ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 2(11), 2012, с. 23–49

УДК 004.424, 004.434

Ю. А. Климов, А. Ю. Орлов, А. Б. Шворин

# Программный инструментарий для трафаретных вычислений на гибридных суперкомпьютерах

Аннотация. Рассматривается проблема переноса программ на гибридные суперкомпьютеры. В общем случае эта проблема сложна и требует вложения значительного количества высококвалифицированного труда, однако для ограниченных классов программ перенос всё же поддается автоматизации. Одним из таких классов являются трафаретные программы, имеющие широкое применение в научных вычислениях.

Излагаются идеи и описывается реализация программного инструментария, разработанного авторами в рамках проекта Кентавр, направленного на автоматическое распараллеливание трафаретных программ для исполнения на гибридных суперкомпьютерах. Основная задача, решаемая инструментарием Кентавр, — автоматизация обменов между вычислительными узлами суперкомпьютера, а также между ускорителем и центральным процессором в рамках узла.

*Ключевые слова и фразы:* трафаретные вычисления, гибридные суперкомпьютеры, проблемно-ориентированные языки программирования, проект Кентавр.

## Введение

В последние пару лет появилось множество гибридных суперкомпьютеров — машин с графическими ускорителями (Tianhe-1A, Tsubame 2.0, Ломоносов, K-100), и такие машины будут появляться еще (Titan, Bluewaters). Чтобы в полной мере воспользоваться вычислительной мощностью подобных суперкомпьютеров, программы, разработанные для последовательного исполнения или исполнения на кластере без ускорителей, необходимо переписывать (заметим в скобках, что такая ситуация — необходимость переписывания программного кода под вновь появившуюся аппаратуру — возникает в отрасли далеко не в первый раз).

Работа выполнена при поддержке Министерства образования и науки Российской Федерации (госконтракт № 07.514.11.4014).

Разрабатывать программы для суперкомпьютеров с ускорителями вычислений значительно сложнее, чем для «классических» суперкомпьютеров. И графические ускорители — не исключение. Хотя само по себе программирование ускорителя делается на почти привычном языке (и CUDA [1], и OpenCL [2] основаны на Си), для эффективной работы скорее всего потребуется организация данных отличная от той, что использовалась ранее. Кроме этого, ускорители добавляют новые явные уровни в иерархию памяти современных суперкомпьютеров, которые приходится учитывать при реализации.

Существует класс программ, для которых можно достаточно легко и естественно специфицировать всю необходимую информацию, чтобы распараллеливание программы выполнялось автоматически, а именно — так называемые *трафаретные программы* (stencil codes). К ним относятся многие алгоритмы, применяемые в научных расчетах, например, при решении дифференциальных уравнений методом конечных разностей в различных задачах вычислительной механики. В статье описывается разрабатываемый авторами набор программных инструментов Кентавр, позволяющий автоматизировать разработку параллельной версии трафаретной программы для гибридных суперкомпьютеров, обеспечивая все необходимые обмены данными между памятью ускорителя и системной памятью, а также между вычислительными узлами суперкомпьютера, и не требуя их спецификации в программе в явном виде.

Сразу оговоримся, что данная статья в частности и проект Кентавр [3] в целом не направлены на создание программного кода и организацию вычислений, обеспечивающих максимально эффективную работу (высокий КПД) отдельных вычислительных устройств — как классических процессоров, так и графических ускорителей. Это отдельные задачи, которые для трафаретных вычислений могут быть эффективно разрешены [4]. Проект Кентавр ориентирован на быструю разработку достаточно эффективных параллельных программ для гибридных суперкомпьютеров на основе последовательных программ, автоматизируя необходимые обмены и избавляя тем самым программиста от трудоемкой и кропотливой работы [5].

Кентавр, в отличие от многих других проектов со схожими целями, использует идеологию, в чем-то близкую к системе OpenMP [6]. При использовании Кентавра программист не управляет распределением данных и вычислений ни между узлами вычислительной системы, ни между процессорами и ускорителями вычислений в рамках

одного узла. Ему необходимо лишь декларативно задать для распараллеливаемых циклов *трафарет* — описание зависимостей по данным, — а система сама автоматически распределит данные и обеспечит пересылки и синхронизацию.

