УДК 004.93

СИСТЕМА ПАРАЛЛЕЛЬНОГО РАСПРЕДЕЛЕННОГО ПРОГРАММИРОВАНИЯ MC# 2.0

А. В. Петров[1], Ю. П. Сердюк[2]

Язык программирования MC# является универсальным высокоуровневым языком программирования, предназначенным для создания программ, работающих на всем спектре параллельных архитектур — многоядерных процессорах, кластерах и Grid-системах. Рассматриваются основные идеи и специфические конструкции языка MC#. Перечисляются отличия системы программирования MC# 2.0 от ее предыдущей версии и обсуждаются вопросы многопоточного программирования в рамках этой системы. Приводится обзор MC# Grid System — системы исполнения MC#-программ на Grid-сетях. Описывается опыт использования этой системы для распределенного рендеринга изображений с помощью пакета Povray. Формулируются основные направления дальнейшей работы и отмечается применение языка MC# в учебно-образовательных целях. Статья подготовлена по материалам докладов авторов на Всероссийской научной конференции "Научный сервис в сети Интернет: многоядерный компьютерный мир" (http://agora.guru.ru/abrau2007).

Ключевые слова: параллельное распределенное программирование, параллельное программирование, grid-сети, распределенный рендеринг изображений, языки программирования.

1. Введение. Широкое распространение вычислительных систем с массовым параллелизмом, таких как многоядерные процессоры, кластеры и Grid-архитектуры, вновь поставило вопрос о разработке высокоуровневых, мощных и удобных языков программирования, позволяющих создавать сложное, но в то же время и надежное программное обеспечение, эффективно использующее возможности параллельных распределенных вычислений и легко масштабируемое на заданное количество процессоров, узлов и машин.

Доступные на настоящий момент программные интерфейсы для организации параллельных вычислений, такие как OpenMP [1] (для систем с разделяемой памятью) и MPI [2] (для систем с передачей сообщений), созданы для языков C и Фортран, а потому являются низкоуровневыми и не адекватными современным языкам объектно-ориентированного программирования, таким как C++, C# и Java. Кроме того, эти интерфейсы базируются на использовании библиотек, а не на соответствующих конструкциях языков программирования.

С другой стороны, программные средства для организации Grid-вычислений, такие как Globus Toolkit [3], предназначены для сбалансированного распределения множества заданий (приложений) по вычислительной сети, но не для распараллеливания отдельных программ как таковых [4]. Кроме того, эти средства очень сложны в конфигурировании, развертывании и использовании.

Язык MC# [11] есть адаптация базовой идеи языка Polyphonic C# [5] (более точно, join-исчисления [6]) на случай распределенных вычислений. В то же время, MC# включает в себя конструкции, которые делают его средством высокоуровневого многопоточного программирования для мультиядерных процессоров.

Базовая концепция параллелизма языка MC# заключена в понятиях "асинхронного" (async) и "перемещаемого" (movable) методов. В тексте программы в определение таких методов добавляется одно из двух специальных ключевых слов async или movable:

```
modifiers  [ async | movable ] method_name ( arguments )
{
 < method body>
}
```

Вызов async-метода аналогичен порождению нового (локального) потока, в котором будет выполняться тело этого метода. При вызове же movable-метода он может быть спланирован для исполнения на

[1] Рыбинская государственная авиационная технологическая академия, факультет радиоэлектроники и информатики, ул. Пушкина, 53, 152934, г. Рыбинск, Ярославская обл.; e-mail: avpetrov@computer.org

[2] Исследовательский центр искусственного интеллекта Института программных систем РАН, 152020, г. Переславль-Залесский, Ярославская обл.; e-mail: Yury@serdyuk.botik.ru

любом узле кластера или Grid-сети. Взаимодействие async- и movable-методов осуществляется через каналы и обработчики, являющиеся специальными синтаксическими категориями языка MC#. Интересно отметить, что после введения этих средств в язык MC# понятие (синхронного и асинхронного) канала было также введено в состав базовых объектов библиотеки Joins [12] — реализации функциональности языка Polyphonic C# в виде библиотеки для платформы .NET Framework 2.0.

Определение каналов и связок в MC#-программах производится с помощью связок в стиле языка Polyphonic C#:

```
public handler getInt int ( )   &  public channel  sendInt ( int x )
{
  return ( x );
}
```

Тело связки исполняется только в случае вызова всех методов (каналов и обработчика), присутствующих в заголовке связки. Этих базовых средств языка MC# оказывается достаточно для построения любого параллельного распределенного приложения с произвольной схемой взаимодействия его отдельных компонентов [7].

2. Многопоточное программирование на языке MC# 2.0. В декабре 2006 г. специалисты университета г. Беркли (США) опубликовали отчет “The Landscape of Parallel Computing Research: A View from Berkeley” [13] о современном состоянии исследований в области параллельных вычислений. В нем отмечается бурный рост выпуска процессоров с многоядерной архитектурой, когда рынок постепенно заполняют компьютеры теперь уже с 4- и 8-ядерными процессорами, и предсказывается появление систем с 16 и 32 ядрами для массового применения. Более того, в начале 2007 г. корпорация Intel представила 80-ядерный однокристальный процессор, получивший название Teraflops Research Chip [14]: этот процессор может достигать производительности 1,01 терафлоп при минимальной тактовой частоте ядра 3,16 ГГц. По прогнозам Intel, к 2010 г. четверть объема всех поставляемых серверов будут иметь терафлопную производительность. Авторы упомянутого отчета в качестве основного своего вывода утверждают, что главное стратегическое направление развития параллельных технологий — это упрощение процессов создания программ, которые способны эффективно работать на параллельных архитектурах и, в частности, на чипах, насчитывающих (многие) тысячи ядер/процессоров. При этом особую значимость приобретает выработка новых моделей программирования, которые не зависят от числа процессоров и являются универсальными, т.е. допускают распараллеливание как по функциям, так и по данным. Появляющиеся в настоящий момент модели многопоточного программирования для мультиядерных процессоров можно разбить на две группы: библиотеки, расширяющие традиционные языки средствами многопоточного программирования, и новые высокоуровневые языки параллельного программирования. Типичными представителями первой группы являются такие общеизвестные технологии, как OpenMP [15], Intel Threading Building Blocks [16], а также пакет Qt Concurrent [17]. Чаще всего они представляют собой набор компонентов и классов (прежде всего ориентированных на язык C++) для создания многопоточных приложений без использования классических низкоуровневых механизмов, применяемых в параллельных алгоритмах, таких, как критические секции, флаги, семафоры и др. Поддержка параллельной обработки включена непосредственно в такие компоненты (например, автоматическое масштабирование на количество доступных ядер), а прикладному программисту предоставляется API (Application Programming Interfaces) для их использования (обычно на основе параметризованных по типам (шаблонов) классов).

