ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 3(17), 2013, с.??—??

УДК 004.75, 004.428.4

И. А. Валуев, И. В. Морозов

GridMD: компактная переносимая библиотека C++ для управления распределенными вычислениями

Аннотация. В работе описаны архитектура, функциональность и примеры использования открытой библиотеки GridMD, предназначенной для разработки распределенных приложений, выполняющихся на современных Грид-системах и системах управления заданиями суперкомпьютерных кластеров. Библиотека изначально создавалась для разработчиков молекулярно-динамических приложений, однако впоследствии сфера применения библиотеки была значительно расширена. Сейчас она служит универсальным средством для создания приложений, поддерживающих распределенные вычисления, а также для управления удаленными задачами из компактного клиентского приложения средствами языка программирования C++. В первую очередь библиотека GridMD необходима при создании программ, которые содержат большое число вычислительных этапов с возможными связями по данным и могут эффективно выполняться в распределенной вычислительной среде.

Ключевые слова и фразы: распределенные вычисления, Грид-технологии, граф исполнения, молекулярная динамика, многомасштабное моделирование

Введение

Большинство масштабных численных экспериментов обладают характеристиками, позволяющими разбить эксперимент на ряд независимых стадий и выполнить их на распределенных вычислительных ресурсах. В простейших случаях таковыми задачами являются вариация начальных данных расчета с целью набора статистически независимых результатов, поиск оптимальных параметров модели или расчет зависимостей измеряемых величин от параметров численного эксперимента. Идея создания библиотеки GridMD – обеспечить автоматическое разбиение численного эксперимента на этапы, генерацию и обработку сценария исполнения, на основе инструкций,

включаемых в исходный текст программы [1-2]. Таким образом, GridMD является системой формирования и управления сценариями выполнения программы (workflow system) на распределенной вычислительной системе.

Системы управления сценариями [3-4] могут быть классифицированы как инфраструктурные и универсальные. Инфраструктурные проекты требуют поддержания определенной постоянно действующей среды (инфраструктуры), через которую осуществляется авторизация пользователей и постановка задач. В качестве такой инфраструктуры может выступать Grid (проекты Unicore, Triana, gUSE), распределенная база данных (проект Taverna). Универсальные системы могут взаимодействовать с широким классом систем планирования и выполнения заданий. Существующие универсальные системы (Kepler, Pegasus), как правило, используют планировщик Condor как основное средство запуска заданий на разных классах вычислительных ресурсов.

Еще одним критерием для классификации может служить то, для кого предназначена система: для конечного пользователя или разработчика программного обеспечения. Системы, ориентированные на конечного пользователя, часто предоставляют графический интерфейс, который служит для описания сценария выполнения программы. При этом специализированная настройка под конкретную задачу происходит на уровне отдельных элементов сценария (составных блоков, доступных к выбору в интерфейсе). Эти блоки настраиваются разработчиками промежуточного ПО так, чтобы отражать предметную область задачи (см., например, проект gUSE). Преимуществом такого подхода является универсальность интерфейса системы управления сценарием. Однако, существует и недостаток: разработчик промежуточного ПО для предметной области не обладает достаточными средствами управления на уровне сценариев и ему навязываются программные решения, определяемые высокоуровневым интерфейсом управления.

В отличие от большинства существующих систем и платформ управления распределенными сценариями, библиотека GridMD является компактным универсальным клиентским средством, ориентированным на разработчика. Для своего функционирования она не требует установки и поддержки сложных инфраструктур, таких как Грид-системы, порталы, системы совместного пользования, однако может запускать вычислительные задачи на них от имени пользователя. Кроме того, GridMD может использоваться для разработки предметно-ориентированных расчетных платформ, требующих низкоуровневого (скрытого от конечного пользователя) управления сценариями вычислений. Пример такого использования обсуждается ниже.