Необходимо отметить, что под близким девизом разрабатывалась система Cluster OpenMP для классических суперкомпьютеров [7], однако она не получила сколько-нибудь заметного распространения и в настоящее время уже не поддерживается. При использовании Cluster OpenMP программист, действительно, не задает распределение и передачи данных, он лишь указывает (как и в OpenMP), что витки циклов независимы — то есть могут выполняться параллельно. Однако этого недостаточно, и для достижения приемлемого результата программист должен представлять, как именно данные распределятся по узлам суперкомпьютера, и какие возникнут передачи между ними в процессе работы. Таким образом, программисту всё равно приходится думать в терминах параллельно работающих процессов и обменов между ними, но на уровне языка Cluster OpenMP нет средств для выражения этих понятий. Однако если программист понимает, как требуется распределить данные и организовать обмены, то ему под силу реализовать параллельный алгоритм на более распространенных параллельных системах (например, MPI [8] или UPC [9]), возможно, с существенными оптимизациями. Таким образом, оказывается, на практике не существует ниши для применения Cluster OpenMP. Одной из причин данного недостатка является универсальность этого инструмента — ориентация на автоматический перенос *любых* программ, записанных при помощи OpenMP, на суперкомпьютер.

В отличие от Cluster OpenMP, проект Кентавр направлен на более узкий класс программ — на трафаретные программы и, в частности, на сеточные методы, но на более широкий класс аппаратуры — на суперкомпьютеры как с ускорителями вычислений, так и без них. Для использования Кентавра программа должна быть изначально написана или преобразована к необходимому виду по разработанной авторами методике [10]. В методике описаны преобразования в терминах исходной последовательной программы, причем, что немаловажно, без привязки к параллельной реализации. Следует отметить, что описанные преобразования необходимы как для распараллеливания программы на многоузловую систему, так и для переноса на графические ускорители. Оказывается, требования, предъявляемые этими направлениями, достаточно близки.

Дальнейшее изложение построено по следующей схеме. В части 1 определяется класс алгоритмов, на работу с которыми ориентирован Кентавр, — трафаретные вычисления. В части 2 дается описание инструментария Кентавра на идейном уровне и приводится сравнение с близкими проектами. В части 3 рассказывается о внутреннем устройстве инструментария.

## 1. Трафаретные вычисления

Многие прикладные задачи устроены таким образом, что доступ к данным на чтение и на запись происходит в них регулярным образом (регулярным с точки зрения самой задачи, а не конкретной реализации!). Выделение и формализация такого рода шаблона доступа во многих случаях позволяет упростить извлечение параллелизма и построить эффективную реализацию для требуемой аппаратуры. Разумеется, в общем случае бывает непросто выделить такой шаблон и, тем более, реализовать эффективный параллельный алгоритм.

Существует широкий класс задач, в которых имеется пространство ячеек, над которым пошагово проводятся вычисления таким образом, что новое состояние данной ячейки на текущем шаге зависит от состояний некоторого подмножества ячеек на предыдущем шаге. Сюда входят, например, сеточные задачи вычислительной механики, где состояние определяется значениями в элементах (узлах, ребрах, ячейках) сетки.

Важный подкласс образуют так называемые *трафаретные вычисления*. Его особенностью является то, что множество ячеек, от которых зависит состояние данной ячейки, устроено следующим регулярным образом. Пусть пространство представлено многомерным массивом, и состояние данной ячейки массива на текущем шаге зависит только от состояний «соседних» ячеек на предыдущем шаге. Понятие соседства определяется так: ячейка $V_i$ является соседней по отношению к ячейке $U$, если ее координаты (индексы в массиве) получаются сдвигом координат $U$ на один из векторов $t_i$ из заданного набора $T$. Этот набор векторов $T = \{t_1, \ldots, t_n\}$ называется *трафаретом* и должен быть задан заранее. *Трафаретные вычисления* заключаются в том, что для каждой ячейки ее новое состояние определяется как функция от (старых) состояний ее соседей. Отметим, что вычисления для каждой ячейки происходят независимо от вычислений для других ячеек.