Ко второй группе средств относятся такие языки, как UPC (Unified Parallel C [18]), Chapel [19], Titanium [20], Fortress [21], X10 [22], а также язык MC#, разработанный в Институте программных систем РАН, г. Переславль-Залесский. Следует отметить, что многие из этих языков стали номинантами HPC Challenge Award Competition 2006 [23] и были отмечены на конференции SuperComputing’06 в г. Тампа, США (ноябрь 2006 г.).

Основное отличие этих языков от библиотек для многопоточного программирования состоит в том, что они включают в себя новые, более абстрактные и высокоуровневые конструкции, позволяющие точно, компактно и понятно для человека описывать параллельные процессы и их взаимодействие между собой. В этих языках присутствуют три основных вида конструкций:

— конструкции для порождения параллельных процессов (потоков): async-методы, activities и т.д.;

— средства синхронизации параллельных процессов: барьеры, связки (chords), конструкции finish, clocks и др.;

— средства распределения данных между процессами: shared-переменные, regions, distributions и т.п.

Ключевыми понятиями первоначальной версии языка MC# [8] были так называемые movable-методы — методы, которые могут быть спланированы для исполнения на удаленной машине, а также одно-

направленные каналы и BD-каналы — каналы, по которым данные могут передаваться в обоих направлениях. В качестве средства синхронизации в этой версии использовались связки (chords) в стиле языка Polyphonic C#. Основные отличия системы программирования MC# 2.0 [7] от исходного варианта состоят в следующем:

— добавлены async-методы и система автоматической балансировки нагрузки на ядра/процессоры локальной машины;

— добавлены обработчики канальных сообщений взамен BD-каналов;

— изменена семантика работы с каналами и обработчиками;

— реализована интеграция языка MC# в Microsoft Visual Studio 2005.

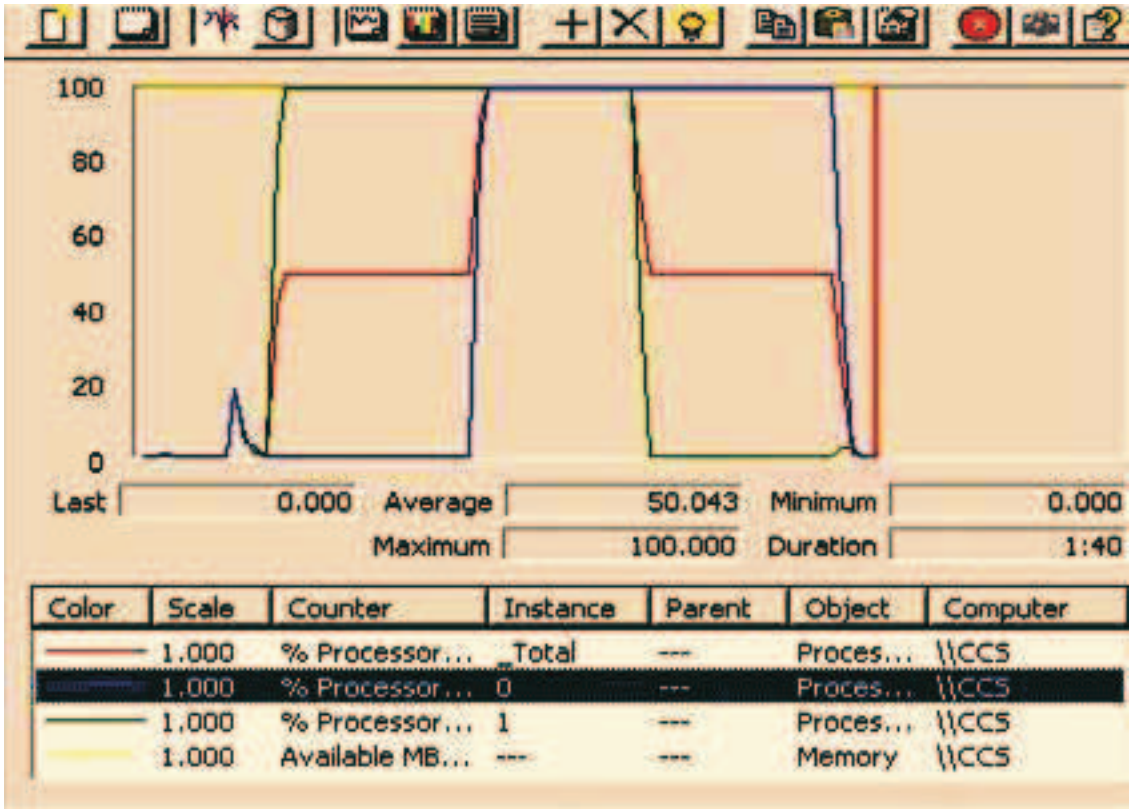

Рис. 1. Автоматическая балансировка нагрузки на процессоры локальной машины

В дополнение к механизму сбалансированного распределения movable-методов по машинам сети (в частности, узлам кластера) в систему программирования MC# 2.0 для ОС Windows включена подсистема автоматической балансировки нагрузки на ядра/процессоры локальной машины. Она базируется на оценке степени текущей загруженности процессоров машины и привязке вновь запускаемых async-методов к наименее загруженным из них. На рис. 1 иллюстрируются два асинхронных метода, запущенных со сдвигом по времени.