1. Архитектура и функциональность библиотеки GridMD

1.1. Основные компоненты библиотеки

Внутри библиотеки GridMD выделяются два уровня виртуализации ресурсов, с которыми может взаимодействовать программа пользователя: «менеджер сценариев» и «менеджер заданий» (Рис. 1). В обычном режиме менеджер сценариев, обрабатывая автоматически или вручную сгенерированный граф исполнения, отправляет менеджеру заданий запросы на выполнение отдельных наборов команд и пересылку данных. При этом менеджер заданий имеет унифицированный интерфейс для различных способов доступа к удаленной системе, а также для различных способов выполнения, сформированных по общему образцу заданий, начиная от простейшего выполнения скрипта в командном интерпретаторе, до постановки параллельной задачи в очередь с использованием различных систем управления заданиями (PBS, SLURM и др.).

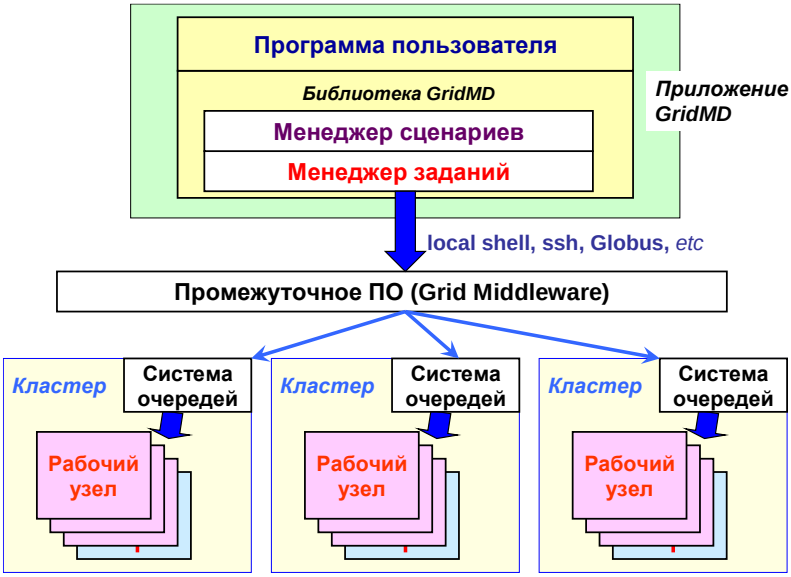

Рис. 1. Архитектура GridMD-приложения и взаимодействие с вычислительными ресурсами

В функции менеджера заданий входит подготовка удаленной системы к выполнению сценария, копирование входных данных для каждой отдельной задачи, запуск задачи с помощью выбранной системы очередей, контроль за ее выполнением и копирование выходных данных. Базовой подсистемой менеджера заданий является набор функций для выполнения удаленных команд и копирования файлов между локальной и удаленной системой. Разработчик может применять эти функции отдельно от оставшейся части библиотеки (без менеджера сценариев), получая таким образом, высокоуровневый стандартизированный интерфейс для выполнения команд (Unix или Windows shell) на локальной или удаленной системе. Доступ к удаленной системе осуществляется с помощью встроенного протокола ssh или внешней утилиты. В настоящее время менеджер заданий поддерживает запуск задач на системах под управлением

PBS(torque), SLURM, СУППЗ (ИПМ РАН), Globus, а также непосредственно запуск и контроль процессов в ОС Unix и Windows.

Менеджер сценариев GridMD использует стандартный способ формулировки потока обработки (workflow), основанный на направленном ациклическом графе сценария [5-6]. Основными элементами графа сценария являются узлы и связи. С узлами ассоциируются определенные действия программы, а со связями – данные (либо файлы данных), являющиеся их входными или выходными параметрами. Связи определяют зависимости между вычислениями на узлах. Считается, что система может исполнить все действия узла, если известны результаты всех входящих в узел связей. В интерфейс разработчика входят такие функции, как конструирование графа сценария (указание действий и данных), назначение критерия выбора ресурсов для исполнения конкретного узла, мониторинг состояния узлов в процессе исполнения, создание контрольных точек и перезапуск сценария.