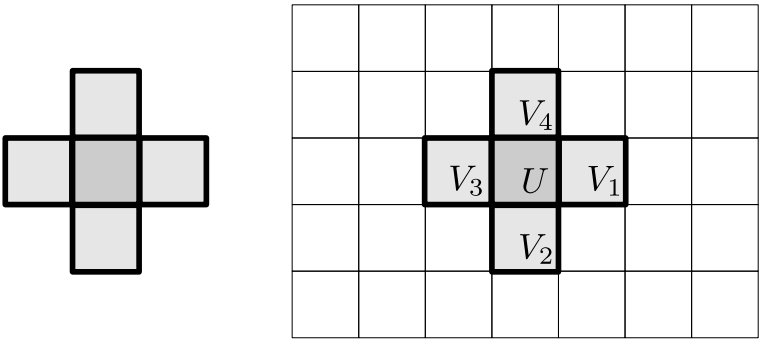


Рис. 1. Трафарет $\{(\pm 1, 0), (0, \pm 1)\}$ (слева) и его наложение на ячейку $U$ двумерного массива (справа)

Например, на рис. 1 изображен трафарет, состоящий из векторов $(\pm 1, 0), (0, \pm 1)$, и его наложение на одну из ячеек массива.

В общем случае задача может содержать несколько трафаретов, применяемых независимо друг от друга. Кроме того, реальные задачи не обязательно целиком состоят из трафаретных вычислений, а могут, в частности, содержать операцию редукции. Редукция не является трафаретной операцией, поскольку для нее нельзя заранее выбрать трафарет: он должен был бы охватывать всю сетку, однако, согласно определению, трафарет должен быть зафиксирован до того, как будут выбраны параметры сетки.

Оказывается, что при помощи только лишь последовательно применяемых трафаретов и операций редукции можно записать достаточно представительный класс практических задач.

### 1.1. MapReduce

В качестве простого, но достаточно наглядного примера вычислительного шаблона можно упомянуть модель вычислений MapReduce [11]. Процесс вычисления в этой модели состоит из двух фаз: Map — независимая обработка большого числа небольших подзадач (например, обработка элементов массивов) и Reduce — сбор и обработка всех результатов подзадач (например, редукция всех элементов массива).

Операция Map является вырожденным примером трафаретного вычисления: трафарет состоит из единственного элемента $(0, \ldots, 0)$.

```
/* Шаги по итерациям */
for (s = 0; s < S; s++) {
    /* Трафаретный цикл.
     * Трафарет: {(+1,0), (-1,0), (0,+1), (0,-1)}. */
    for (x = 1; x < (X-1); x++) {
        for (y = 1; y < (Y-1); y++) {
            a_new[x][y] = (a[x+1][y]+a[x-1][y]
                           +a[x][y+1]+a[x][y-1])/4;
        }
    }

    /* Трафаретный цикл. Трафарет: {(0,0)}. */
    for (x = 1; x < (X-1); x++) {
        for (y = 1; y < (Y-1); y++) {
            a[x][y] = a_new[x][y];
        }
    }
}
```

Рис. 2. Ядро вычислений метода Якоби

При реализации модели MapReduce на параллельной машине накладные коммуникационные расходы на операцию Map можно считать равными нулю (если не учитывать начальное распределение данных), поскольку она выполняется локально. Остаются лишь коммуникации, обусловленные редукцией. Очевидным достоинством этой модели вычислений является простота эффективной реализации базовых операций — Map и Reduce — на многоузловой аппаратуре.

Тем не менее применение модели MapReduce существенно ограничено вследствие того, что далеко не всякая задача может быть эффективно выражена композицией таких простых операций как Map и Reduce. Для решения более широкого класса задач может понадобиться обобщение подхода MapReduce, и трафаретные вычисления (в широком смысле — включая редукцию) как раз являются таким обобщением.

### 1.2. Метод Якоби для задачи теплопроводности

Рассмотрим более интересный пример. Для решения задачи теплопроводности и некоторых других задач вычислительной механики может применяться метод Якоби. Ядро вычислений для решения уравнения теплопроводности в двумерном случае показано на рис. 2.