На рис. 2 и 3 показаны графики времени работы последовательных и параллельных вариантов программ перемножения матриц с удвоенной точностью и быстрого преобразования Фурье (БПФ).

Добавление обработчиков канальных сообщений (конструкций handler) взамен BD-каналов позволило получить более компактное и теоретически обоснованное множество базовых конструкций языка MC# [9]. В соответствии со спецификой языка MC#, каналы и обработчики могут передаваться в качестве аргументов методов (в частности, async- и movable-методов) отдельно от объектов, которым они принадлежат (в рамках которых они объявлены). Кроме того, в MC# 2.0 изменилась семантика работы с ними при распределенных вычислениях: если каналы и обработчики копируются (пересылаются) на удаленную машину автономно или в составе некоторого объекта, то там они становятся прокси-объектами, т.е. посредниками для исходных каналов и обработчиков. Эта подмена скрыта от прикладного программиста — он может использовать переданные каналы и обработчики (а в действительности, прокси-объекты для них) на удаленной машине так же, как и оригинальные: как обычно, все действия над прокси-объектами переправляются Runtime-системой языка MC# на исходные каналы и обработчики. В этом отношении они

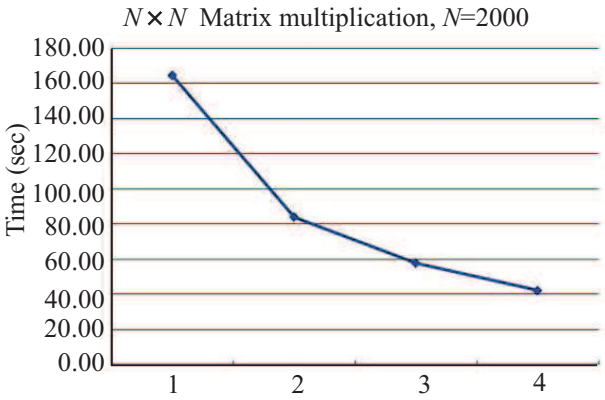

Рис. 2. График времени работы программы перемножения матриц

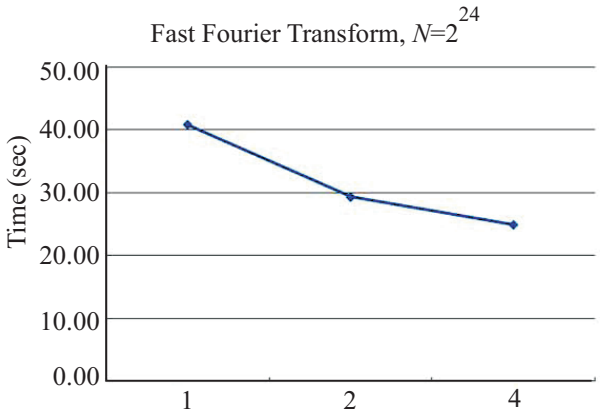

Рис. 3. График времени работы программы БПФ

отличаются от обычных объектов: действия над последними на удаленной машине на исходные объекты не передаются. Более подробно об этом, а также примеры программ, использующих каналы и обработчики, можно найти в [7]. Для локальной версии языка MC#, использующей async-методы, для ОС Windows реализована интеграция с Microsoft Visual Studio 2005 (рис. 4).

