегодняшний день алгоритмов для решения задачи о клике считается алгоритм Брона–Кербоша (вариант метода ветвей и границ) [6].

Данный алгоритм оперирует тремя множествами вершин графа:

- Множество *compsub* – множество, содержащее на каждом шаге рекурсии полный подграф для данного шага. Строится рекурсивно.
- Множество *candidates* – множество вершин, которые могут увеличить *compsub*.
- Множество *not* – множество вершин, которые уже использовались для расширения *compsub* на предыдущих шагах алгоритма.

Алгоритм является рекурсивной процедурой, применяемой к этим трем множествам:

ПРОЦЕДУРА $bron\_kerbosh(compsub, candidates, not)$
0 ПОКА *candidates* НЕ пусто И *not* НЕ содержит вершины,
СОЕДИНЕННОЙ СО ВСЕМИ вершинами из *candidates*
ВЫПОЛНЯТЬ
1 Выбираем вершину v из *candidates* и добавляем её в *compsub*
2 Формируем $new\_candidates$ и new_not, удаляя из *candidates* и *not*

вершины, НЕ СОЕДИНЕННЫЕ с v
3 ЕСЛИ $new\_candidates$ и new_not пусты
4 ТО $compsub$ – клика
5 ИНАЧЕ
рекурсивно вызываем
$bron\_kerbosh(compsub, new\_candidates, new\_not)$
6 Удаляем вершину v из $compsub$ и $candidates$, добавляем ее в not

Здесь в отличие от варианта из [6] множество $compsub$ сделано параметром процедуры для того, чтобы последнюю можно было использовать как часть параллельного алгоритма. Описанный алгоритм (последовательный) также был реализован и использовался для оценки ускорения, которого можно достичь за счет параллельной работы.

Для обеспечения возможности распараллеливания приведенный алгоритм решения задачи можно переформулировать следующим образом:

ПРОЦЕДУРА $par\_bron\_kerbosh(compsub, candidates, not, bound)$
0 ЕСЛИ not содержит вершину, СОЕДИНЕННУЮ СО ВСЕМИ
вершинами из $candidates$
ТО завершить процедуру
1 ЕСЛИ размер $candidates$ не превышает $bound$
ТО вызываем последовательный вариант
$bron\_kerbosh(compsub, candidates, not)$
завершаем процедуру
2 Выбираем первую вершину v из $candidates$
3 Формируем $new\_candidates$ и new_not, удаляя из $candidates$ и not
вершины, НЕ СОЕДИНЕННЫЕ с v
4 ЕСЛИ $new\_candidates$ и new_not пусты
ТО $compsub \cup v$ – клика
5 ИНАЧЕ вызываем
$par\_bron\_kerbosh(compsub \cup v, new\_candidates, new\_not, bound)$
6 Удаляем вершину v из $compsub$ и $candidates$, добавляем ее в not
7 Рекурсивно вызываем $par\_bron\_kerbosh(compsub, candidates, not, bound)$

Здесь дополнительно фигурирует параметр $bound$, задающий предельный размер списка кандидатов для дальнейшего разбиения задачи на параллельные ветви (такое ограничение нужно для сокращения накладных расходов на распараллеливание [1]). Очевидно, что вызовы из пунктов 5 и 7 могут выполняться параллельно. Здесь для компактности изложения опущена проверка перспективности оставшихся на ветви вычислений (все-таки это метод ветвей и границ).

В [7] для тестирования версии РП-библиотеки для языка C++ алгоритм выстраивался примерно так же. Там же было отмечено, что трудоемкость вычислений на параллельных ветвях 5 (назовем ее "левой") и 7 ("правая" ветвь) заметно различается. При проведении описанного там эксперимента эту особенность можно было даже считать полезной, поскольку она позволяла оценить качество работы распределительного механизма, что, кстати, и было основной целью.

Для оценки работоспособности библиотек поддержки РП-программирования для .NET Framework автором был в качестве первого шага реализован только что описанный параллельный алгоритм ее решения на языке C# с использованием рассматриваемых компонентов [4,5]. Для тестирования генерировался случайный граф с заданным количеством вершин N и вероятности наличия ребра p. Очень быстро стало ясно, что достигаемое ускорение не соответствует ожиданиям, например, на графе с $N = 100, p = 0.5$ оно даже на сети с 10 рабочими станциями едва превышало 2. Как выяснилось, несмотря на кажущуюся естественность этого варианта рекурсивного распараллеливания, он обладает существенным недостатком.