Здесь заданы массивы `a[X][Y]` и `a_new[X][Y]`, в которых хранится состояние системы на предыдущем временно́м шаге и текущем соответственно. Элементарное вычисление устроено таким образом, что новое значение в каждой ячейке зависит от значений в смежных ячейках — трафарет первого цикла представляет собой набор из четырех элементов $\{(\pm 1, 0), (0, \pm 1)\}$ (на рис. 1 изображен именно этот случай), а трафарет второго — $\{(0, 0)\}$.

### 1.3. Обобщение трафаретных вычислений

Многие сеточные методы на регулярных неадаптивных сетках также являются трафаретными по сути, но не по данному выше определению, поскольку их сетки не обязательно представлены многомерными массивами. Поэтому позволим себе трактовать понятие трафарета и трафаретных вычислений более общо. А именно — не будем ограничиваться многомерным массивом как представлением пространства. Пусть оно по-прежнему состоит из элементов (ячеек) и обладает некоторой заданной структурой. И пусть имеется несколько преобразований — *трансляций*[1] — пространства в себя, определенных, возможно, не на всем пространстве. Тогда, согласно более общему определению, трафаретом будет называться набор трансляций, а соответствующее этому трафарету вычисление заключается в том, что новое состояние ячейки $U$ вычисляется как функция от состояния ячеек, получающихся при применении к $U$ трансляций из трафарета. На рис. 3 приведен пример трафарета, определенного не на двумерном массиве, а на пространстве несколько более общего вида.

Существует ряд методик программирования в рамках формализма трафаретных вычислений (например, [4, 12–14]). Предлагаемая ниже технология действует в том же русле, а именно — она предназначена для адаптации близкого к трафаретным вычислениям класса задач к исполнению на гибридном суперкомпьютере.

[1] В математике трансляция означает параллельный перенос, то есть сдвиг пространства вдоль вектора. Здесь этот термин употреблен в более широком смысле.

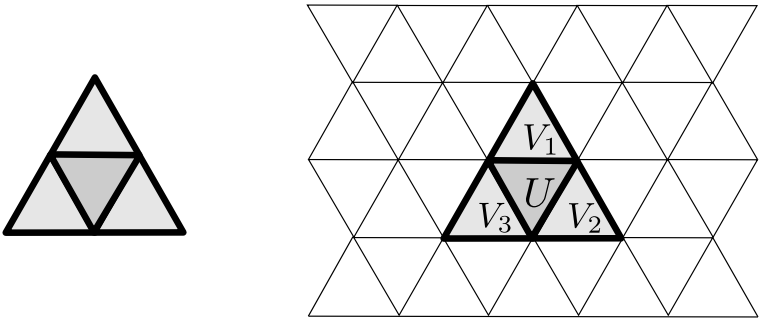


Рис. 3. Пример трафарета на пространстве более общего вида

## 2. Программный инструментарий Кентавр

Проект Кентавр ставит своей целью разработку инструментария для быстрого создания параллельных приложений, предназначенных для запуска на гибридных суперкомпьютерах — многоузловых установках, в состав которых входят ускорители вычислений, прежде всего графические.

Ключевая сложность программирования гибридных суперкомпьютеров, с нашей точки зрения, — это не дополнительный язык для программирования ускорителей, а наличие нескольких дополнительных уровней иерархии памяти, из-за которого сложность задания корректного и эффективного распределения и перемещения данных достигает критического уровня.

В связи с этим Кентавр направлен на разработку инструментария, скрывающего от программиста множество уровней памяти и обеспечивающего автоматическое распределение и перемещение данных между ними. Осознавая, что разработка универсального инструмента крайне сложна, если вообще возможна, мы сосредоточились на подклассе, который составляют трафаретные и близкие к ним задачи. Использование заданного программистом трафарета позволяет автоматически распределить данные между узлами суперкомпьютера и выполнить трафаретные циклы параллельно на узлах суперкомпьютера с использованием графических ускорителей.