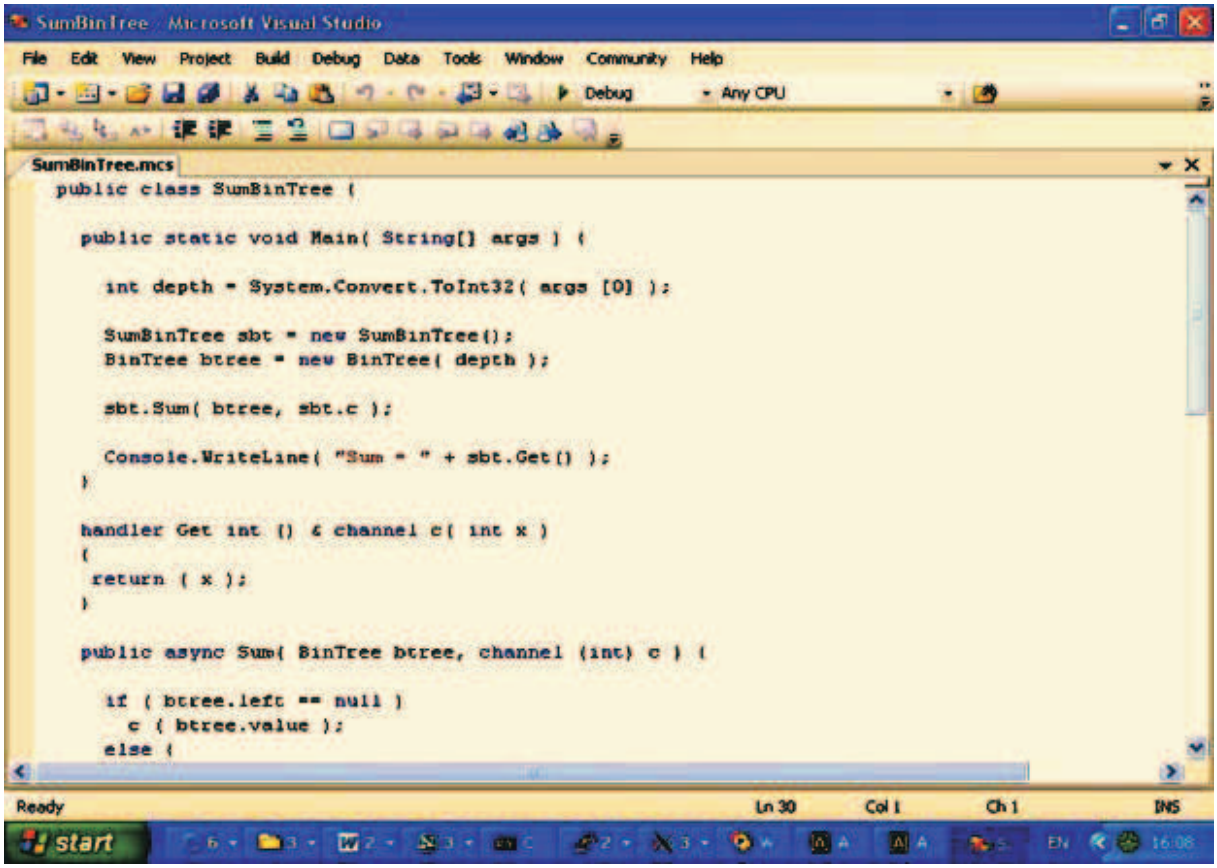

Рис. 4. Интеграция MC# в Visual Studio 2005

В рамках Visual Studio 2005 программист теперь может создавать MC#-проекты, состоящие из MC#- и C#-модулей, компилировать их и запускать на выполнение.

В начале статьи мы отмечали, что одним из средств повышения уровня абстракции в параллельном программировании является разработка библиотек шаблонов для решения типовых задач в этой области.

Примерами таких библиотек являются Intel Threading Building Blocks (далее Intel TBB), Qt Concurrent и др. Среди шаблонов, которые в них реализованы, имеются parallel_for, parallel_reduce, parallel_while, pipelining и др., а также такой специальный компонент как "планировщик задач" (task scheduler). Основное назначение этих шаблонов — освободить прикладного программиста от явного порождения параллельных потоков и управления ими при выполнении циклических вычислений. В частности, в библиотеке Intel TBB имеются классы AutoPartitioner и ManualPartitioner, с помощью которых производится либо автоматическое, либо управляемое программистом отображение пространства итераций на отдельные потоки. Так, типичными аргументами вызова шаблона parallel_for в Intel TBB являются:

— размер пространства итераций,

— функциональный объект, включающий в себя обрабатывающую функцию и обрабатываемые данные,

— тип разбиения пространства итераций: автоматический или ручной.

Пример такого вызова:

```
parallel_for ( blocked_range <size_t> ( 0, n ), Apply ( a ), auto_partitioner() );
```

Поскольку язык MC# является универсальным языком параллельного программирования, то на нем также могут быть реализованы шаблоны, перечисленные выше. Часть API-библиотеки шаблонов на языке MC# показана в следующем примере.

```
namespace MCSharp
{
    class Algorithm
    {
        //ParallelFor API
        public delegate async Exec(Range2d   range, channel ()  result);
        public static void ParallelFor(Exec exe, Range2d   range, IPartioner  p);
        . . .
        //ParallelReduce API
        public delegate async Reduce<T>(T Result);
        public delegate async Exec<T>(Range1d   range, channel <T> (T) result);

        public static void ParallelReduce<T> (Exec<T> exe, Reduce<T> reduce,
                                              Range1d range, IPartioner p)
        . . .
    }
}
```

Основные отличия шаблонов MC# от шаблонов Intel TBB состоят в следующем:

— пространство итераций (поле индексов) задается специальными объектами Range1d и Range2d;

— функция обработки части пространства итераций задается посредством асинхронного делегата;

— возврат результатов (и, в частности, сигнала об окончании работы) обрабатывающей функции производится через каналы.

Следует отметить, что подход на основе введения в язык специальной конструкции, задающей абстрактное пространство итераций, принят также и в языке X10. Данная конструкция называется там region и может задавать поле индексов произвольной размерности. Пример реализации алгоритма перемножения квадратных матриц с помощью шаблона parallel_for для Intel TBB и MC# приведен в таблице.

Заметим, что в Intel TBB проблема потокобезопасности — корректной работы нескольких потоков над одними и теми же данными — решается введением в описание функционального объекта специального конструктора копирования, создающего отдельную копию аргументов обрабатывающей функции для каждого потока. На настоящий момент при использовании шаблонов MC# потокобезопасность обеспечивается самим программистом путем введения в обрабатывающую функцию необходимых операторов копирования аргументов. Облегчение этой задачи для программиста является нашим будущим направлением исследований в этой области. Тем не менее, параллельное программирование с помощью шаблонов имеет и свои недостатки. Основной из них состоит в том, что даже относительно несложные параллельные алгоритмы не всегда соответствуют имеющимся стандартным шаблонам. Так, например, текст программы нахождения простых чисел методом просеивания (решета Эратосфена), реализованный с помощью шаблона parallel_reduce из библиотеки Intel TBB, занимает семь страниц [24]. Соответствующие трудно-

Intel TBB:

```
#include "tbb/parallel_for.h"
#include "tbb/blocked_range2d.h"
using namespace tbb;
const size_t N = 300;
class MatrixMultiplyBody2D
{
   float (*my_a)[N];
   float (*my_b)[N];
   float (*my_c)[N];
public:
   void operator()
   (const blocked_range2d<size_t>& r ) const
   {
     float (*a)[N] = my_a;
     float (*b)[N] = my_b;
     float (*c)[N] = my_c;
     for( size_t i=r.rows().begin();
          i!=r.rows().end(); ++i )
     {
          for( size_t j=r.cols().begin();
               j!=r.cols().end(); ++j )
          {
          float sum = 0;
          for( size_t k=0; k<N; ++k )
               sum += a[i][k]*b[k][j];
          c[i][j] = sum;
          }
     }
   }
   MatrixMultiplyBody2D( float c[N][N],
          float a[N][N], float b[N][N] ) :
          my_a(a), my_b(b), my_c(c)
   {}
};
void ParallelMatrixMultiply(float c[N][N],
          float a[N][N], float b[N][N])
{
   parallel_for( blocked_range2d<size_t>(0, N,
          0, N),
          MatrixMultiplyBody2D(c,a,b),
          auto_partitioner());
}
```