1.2. Дополнительные возможности менеджера сценариев

Помимо стандартных средств работы с графом сценария, GridMD предоставляет разработчику ряд дополнительных возможностей, являющихся отличительными особенностями библиотеки. К ним относятся: использование процедур или участков кода клиентского приложения для автоматического создания элементов сценария, динамическое управление графом сценария, алгоритмические шаблоны для основных типов сценариев в задачах численного моделирования.

Обычно системы управления сценариями запускают задачи, являющиеся какими-либо командами или сторонними приложениями. Библиотека GridMD позволяет сформулировать вычислительную задачу в виде процедуры, описанной на языке C++ и встроенной в управляющее клиентское приложение. Такая процедура может быть объявлена узлом графа сценария, т.е. получать управление и данные в порядке, определяемом сценарием вычислений. При этом, если такая процедура исполняется удаленно, на используемом вычисли-

тельном ресурсе должна быть доступна копия клиентского приложения, скомпилированная для соответствующей операционной системы. При запуске такой копии (в данном случае в качестве расчетного приложения), требуемая встроенная процедура будет найдена и исполнена автоматически. В случае, когда встроенная процедура запускается на стороне клиента, переноса клиентского приложения на удаленные вычислительные системы не требуется. В приложении GridMD могут определяться действия, модифицирующие граф исполнения задачи. Таким образом, граф сценария GridMD является динамическим. Для корректного восстановления графа сценария при перезапуске модифицирующие действия должны быть привязаны к событиям, соответствующим изменению состояния графа – при этом последовательность динамических модификаций графа может быть однозначно восстановлена. Возможность контроля событий графа обеспечивается интерфейсом разработчика GridMD.

Алгоритмические шаблоны, или "алгоритмические скелетоны" (algorithmic skeletons) – термин, который применяется для обозначения высокоуровневых моделей программирования для параллельных или распределенных вычислений [7-9]. Существует стандартная классификация алгоритмических шаблонов, наиболее употребительные среди которых – последовательность (pipeline), ветвление (fork), отображение (map) и др. Преимущество алгоритмического шаблона заключается в том, что для использования реализуемого им алгоритма необходимо только создать его наполнение, т.е. указать те конкретные процедуры, которые будут исполняться на определенных этапах алгоритма. Все служебные действия, связанные с передачей данных, взаимодействием между параллельно запущенными процессами и т.д. выполняются автоматически и скрыты от пользователя алгоритмического шаблона.

1.3. Пример использования шаблона ветвления

Пример программы, основанной на применении шаблона fork, который реализован в виде класса gmFork, показан на Рис. 2, а соответствующий ей граф на Рис. 3.

Узлы, входящие в ветви шаблона ветвления классифицируются в определенном порядке: начальный узел start (точка расхождения ветвей), узел расщепления split (начальный узел каждой ветви), узел слияния merge (конечный узел каждой ветви), конечный узел шаблона finish (точка слияния ветвей). Между узлами split и merge может быть помещено произвольное число других узлов, в том числе допускается вложение шаблонов ветвления друг в друга. В этом случае шаблон нижнего уровня входит в ветвь шаблона более высокого уровня между узлами типа split и merge.

Помимо организации ветвей, шаблон gmFork позволяет привязать определенный тип данных к связям между узлами определенного типа. Такая привязка обеспечивает строгий контроль типов на уровне компилятора при использовании функций доступа к данным узла. Вместо использования общих функций node_input() и node_output(), в которых проверка соответствия типов передаваемых данных возможна только на этапе исполнения приложения, программисту предлагается пользоваться типизированными шаблонными функциями fork<type1, type2, type3>.vsplit_out(), fork<type1, type2, type3>.vsplit_in(), в которых типы используемых данных определяются шаблонными параметрами класса gmFork<> и известны на этапе компиляции.

```
begin_distributed();
// Define the skeleton and internal data link types
gmFork<void,val_t,void> fork1("loop");
fork1.begin_here();     // marks the loop 'begin' node

int nterms = 3;         // number of terms in the series
val_t sum = 0.;         // 'sum' accumulates the result
for(int i=0; i<nterms; i++){
  // Create a new 'split' node and define its output
  if(fork1.split())
    fork1.vsplit_out() = pow(x, (val_t)i) / i;

  // Define the action of the 'merge' node
  if(fork1.merge())
    sum += fork1.vmerge_in();  // accumulation of the terms
}
if(end_distributed()) // loop 'end' node is optional
  printf("The result is %g\n", sum);
```