При сравнительно небольшой плотности графа трудоемкости "левой" и "правой" ветвей различаются настолько сильно, что разбиение задачи на две ветки приводит к "отщипыванию" очень небольших кусочков вычислений, так что механизм динамической балансировки просто лишен возможности равномерно распределить работу. В эксперименте же, описанном в [7], вероятность наличия ребра бралась равной 0.9, поэтому при не слишком большом количестве исполнителей работу можно было распределить более или менее равномерно, что и обусловило хорошие показатели достигнутого ускорения.

В качестве альтернативного способа выстраивания параллельного алгоритма рассмотрим следующий. Пусть $SubTask_n$ – подмножество всех вариантов, построенное для следующих исходных значений трех основных подмножеств алгоритма Брона–Кербоша:

- Множество *compsub* состоит из одной вершины n.
- Множество *candidates* состоит из вершин, смежных с n.
- Множество *not* состоит из вершин с номерами от 0 до $n-1$ (при программировании удобнее использовать нумерацию от нуля).

Очевидно, что перебирать варианты $SubTask_n$, где n лежит в пределах от 0 до $N-1$, можно независимо, причем пересечение этих множеств пусто, а объединение содержит в себе все возможные клики. Для поиска решения на каждом множестве $SubTask_n$ вполне годится приведенный выше последовательный алгоритм Брона–Кербоша. Можно было бы и здесь применить распараллеливание, но смысла, по-видимому, в этом нет, по крайней мере, в случае, если N хотя бы в несколько раз больше количества имеющихся процессорных модулей, а это верно практически всегда, когда вообще есть смысл задействовать параллельный алгоритм.

Теперь рекурсивно-параллельный алгоритм решения нашей задачи можно выстроить по традиционной схеме [1], а именно: множество подмножеств $SubTask_n$, где n лежит в пределах от 0 до $N-1$, разбить пополам, потом еще раз и так далее, возможно, задав некоторый порог, после которого идет последовательная обработка в цикле. Этот вариант был также запрограммирован и продемонстрировал хорошее ускорение, однако и его, как выяснилось, можно улучшить.

Причина, по которой мы еще имеем возможность увеличить эффективность параллельного исполнения нашего алгоритма, заключается в характере изменения трудоемкости перебора вариантов из множества $SubTask_n$ с увеличением n. То, что она уменьшается, достаточно очевидно, однако интерес представляет характер ее зависимости от n.

Для получения ответа на этот вопрос было сгенерировано некоторое количество случайных графов с числом вершин N от 50 до 300 и вероятностью ребра p от 0.1 до 0.9. Эти же графы впоследствии использовались при изучении поведения параллельного алгоритма. На них была собрана статистика о трудоемкости перебора $SubTask_n$. В качестве единицы измерения бралось количество рекурсивных вызовов последовательной процедуры $bron\_kerbosh()$ для выполнения работы. Типичный характер зависимости этой величины от n демонстрируется графиком на рис. 1.