Язык MC#:

```
using MCSharp;
class FooClass
{
 float[,] A;
 float[,] B;
 float[,] C;
 public void MMultiply(Range2d arr,
                       channel()complete)
 {
  for (int i = arr.row_start; i < arr.row_end;
       i++)
   for (int j = arr.col_start; j < arr.col_end;
        j++)
   {
     float sum = 0;
     for (int k = arr.col_start;
               k < arr.col_end; k++)
          sum += A[i, k] * B[k, j];
     C[i, j] = sum;
   }
   complete.Send();
 }
 void ParallelMatrixMultiply(float c[,],
                 float a[,], float b[,])
 {
   A=a;
   B=b;
   C=c;
   Algorithm.ParallelFor(
             new Algorithm.Exec(MMultiply),
             new Range2d(0, c.GetLength(0), 0,
             c.GetLength(0),));
 }
}
```

сти имеют место и при реализации этой программы с помощью библиотеки шаблонов MC#. С другой стороны, прямая запись этого алгоритма на языке MC# является компактной и прозрачной [7].

3. MC# Grid System. Тема "Grid-вычислений" остается популярной темой в области вычислительной науки и практики. С одной стороны, появляются новые, усовершенствованные версии традиционных Grid-инструментов, таких как Globus Toolkit 4.0 [25], Grid Programming Environment [26], а с другой стороны, продолжают развиваться системы на базе MPI для Grid-вычислений, такие как MPICH-GX [27]. Традиционные Grid-инструменты ориентированы на поддержку исполнения автономных заданий, которые не взаимодействуют между собой в процессе вычислений, что резко снижает круг задач, которые можно решить с помощью этих средств. Недостатками подходов, использующих ту или иную версию MPI для Grid-вычислений, являются традиционные недостатки этого интерфейса: фактически явное управление распределением вычислительных процессов по узлам, необходимость программировать вручную передачу сложных данных между узлами Grid-сети (осуществлять сериализацию/десериализацию данных), к которым добавляются проблемы установки, развертывания и конфигурирования самих MPI-систем.

Предлагаемый нами подход, который дает возможность устранить многие из названных выше недостатков, состоит в использовании высокоуровневого языка параллельного программирования MC#, позволяющего разрабатывать параллельные распределенные приложения, которые могут исполняться единым образом на всем спектре параллельных архитектур — от многоядерных процессоров до Grid-сетей. Компилятор этого языка совместно с Runtime-системой, ориентированной на поддержку исполнения программ на множестве кластеров и отдельных машин, и составляет основу системы MC# Grid System. Выбор в качестве базового языка C# (в отличие от языка Java, традиционно используемого в Grid-системах) дает возможность использовать такие его мощные расширения, как параметризуемые по типам классы (generic classes), средства работы с реляционными и слабоструктурированными данными из стандарта C# 3.0 и др. Поддержку исполнения MC#-программ на глобальной (Grid-) сети осуществляет Runtime-система — ключевой компонент MC# Grid System. Основными этапами ее работы являются:

— загрузка Grid-сети (инициализация Runtime-системы),

— запуск приложения на исполнение,

— остановка Grid-сети (завершение работы Runtime-системы).

Под Grid-сетью понимается совокупность кластерных вычислительных систем, а также отдельных машин, которые работают под управлением ОС Linux и на которых установлена система Mono — свободно распространяемая реализация платформы .NET для Unix-подобных систем [28]. Одна из машин этого множества назначается главной машиной Grid-сети — с нее происходит загрузка и остановка Grid-сети, а также запуск пользовательских приложений. Загрузка Grid-сети осуществляется запуском процедуры gboot. Ее параметром является имя XML-файла с описанием конфигурации Grid-сети. В этом файле описываются каждый кластер и каждая отдельная машина, на которых предполагается исполнение MC#-программ. Примером такого конфигурационного файла является следующая конструкция:

```
<?xml version="1.0"?>
<grid id="skif_al" host="skif.botik.ru" port="30003">
    <cluster id="skif" host="skif.botik.ru" port="30004" accesstype="ssh"
      mcsgpath="/home/mcsharp.grid.system/">
        <node host="localhost" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-11" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-12" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-13" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-14" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-21" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-22" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-23" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-24" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-31" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-32" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-33" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-34" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-41" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-42" port="30005" processors="2" accesstype="rsh"/>
        <node host="node-44" port="30005" processors="2" accesstype="rsh"/>
    </cluster> <!-- 32 -->
    <cluster id="al"   host="al.botik.ru"   port="30004" accesstype="ssh"
      mcsgpath="/home/mcsharp.grid.system/">
        <node host="localhost" port="30005" processors="8" accesstype="ssh"/>
    </cluster> <!-- 8 -->
    <cluster id="hpi1" host="hpi1.botik.ru" port="30004" accesstype="ssh"
      mcsgpath="/home/distr/mcsharp.grid.system/">
        <node host="localhost" port="30005" processors="2" accesstype="ssh"/>
    </cluster> <!-- 2 -->
    <cluster id="hpi2" host="hpi2.botik.ru"   port="30004" accesstype="ssh"
      mcsgpath="/home/distr/mcsharp.grid.system/">
        <node host="localhost" port="30005" processors="2" accesstype="ssh"/>
    </cluster> <!-- 2 -->
</grid> <!-- 44 -->
```