Рис. 2. Программа для вычисления членов разложения в ряд функции ln(x) с использованием шаблона ветвления. Контроль типа результатов вычислений (val_t) осуществляется на этапе компиляции

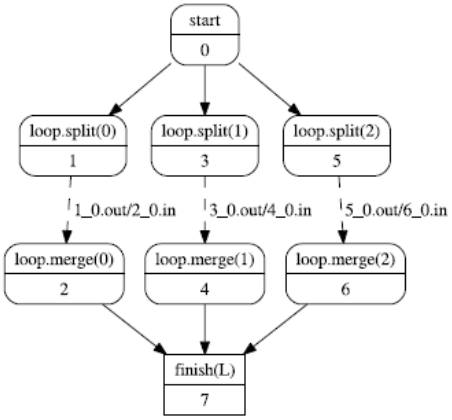

Рис. 3. Программа для вычисления членов разложения в ряд функции ln(x) с использованием шаблона ветвления. Контроль типа результатов вычислений (val_t) осуществляется на этапе компиляции

Отметим, что шаблон ветвления является исключительно интерфейсной конструкцией, упрощающей конструирования сценариев определенного типа. Исполнение узлов, сконструированных с помощью шаблона ветвления происходит с помощью универсальной процедуры обработки графа исполнения. Возможно ручное создание аналогичного сценария с использованием функций добавления узлов и связей. Такое ручное создание потребовало бы гораздо больше вызовов функций и увеличило бы вероятность программистской ошибки при конструировании сценария. На основе шаблона ветвления возможно создание более специализированных алгоритмических шаблонов, в которых роли отдельных компонент алгоритма (например, число ветвей, функции каждой ветви и т.д.) известны более детально.

1.4. Непосредственное использование менеджера заданий

Компонент GridMD, называемый “менеджером заданий” (Job Manager), отвечает за выполнение отдельного узла графа на локальной или удаленной машине. Под удаленной системой предполагается головная машина кластера или Грид-системы, на которую менеджер заданий должен скопировать входные данные, после чего передать задачу удаленной системе очередей, осуществить контроль за ее выполнением и получить результат. В обычном режиме внутренние функции GridMD обращаются к менеджеру заданий автоматически, однако программист может использовать его интерфейс напрямую или даже отдельно от остальных функций GridMD.

В настоящее время менеджер заданий поддерживает различные типы систем очередей для кластеров под управлением Portable Batch System (PBS), Simple Linux Utility for Resource Management (SLURM), ССРВ ИПМ РАН.

При инициализации менеджера заданий кроме способа запуска заданий необходимо указать также способ доступа к системе очередей, установленной на локальном или удаленном узле. Для удаленного доступа в настоящее время поддерживаются протокол SSH (реализовано с использованием кросс-платформенной библиотеки LibSSH). Если приложение GridMD выполняется по MS Windows, то

для доступа возможно использование внешних утилит plink и pscp из свободно распространяемого пакета PuTTY. Классы, реализующие протоколы доступа, предоставляют функции передачи и преобразования файлов, управления каталогами пользователя, выполнения отдельных команд на удаленной системе.

После инициализации пользовать или менеджер сценариев создает объекты класса gmJob, описывающие отдельную задачу, и передает их на выполнение тому или иному менеджеру заданий. Объект gmJob хранит информацию о входных и выходных файлах, командах, исполняемых на удаленной системе, требуемом числе процессоров и других ресурсов. Описание задачи не зависит от типа менеджера заданий, таким образом задача может быть передана на выполнение любому менеджеру, однако, после запуска в объекте gmJob сохраняется информация о том, где задача была запущена. Используя функции-члены класса gmJob пользователь может управлять заданием, копировать файлы во временный каталог задачи на удаленной системе и получать промежуточные/конечные файлы с результатами, определять текущее состояние задания.