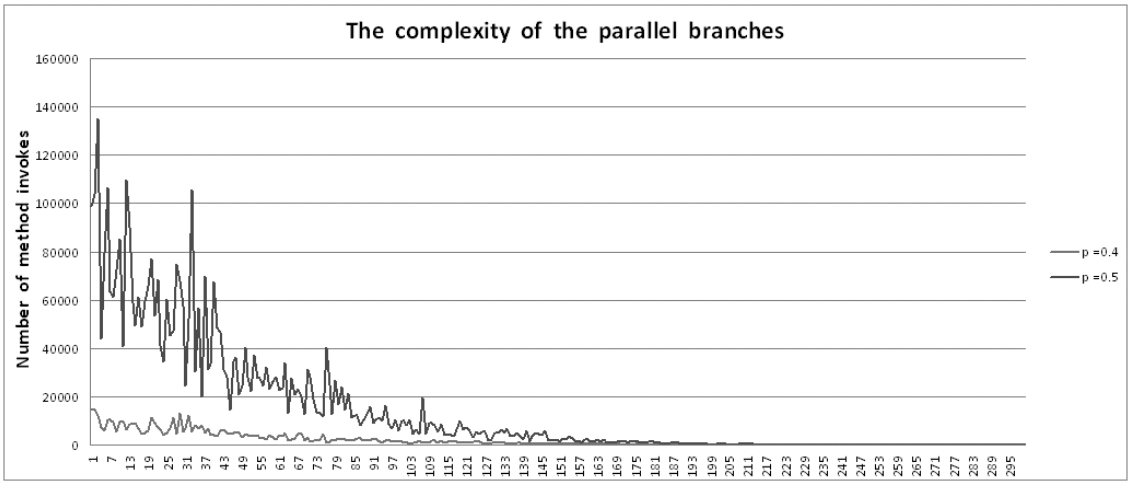

Рис. 1. Трудоемкость ветвей параллельного алгоритма поиска максимальной клики

Здесь $N = 300$, верхний график соответствует значению $p = 0.5$, а нижний — $p = 0.4$. Графики наглядно демонстрируют экспоненциальный характер уменьшения трудоемкости подзадач $SubTask_n$ с увеличением n. Для больших значений p скорость уменьшения трудоемкости еще выше. По этой причине разбивать работу в соответствии со стандартной схемой неразумно, поскольку емкость листьевых активаций сильно разнится, и возникает необходимость многочисленных передач кусков вычислений с модуля на модуль.

Предлагаемая модификация алгоритма состоит в том, что работа сначала делится на множества $SubTask_n$ с четным и нечетным значением n, на следующем уровне в одну группу попадают уже множества $SubTask_n$ с одинаковым остатком от деления n на 4 и так далее. После того, как количество порожденных активаций достигнет заданного программистом уровня, начинается обычный цикл. Такое деление приводит к достаточно равномерному распределению работы по листьевым активациям, результаты численного эксперимента, приведенные ниже, относятся именно к этому варианту алгоритма.

5. Результаты эксперимента

В процессе тестирования использовались компьютеры на базе четырехъядерного процессора Intel Core i3 с тактовой частотой 3.07 GHz и 4 GB оперативной памяти, работающие под управлением 64-разрядной ОС Windows 7. Пропускная способность сети 100 Mb/s.

Описанный выше алгоритм был реализован на языке C# с использованием библиотек [4, 5]. В таблице 1 приводятся данные о времени выполнения алгоритма в зависимости от количества задействованных рабочих станций и наблюдаемом ускорении. Надо иметь в виду, что при повторении эксперимента разброс полученных временных показателей в пределах 10% является вполне обычным.

Граф на 300 вершинах, вероятность ребра $p = 0.6$, Время работы последовательного алгоритма (в мс) 258478			
Кол-во ПМ	Время в мс	Ускорение (отн. парал.)	Ускорение (отн. посл.)
1	335899	1	0.769511
2	82286	4.082092	3.141215
4	49402	6.7993	5.232136
8	36838	9.118275	7.016613
12	30522	11.00514	8.46858
Граф на 300 вершинах, вероятность ребра $p = 0.7$, Время работы последовательного алгоритма (в мс) 10620270			
Кол-во ПМ	Время в мс	Ускорение (отн. парал.)	Ускорение (отн. посл.)
1	12192987	1	0.871015
2	3245722	3.756633	3.272082
4	1659134	7.349007	6.401092
8	1131466	10.77627	9.386292
12	900283	13.5435	11.79659

Таблица 1. Ускорение в зависимости от количества процессорных модулей

Можно отметить, что в ряде случаев ускорение не только по отношению ко времени работы параллельного алгоритма на одном компьютере, но и по отношению ко времени работы последовательного его варианта превышает количество задействованных ПМ. Это объясняется в основном тем, что параллельный алгоритм в отличие от последовательного использует многоядерность компьютера. Кроме того, на увеличении ускорения сказывается и тот факт, что информация о текущем рекорде передается немедленно всем исполнителям, и значительное количество неперспективных ветвей вычислений отсекается раньше. Поэтому даже при запуске на одном компьютере двух экземпляров приложения они зачастую выполняют работу быстрее, чем один экземпляр.