Атрибут accesstype задает тип входа на машину Grid-сети. Для осуществления автоматического входа на машины должен быть обеспечен (сконфигурирован) беспарольный вход через rsh/ssh [29]. Соответственно, на каждой из машин должны быть открыты порты, перечисленные в конфигурационном файле. Задание количества процессоров каждой машины используется Runtime-системой, поддерживающей автоматическое распределение нагрузки (а именно, запускаемых movable-методов) по узлам Grid-сети. Атрибут mcsgpath задает расположение библиотек Runtime-системы на удаленной машине, что позволяет свести к минимуму работу по установке и конфигурированию MC# Grid System на узлах Grid-сети. Поскольку фронтенды (главные узлы) кластеров часто включены в несколько сетей — обычно это частные (private) сети, объединяющие его с рабочими узлами кластера, и общие (public) глобальные сети, то в качестве их имен (атрибут host) должны указываться имена (адреса), общие для всех таких сетей. Перед запуском приложения его исполнимые модули и библиотеки должны быть размещены на каждом узле Grid-сети. Это размещение пользователь может выполнить с помощью специальной процедуры apploader. Перед запуском в Grid-сети MC#-программа может быть отлажена как локально, т.е. в рамках отдельной машины, так и распределенно в рамках одного кластера. Тем самым отсутствует необходимость отладки приложений непосредственно на глобальной сети.

Рендеринг изображений — это построение графических образов по их описаниям на формальном языке. В настоящее время разработано множество пакетов для решения этой задачи, среди которых одним из основных является пакет Povray (Persistence of Vision Ray Tracer) [30]. Этот пакет реализован на языке C++ и первоначально не был ориентирован на параллельные вычисления. Со временем появились дополнительные патчи к этому пакету, которые позволяют запускать его как MPI-программу на различных платформах. Однако каждый из этих патчей обычно ориентирован на какую-либо конкретную версию пакета Povray, и до сих пор не существует официальной параллельной реализации этого пакета. Тем не менее, существует очень простой способ, позволяющий построить параллельную версию программы Povray.

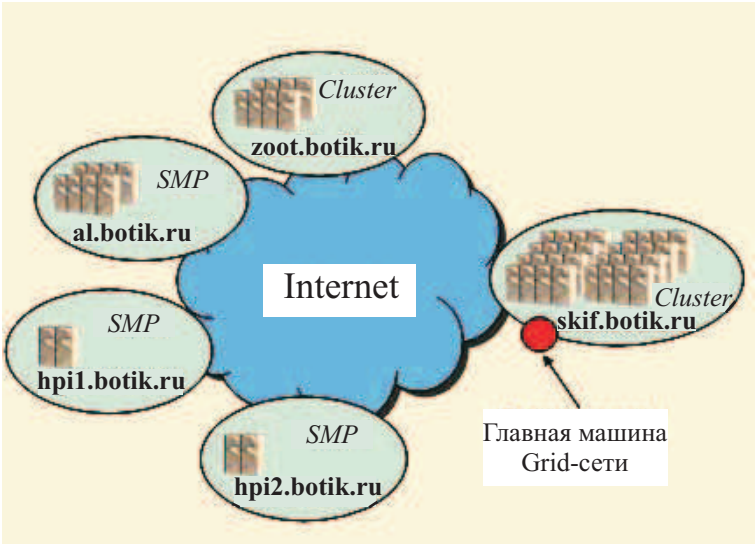

Рис. 5. Конфигурация Grid-сети для рендеринга изображений

Он использует тот факт, что эта программа имеет в качестве своих входных параметров, среди прочих, параметры +SC и +EC, с помощью которых задаются начальный и конечный столбец (а фактически, вертикальная полоса) изображения, рендеринг которого необходимо осуществить. Таким образом, зная ширину и высоту всего изображения, а также количество процессоров, на которых будет производиться рендеринг, можно разбить все изображение на отдельные фрагменты и запустить необходимое количество копий программы Povray с соответствующими значениями параметров +SC и +EC. Именно такой подход был выбран при реализации параллельной части программы рендеринга с помощью пакета Povray на языке MC#. Фактически, параллельная программа рендеринга на языке MC#, ориентированная на исполнение в Grid-среде, выполняет следующие действия:

— считывает файлы с описанием изображения, которое необходимо построить;

— на основании размеров изображения и заданного количества фрагментов, на которое оно разбивается, запускает соответствующее количество movable-методов; входными аргументами этих методов являются символьное описание требуемого изображения и параметры для старта программы Povray;

— каждый movable-метод, исполняясь на отдельном узле, запускает задачу Povray для рендеринга соответствующего фрагмента;

— по окончании работы Povray каждый movable-метод считывает построенный файл-изображение с построенным фрагментом и пересылает данный фрагмент в главную программу;

— по получении всех фрагментов главной программой строится результирующее изображение, которое записывается на диск и воспроизводится на экране.

Особенность выполнения этой программы в Grid-среде состоит в том, что пересылка файлов, а точнее,