## 2.1. Близкие проекты

В эпоху суперкомпьютеров, состоящих из классических процессоров, выгода от использования специального инструмента для распараллеливания трафаретных программ была не очень велика — для квалифицированного специалиста преобразование последовательной трафаретной программы в параллельную с использованием MPI является чисто технической типовой задачей. При этом можно ожидать, что специализированный инструмент будет иметь ряд недостатков, связанных в первую очередь с доступностью, переносимостью и, самое главное, взаимодействием с более универсальными средствами (как MPI) в тех случаях, когда возможностей инструмента оказывается недостаточно. Несмотря на это, проекты по распараллеливанию трафаретных программ все же имеют некоторую историю. По-видимому, самым успешным из них является библиотека LibGeoDecomp [12], основанная на шаблонах C++.

Однако сложность программирования суперкомпьютеров с ускорителями заставляет пересмотреть имеющийся арсенал средств и методов и предусмотреть автоматизацию там, где раньше это было нецелесообразно. Помимо обсуждаемого в данной работе инструмента Кентавр, за последний год появилось еще по крайней мере два проекта, нацеленных на распараллеливание трафаретных программ на гибридные суперкомпьютеры:

(1) был реализован соответствующий back-end в рамках уже упомянутого проекта LibGeoDecomp;
(2) для использования на суперкомпьютере Tsubame 2.0 был запущен проект Physis [13].

Также хочется отметить интересный проект Microsoft Research Accelerator [15], который, хотя и не имеет пока выхода на гибридные суперкомпьютеры, ставит своей целью строить эффективные программы для совершенно различных типов аппаратуры, исходя из высокоуровневого описания трафаретных программ. В данной работе не будут обсуждаться сравнительные характеристики этих проектов в силу того, что все эти проекты еще молоды и относительно незрелы. На данный момент нам достаточно показать, что тема вызывает горячий интерес в мире, и продемонстрировать возможности нашего подхода. Скажем только, что отличительной особенностью подхода Кентавра является отсутствие принципиальных ограничений на

конфигурации сеток, возможность работы с неструктурированными, многоблочными и адаптивными сетками.

В качестве схожих средств программирования ускорителей упомянем OpenACC [16] и StarPU [17].

Стандарт OpenACC, анонсированный осенью 2011 года, предназначен для упрощения использования ускорителей вычислений по сравнению со специальными системами программирования, такими как CUDA или OpenCL. По аналогии с OpenMP, OpenACC предоставляет возможность давать указания о том, как должна исполняться программа, при помощи аннотаций к коду на языках Си или Фортран. В отличие от Кентавра, OpenACC предназначен для исполнения программ на одном узле (но, возможно, с несколькими ускорителями), причем для эффективного исполнения программисту необходимо описывать передачи данных между памятью и ускорителем вычислений.

Система StarPU предназначена для исполнения на гибридном вычислительном узле, одновременно задействуя все вычислительные ресурсы: и процессоры, и ускорители вычислений, причем система автоматически заругружает все доступные ресурсы. Ключевым понятием системы является Codelet — часть программы, которая может исполняться независимо. Ее описание содержит все необходимые для исполнения данные и один или несколько исходных кодов для исполнения на разных архитектурах. Существует расширение StarPU, позволяющее задействовать и несколько вычислительных узлов.

Также необходимо упомянуть проект PETSc [14], который представляет собой набор структур и подпрограмм для параллельного программирования научных приложений. PETSc включает понятие распределенного массива с теневыми гранями, который удобно использовать для реализации трафаретных вычислений — не нужно явно описывать обмены данными между узлами, достаточно сделать специальный вызов. В отличие от Кентавра, PETSc ориентирован на классические суперкомпьютеры без ускорителей. Также в PETSc программисту необходимо явно распределить данные между узлами и выполнять обмены.

## 2.2. Идеология Кентавра

В основу реализации Кентавра лег подход незначительной модификации исходного кода для распараллеливания на множество узлов с ускорителями. Причем ключевым моментом является то, что в

типичных ситуациях программист не задает распределение данных по вычислительным устройствам и вообще не думает о своей программе как о параллельной. Он лишь специфицирует те действия в своем алгоритме, которые могут исполняться независимо. Данный подход очень успешно применяется для систем с общей памятью, ярким представителем которых является OpenMP. Однако для машин с распределенной памятью подход «данные идут за кодом» — когда распределение вычислений между узлами автоматически определяет распределение данных между узлами — часто оказывался неуспешным (например, Intel Cluster OpenMP). Ключевой проблемой таких универсальных проектов является то, что заранее, например во время компиляции программы, неизвестно, какие данные будут использованы и как. Отметим, что во всех более или менее успешных проектах для машин с распределенной памятью и распределение данных, и распределение вычислений описывает программист.