Пример программы, иллюстрирующей основные функции создания и управления заданием показаны на Рис. 4. Инициализация задачи выполняется локально путем создания объекта gmJob (строка 2) и установки его атрибутов. В приведенном примере поле job.command содержит команду для выполнения, которая имеет различный синтаксис в зависимости от того, работает ли удаленная система под управлением ОС Linux (строки 4-7) или ОС Windows (строки 9-12). Далее функции AddInFile и AddOutFile в строках 14-18 задают набор входных и выходных файлов, которые будут загружены на удаленную системы перед выполнением и выгружены после успешного выполнения. Передача задачи на выполнение удаленной системе происходит в строке 21 при вызове функции Submit. В то время, как задача находится в очереди или выполняется, приложение GridMD может выполнять мониторинг ее состояния, выгружать промежуточные результаты и log-файлы, ожидать завершения (строки 26-34).

```
1   // Создание (инициализация задачи)
2   gmJob job;
3   job.command = gmdString(
4     "grep \"String 0\" data0.txt\n" )
5     + (script_type != WIN_SCRIPT ?
6       "sleep 4s; mkdir resdir; cat data0.txt >>resdir/res.txt\n"
7       "echo \"'String' 1\"\n"
8     :
9       "cscript /b sleep.vbs 4000\n"
10      "md resdir\n"
11      "copy data0.txt resdir\\res.txt >nul\n"
12      "echo 'String' 1\n"
13    );
14  job.AddInFile(indir + "/data0.txt", "", gmJob::TEXT);
15  if(script_type == WIN_SCRIPT) job.AddInFile(indir + "/sleep.vbs");
16  job.AddOutFile(outdir + "/res.txt", "resdir/res.txt", gmJob::TEXT);
17  job.AddOutFile(outdir + "/out.txt", "STDOUT");
18  job.AddOutFile(outdir + "/err.txt", "STDERR");
19
20  // Передача задачи на выполнение удаленной системе
21  state = job.Submit(*mngr, "batch-test", true);
22  if(state != JOB_SUBMITTED) return -1;
23  printf(" Job id = %s, subm_id = %s\n",
24    job.GetID().c_str(), job._subm_id().c_str() );
25
26  // Определение состояния задачи
27  printf("Job state is %s\n", job.GetStateStr());
28
29  // Копирование промежуточных результатов
30  job.StageOut(outdir + "/res1.txt", "resdir/res.txt", gmShell::TEXT);
31
32  // Ожидание завершения
33  puts("Waiting for the job to complete...");
34  state = job.Wait();
35
36  // Получение результатов
37  if(state == JOB_COMPLETED) state = job.FetchResult();
38
39  // Уничтожение задачи
40  job.Clear();
```

Рис. 4. Программа, иллюстрирующая основные функции работы с заданием

1.5. Производительность различных протоколов доступа к удаленной системе

Для программы, обсуждаемой в предыдущем разделе, были проведены исследования задержек, связанных с выполнением удаленных команд t_1 и копированием файлов между локальной и удаленной системой t_2. Результаты измерения времени выполнения удаленных команд и копирования файлов показаны в Таблица 1. Размер входного файла data0.txt и выходного res.txt составлял 10 байт, выходных файлов out.txt – 22 байта, err.txt – 0 байт. Таким образом,

размер данных был минимальным и основная задержка была обусловлена передачей служебной информации.

Таблица 1. Времена выполнения служебных и пользовательских операций для различных исполнителей