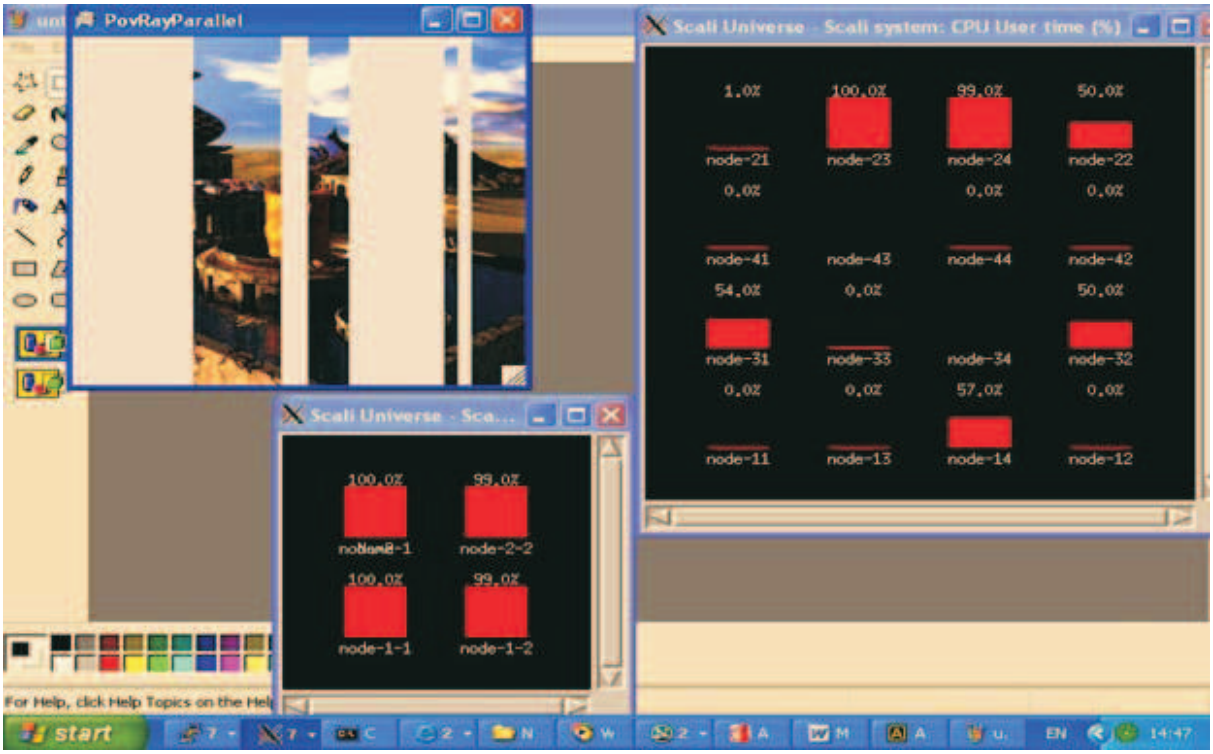

Рис. 6. Загрузка процессоров и частично построенное изображение при работе MC# Grid System

их содержимого, реализована через передачу аргументов movable-методов и пересылку по каналам, объявленным в MC#-программе.

Параллельная часть программы, написанная на языке MC#, имеет небольшой (по количеству строк) объем и легко разрабатывается за 1–2 дня. Конфигурация Grid-сети, на которой проводились эксперименты по рендерингу изображений, показана на рис. 5

Следует отметить, что применение платформы .NET, ориентированной на использование байт-кода, дает возможность применять в Grid-сети машины и кластеры с различной архитектурой и при этом избегать перекомпиляции приложений. В частности, общее количество процессоров в наших экспериментах достигало 56, среди которых были процессоры Itanium-2, AMD Athlon MP 2000+ и Pentium III. Оптимальная конфигурация, на которой достигались наилучшие результаты, состояла из кластера с 16-ю двухпроцессорными узлами на базе AMD Athlon MP 2000+, 8-процессорного сервера на базе Itanium-2 и двух двухпроцессорных серверов на основе Itanium-2. Использование в общем вычислительном поле кластера с малопроизводительными процессорами Pentium III почти в два раза ухудшало показатели скорости рендеринга. Скриншот, на котором показана загрузка

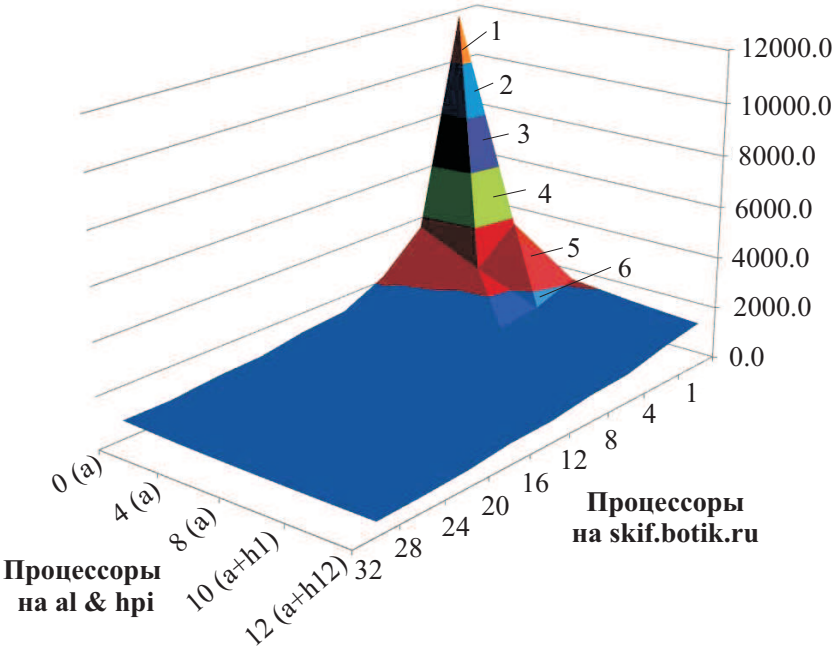

Рис. 7. График времени рендеринга изображения benchmark.pov размером 1024 × 768: 1) 10000.0–12000.0; 2) 8000.0–10000.0; 3) 6000.0–8000.0; 4) 4000.0–6000.0; 5) 2000.0–4000.0; 6) 0.0–2000.0

процессоров двух кластеров и окно с частично построенным изображением, представлен на рис. 6.

На рис. 7 представлен график зависимости времени рендеринга от количества использованных процессоров. По одной оси откладывалось количество использованных процессоров кластера на AMD Athlon MP 2000+ (количество процессоров от 1 до 32), по другой — количество процессоров Itanium-2 (на 8-процессорном сервере и двух одиночных двухпроцессорных серверах). В качестве изображения для рендеринга брался стандартный тест benchmark.pov с размером изображения 1024×768.

Проведенные эксперименты в рамках Grid-сети небольшого размера (из 1–2 кластеров и трех многопроцессорных серверов) дали положительные результаты как в смысле уменьшения времени рендеринга (в 21 раз по сравнению со временем исполнения на одном процессоре), так и в смысле упрощения разработки и запуска распределенных программ на "корпоративных" Grid-сетях. Данный подход кардинально отличается от используемых на данный момент Grid-технологий, которые фактически не позволяют распараллелить исполнение программы на все доступные машины, а дают лишь возможность отправить программу на одну из них с подходящими характеристиками [10]. Кроме того, наш подход значительно упрощает процедуру развертывания и конфигурирования Grid-сети — эта процедура сводится к установке .NET и MC# Grid System на каждой машине сети, настройке беспарольного входа с главной машины сети на все остальные и открытии одного порта на каждой из них.