Список литературы

1. *Васильчиков В.В.* Средства параллельного программирования для вычислительных систем с динамической балансировкой загрузки. Ярославль: ЯрГУ, 2001. [*Vasilchikov V.V.* Sredstva parallelnogo programmirovaniya dlya vychislitelnykh sistem s dinamicheskoy balansirovkoy zagruzki. Yaroslavl: YarGU, 2001 (in Russian)].

2. *Васильчиков В.В., Шубин А.В.* Библиотека параллельного исполнения rpC-программ для Win32 // Моделирование и анализ информационных систем. 2008. Т. 15, №1. С. 37–40. [*Vasilchikov V.V., Shubin A.V.* The library for parallel execution of rpC-programs under

Win32 // Modeling and Analysis of Information Systems. 2008. V. 15, No 1. P. 37–40 (in Russian)].

3. *Бойцов Е.А., Васильчиков В.В.* Кроссплатформенная библиотека параллельного выполнения rpC-программ // Современные проблемы математики и информатики: Сборник научных трудов молодых ученых, аспирантов и студентов / Яросл. гос. ун-т им. П.Г. Демидова. Ярославль, 2011. Вып. 12. С. 71–81. [*Boytsov E.A., Vasilchikov V.V.* Krossplatformennaya biblioteka parallelnogo vypolneniya rpC-programm // Sovremennye problemy matematiki i informatiki: Sbornik nauchnykh trudov molodykh uchenykh, aspirantov i studentov / Yarosl. gos. un-t im. P.G. Demidova. Yaroslavl, 2011. Vyp. 12. S. 71–81 (in Russian)].

4. *Васильчиков В.В.* Коммуникационный модуль для организации полносвязного соединения компьютеров в локальной сети с использованием .NET Framework. Свидетельство о государственной регистрации программы для ЭВМ № 2013619925, 2013. [*Vasilchikov V.V.* Kommunikatsionnyy modul dlya organizatsii polnosvyaznogo soedineniya kompyuterov v lokalnoy seti s ispolzovaniem .NET Framework. Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2013619925, 2013 (in Russian)].

5. *Васильчиков В.В.* Библиотека поддержки рекурсивно-параллельного программирования для .NET Framework. Свидетельство о государственной регистрации программы для ЭВМ № 2013619926, 2013. [*Vasilchikov V.V.* Biblioteka podderzhki rekursivno-parallelnogo programmirovaniya dlya .NET Framework. Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2013619926, 2013 (in Russian)].

6. *Bron C., Kerbosh J.* Algorithm 457 – Finding all cliques of an undirected graph // Comm. of ACM. 1973. 16. P. 575—577.

7. *Бойцов Е.А., Васильчиков В.В.* Решение задачи о клике на языке rpC с помощью библиотеки RPM // Заметки по информатике и математике: Сборник статей / Яросл. гос. ун-т им. П.Г. Демидова. Ярославль, 2011. С. 28–37. [*Boytsov E.A., Vasilchikov V.V.* Reshenie zadachi o klike na yazyke rpC s pomoshch'yu biblioteki RPM // Zametki po informatike i matematike. Sbornik statey / Yarosl. gos. un-t im. P.G. Demidova. Yaroslavl, 2011. S. 28–37 (in Russian)].

On the Recursive-Parallel Programming for the .NET Framework

Vasilchikov V. V.

P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia

Keywords: parallel computing, recursion, .NET

The paper describes software components to support recursive-parallel programming for the .NET Framework. They are dynamic link libraries providing the necessary functionality for developing and debugging applications for parallel execution on a local network. Communication module library classes provide user-friendly software tools to establish "each with each" network connection and reliable asynchronous transmission for serializable objects. Classes of the recursive-parallel programming library provide representation of parallel computation branches as migratory processes, their initial distribution over the network, the transmission parameters and return results with the necessary synchronization, dynamic reallocation of work for load balancing and also sharing data processing. By this example it also describes some variants of the recursive-parallel algorithm to solve the problem of finding a maximum clique in a non-oriented graph and the results of testing the considered components.

Сведения об авторе:
Васильчиков Владимир Васильевич,
Ярославский государственный университет им. П.Г. Демидова,
канд. техн. наук, зав. кафедрой вычислительных и программных систем