Тип протокола доступа/ОС локальной машины	Время выполнения команд на задачу (t_1), с	Время копирования файлов на задачу (t_2), с	Общее время служебных операций на задачу ($t_1 + t_2$), с	Полное время выполнения задачи, с
Система очередей PBS, кластер ОИВТ РАН(nwo5.ihed.ras.ru)				
Plink/Windows	4.06 ± 0.35	3.48 ± 0.26	7.54 ± 0.61	11.53
LibSSH/Windows	2.37 ± 0.25	1.91 ± 0.26	4.28 ± 0.51	11.29
SSH/Unix	1.85 ± 0.16	1.42 ± 0.08	3.27 ± 0.25	10.26
LibSSH/Unix	1.86 ± 0.18	1.41 ± 0.06	3.27 ± 0.24	10.25
Фоновые задачи bash shell, кластер ОИВТ РАН (nwo5.ihed.ras.ru)				
Plink/Windows	4.06 ± 0.48	3.45 ± 0.17	7.52 ± 0.64	11.49
LibSSH/Windows	1.93 ± 0.31	2.01 ± 0.16	3.94 ± 0.47	7.92
SSH/Unix	2.04 ± 0.17	1.56 ± 0.06	3.61 ± 0.23	7.59
LibSSH/Unix	1.55 ± 0.19	1.42 ± 0.06	2.97 ± 0.24	6.96
Фоновые задачи bash shell, кластер K100 ИПМ РАН (k100.kiam.ru)				
Plink/Windows	4.25 ± 0.85	4.71 ± 0.53	8.97 ± 1.38	13.58
LibSSH/Windows	0.69 ± 0.18	0.73 ± 0.02	1.41 ± 0.20	5.40
SSH/Unix	1.63 ± 0.39	1.31 ± 0.19	2.93 ± 0.58	6.92
LibSSH/Unix	0.69 ± 0.13	0.72 ± 0.06	1.42 ± 0.18	5.40

Тестовая задача выполнялась для каждой конфигурации 15 раз подряд, при этом на каждой итерации определялись времена t_1 и t_2, их сумма $t_1 + t_2$, а также полное время выполнения задачи. В конце теста для указанных величин определялись их среднее и стандартное отклонение (с коэффициентом 2), которое указано в таблицах в виде ошибки.

Тестирование производилось на рабочих станциях под управлением Windows и Linux, расположенных в ОИВТ РАН. В качестве удаленных систем использовались кластер ОИВТ РАН (nwo5.ihed.ras.ru, 83.149.226.17), находящийся в локальной сети

ОИВТ, а также внешний кластер К100 ИПМ РАН (k100.kiam.ru, 194.226.59.202). Время отклика для указанных серверов при использовании утилиты ping на 10 пакетах по 64 байта составляло (минимальное/среднее/максимальное):

от Windows-станции до nwo5.ihed.ras.ru: 1/2/6 мс;

от Linux-станции до nwo5.ihed.ras.ru: 0.149/0.215/0.303 мс;

от Windows-станции до k100.kiam.ru: 5/5/6 мс;

от Linux -станции до k100.kiam.ru: 4.773/5.260/6.345 мс.

На кластере ОИВТ РАН использовалась система очередей PBS с менеджером задач gmPBSManager и фоновое выполнение команд с менеджером gmBShManager, на кластере К100 – только фоновое выполнение команд. Из таблиц видно, что использование PBS увеличивает полное время выполнения задачи, однако слабо влияет на времена t_1 и t_2. Таким образом, время выполнения команд на удаленной системе не имело существенного значения и задержка определялась характеристиками сети и реализацией протокола доступа в GridMD.

2. Пример практического использования

Открытый код GridMD был использован компанией ООО «Кинтех Лаб.» для создания программного обеспечения в Российско-европейских проектах по моделированию нанокомпозитов (CompNanoComp) и органических светодиодов (IM3OLED) [10]. Обе платформы реализованы на одной функциональной основе и представляют собой программные комплексы, объединяющие несколько расчетных программ и предназначенные для численного моделирования в конкретной предметной области. Библиотека GridMD используется в обоих комплексах на низком уровне для управления сценариями расчетов. Под низким уровнем в данном случае понимается то, что конечный пользователь не имеет непосредственного доступа к элементам сценария GridMD (графу сценария и элементар-

ным исполняемым заданиям), а составляет сценарий расчета из более крупных блоков. Сами расчеты могут производиться как на удаленных вычислительных ресурсах (кластерах), так и на локальной машине в зависимости от конфигурации расчетных приложений.