Для дальнейшего упрощения настройки и управления Grid-сетью, а также для мониторинга исполнения приложения мы разрабатываем графическую консоль, реализующую эти функции.

Заключение. В данной работе была представлена система программирования MC# 2.0, одной из целей которой является значительное повышение продуктивности работы программистов, занимающихся созданием параллельных и распределенных приложений. Кроме того, высокоуровневость и прозрачность параллельных конструкций этого языка позволит привлечь к регулярному параллельному программированию значительно большее число программистов. Следует также отметить, что для достижения приемлемой производительности программ, написанных на языке MC# и предназначенных для исполнения на многоядерных процессорах, возможно использование специальных библиотек, ускоряющих работу численных алгоритмов, реализованных на языке C#. Среди таких библиотек можно отметить пакет NMath фирмы CenterSpace Software [31]. Основными направлениями дальнейшего развития системы программирования MC# являются разработка версии для Windows-кластеров, что позволит использовать ее на такого рода системах аналогично известному продукту Microsoft Windows Compute Cluster Server 2003, а также разработка средств автоматического распределения данных между async- и movable-методами. Последние из названных средств будут аналогичны средствам, развиваемым в рамках языка X10; они основаны на расширении нотации и действий над массивами, а также на введении новых языковых конструкций типа region и distribution. Что касается MC# Grid System, то здесь первоочередной задачей будет разработка новых Grid-приложений на языке MC#, среди которых факторизация очень больших натуральных чисел методом "квадратичного решета", планирование выполнения операций на технологическом оборудовании, квантово-механические расчеты и др.

К настоящему времени язык MC# начал использоваться и в учебно-образовательных целях. Учебные курсы по языку MC# были прочитаны на зимних школах Intel по параллельному программированию, состоявшихся в феврале 2007 г. в Санкт-Петербурге и Нижнем Новгороде [32]. Кроме того, на сайте библиотеки учебных курсов MSDN Academic Alliance [33] размещен учебный курс "Кластерные вычисления", один из разделов которого посвящен программированию на языке MC#. Язык MC# используется также в курсе "Параллельное программирование" в Рыбинской государственной авиационной технологической академии на кафедре "Математическое и программное обеспечение электронных вычислительных средств" [34].

В заключение, авторы хотели бы поблагодарить В. Гузева за участие в реализации системы программирования MC# и М. Коваленко и М. Недева за помощь в проведении вычислительных экспериментов.

СПИСОК ЛИТЕРАТУРЫ

1. OpenMP specifications (http://www.openmp.org/specs).
2. Message Passing Interface (http://www-unix.mcs.anl.gov/mpi/).
3. Globus Toolkit (http://www.globus.org/toolkit/).
4. *Silva V.* Crunch big numbers with GT3 using a quadratic sieve (http://www-128.ibm.com/developerworks/grid/library/gr-factor/).
5. *Benton N., Cardelli L., Fournet C.* Modern concurrency abstractions for C# // ACM Transactions on Programming Languages and Systems. 2004. **26**, N 5. 769–804.
6. *Fournet C., Gonthier G.* The join calculus: a language for distributed mobile programming // Proc. Applied Semantics Summer School. 2000. Lecture Notes in Computer Science. Vol. 2395. Berlin: Springer, 2000. 268–292.

7. *Serdyuk Yu.* MC# 2.0: a language for concurrent distributed programming based on .NET" // .NET Technologies 2006, 4th International Conference, Plzen, Czech Republic, 29 May – 1 June, 2006 (http://www.mcsharp.net/publications/NET2006.pdf).
8. *Guzev V., Serdyuk Yu.* Asynchronous parallel programming language based on the Microsoft .NET platform // PaCT-2003. Lecture Notes in Computer Science. Vol. 2763. Berlin: Springer, 2003. 236–243.
9. *Serdyuk Yu.* A formal basis for the MC# programming language (Extended Abstract) (http://www.mcsharp.net/publications/formal_basis_extended_abstract.pdf).
10. *Гузев В., Сердюк Ю.* MC# Grid System: подход к Grid-вычислениям на основе параллельного, распределенного языка программирования // Тр. конференции "Научный сервис в сети Интернет: распределенные вычислительные технологии". Новороссийск, сентябрь 2005 г. 41–43.
11. http://www.mcsharp.net
12. http://research.microsoft.com/ crusso/joins/
13. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf
14. http://www.intel.com/research/platform/terascale/teraflops.htm
15. http://www.openmp.org
16. http://www.intel.com/cd/software/products/asmo-na/eng/294797.htm
17. http://labs.trolltech.com/ page/Projects/Threads/QtConcurrent
18. http://upc.gwu.edu/
19. http://chapel.cs.washington.edu/
20. http://titanium.cs.berkeley.edu/
21. http://fortress.sunsource.net
22. http://x10.sourceforge.net/x10home.shtml
23. http://www.hpcchallenge.org/
24. http://softwareblogs.intel.com/2006/12/18/threading-building-blocks-solution-looking-for-a-problem/
25. http://www.globus.org/toolkit/
26. http://gpe4gtk.sourceforge.net/
27. http://www.moredream.org/mpich.htm
28. http://www.mono-project.com
29. http://cfm.gs.washington.edu/security/ssh/client-pkauth/
30. http://www.povray.org
31. http://www.centerspace.net/
32. http://www.itlab.unn.ru/?doc=835
33. http://www.microsoft.com/rus/msdnaa/curricula/Default.mspx
34. http://parallel-brains.narod.ru/

Поступила в редакцию
27.11.2007