Указанный опыт применения библиотеки позволил уточнить набор требований к программному интерфейсу и к протоколам работы с удаленными вычислителями, которые учтены в новой версии GridMD. В частности, это относится к инкапсуляции протокола SSH внутрь библиотеки, созданию групп заданий, динамическому добавлению эле-ментов сценария, реализации менеджера заданий для планировщика SLURM и другие. В целом, опыт использования GridMD показал, что библиотека повышает эффективность приложений такого рода, предъявляет минимальные требования к составу программного обеспечения на управляющей машине, обеспечивает оптимальное управление заданиями и передачей данных.

Заключение

Текст заключения

Благодарности. Работа выполнена при поддержке по программам фундаментальных исследований Президиума РАН №2 (коорд. чл.-к. Канель Г.И.), №14 (ак. Велихов Е.П., ак. Савин Г.И., ак. Жижченко А.Б.), ПФИ ПРАН по стратегическим направлениям развития науки №1 (коорд. ак. Бетелин В.Б.). Расчеты проведены на кластерах “К-100” (ИМП РАН) и в МСЦ РАН.

Список литературы

[1] I.V. Morozov, I.A. Valuev. *Automatic Distributed Workflow Generation with GridMD Library* // Computer Physics Communications. 2011. Vol. 182. p. 2052–2058.

[2] Программа GridMD в открытом доступе, URL: http://gridmd.sourceforge.net

[3] W.M.P. van der Aalst. *The application of Petri nets to workflow management* // The Journal of Circuits, Systems and Computers. 1998. Vol. 8. No. 1. p. 21–66.

[4] Pytlinski, J., Skorwider, L., Benedyczak, K., et al. *Uniform access to the distributed resources for the computational chemistry using UNICORE* // Computational Science — ICCS 2003. Springer Berlin Heidelberg, 2003. — p. 307-315.

[5] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — М.: БХВ-Санкт-Петербург, 2004. — 608 с.

[6] Карпов В.Е., Лобанов А.И. Численные методы, алгоритмы и программы. Введение в распараллеливание. — М.: Физматкнига, 2014. — 196 с.

[7] H. González-Vélez, M. Leyton. *A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers* // Software: Practice and Experience. 2010. Vol. 40. Issue 12. p. 1135–1160.

[8] eSkel project, URL: http://homepages.inf.ed.ac.uk/abenoit1/eSkel

[9] Skandium project, URL: http://skandium.niclabs.cl

[10] Integrated Multidisciplinary & Multiscale Modeling for Organic Light-Emitting Diodes, URL: http:// www.im3oled.eu

Об авторе:

Морозов Игорь Владимирович

Кандидат физико-математических наук, заведующий лабораторией ОИВТ РАН, доцент кафедры информатики МФТИ, доцент кафедры прикладной математики Высшей школы экономики, лауреат медали РАН с премией для молодых учёных России 2004 г., лауреат гранта Президента РФ для молодых учёных-кандидатов наук 2010 г.

e-mail: morozov@ihed.ras.ru

Валуев Илья Александрович

Кандидат физико-математических наук, старший научный сотрудник ОИВТ РАН

e-mail: valuev@ihed.ras.ru

Образец ссылки на публикацию:

И. А. Валуев, И. В. Морозов. GridMD: компактная переносимая библиотека C++ для управления распределенными вычислениями // Программные системы: теория и приложения: электрон. научн. журн. 2013. Т. 4, № 3(17), с. ??–??.
URL: http://psta.psiras.ru/read/???

I. A. Valuev, I. V. Morozov. GridMD: compact portable C++ library for managing distributed simulations.

ABSTRACT. Architecture, functionality and use cases of an open source GridMD library are discussed. The library is intended to facilitate development of distributed applications for contemporary Grid systems and for task management systems of supercomputing clusters. Initially the library was targeted at developers of molecular dynamics simulation codes although later its application area was extended significantly. Nowadays it can serve as a universal tool for creation of distributed computing applications as well as for task management programs based on a compact client-side C++ code. In the first place the GridMD library is required for creating the applications that contain many computation stages with possible data links as such applications that can be efficiently run in the distributed environment.

Key Words and Phrases: distributed computing, Grid technologies, execution graph, molecular dynamics, multiscale simulations.