Область применения Кентавра ограничена классом трафаретных программ, про которые заранее известно, как они обращаются к данным. Это знание позволяет автоматически распределить и данные, и вычисления между узлами, организовав эффективные обмены. При этом, как будет показано ниже, сам код ключевых циклов не приходится изменять: ровно тот же самый код циклов, который использовался в последовательном варианте, может быть эффективно использован и в параллельном. В нем, в частности, не появляется явных обращений к коммуникационным подсистемам. Текущая версия инструментария Кентавр разработана для программ, производящих вычисления на сетках. На текущий момент поддерживаются неадаптивные структурированные сетки для области произвольной формы. Однако разработанная методика позволяет реализовать аналогичные инструменты для неструктурированных, многоблочных и/или адаптивных сеток, что планируется сделать в дальнейшем.

Для управления распределением и перемещением данных Кентавр должен полностью контролировать сетку. После построения сетки в программе или чтения сетки из файла программист должен передать ее в библиотеку, и там будет выбрано распределение данных. Далее, в процессе задания значений, связанных с элементами сетки (вершинами, ячейками, ребрами, границами), система Кентавр автоматически заводит соответствующие распределенные массивы. Затем, при выполнении трафаретного шага (применения трафарета ко всем элементам массива), система автоматически распараллеливает

данную работу согласно построенному распределению данных. Отметим, что хотя распределение данных происходит автоматически и неявно, программист или оператор, запускающий задачу на суперкомпьютере, могут давать подсказки или автоматизированно производить распределение сетки. При этом код программы и ее корректность меняться не будут. В чем-то это аналогично управлению распределением витков циклов в OpenMP или Charm-объектов в системе Charm++ [18]. Имея распределение данных, система уже в полностью автоматическом режиме (корректно, независимо от распределения) заводит теневые грани и осуществляет обмены между узлами и внутри узлов, между системной памятью и памятью ускорителей.

Наконец, важной особенностью модели программирования Кентавр является тот факт, что она опирается на декларативную спецификацию свойств программы. Модель ориентирована на класс программ, для которых, при условии задания всех трафаретов, информация о зависимостях между данными является полной. Отсюда следуют два вывода:

(1) программист работает в понятиях предметной области задачи, специфицируя общие свойства своего алгоритма, не опираясь и не привязываясь к каким-либо свойствам аппаратуры;

(2) заданной информации о программе должно быть достаточно, чтобы автоматически преобразовать ее для исполнения не только на современных гибридных суперкомпьютерах, но и на новых классах аппаратуры, которые будут появляться в дальнейшем.

### 2.3. Как выглядит программа с точки зрения прикладного программиста

Библиотеки, созданные по методике проекта Кентавр, позволяют прикладному программисту работать со своей программой как с последовательной. Это, однако, не означает, что произвольная последовательная программа будет автоматически распараллелена. На программу накладываются достаточно жесткие ограничения и, вообще говоря, в текст имеющейся последовательной программы необходимо внести изменения для того, чтобы в дальнейшем она была распараллелена. Впрочем, эти изменения касаются только свойств самой программы и не затрагивают никаких возможных свойств аппаратуры, на которой будет происходить исполнение (включая какие-либо упоминания параллелизма). Таким образом, надо полагать, что прикладной программист, понимающий свойства своего алгоритма,

должен достаточно легко справиться с адаптацией своей программы под данную технологию.

Система Кентавр предоставляет программисту несколько ключевых понятий, которые необходимо использовать в программе для успешного распараллеливания.

Первым таким понятием является *сетка* — набор структур, описывающих множество элементов (вершин, ребер, ячеек и т. п.), со значениями в которых будут проводиться вычисления. Кентавр предоставляет структуры для описания, а пользователь должен с их помощью описать используемую сетку и передать ее системе. В описание входят массивы элементов сетки, такие как: массив (всех) вершин, массивы ребер (разные массивы для ребер вдоль разных направлений), массив ячеек. Могут также включаться массивы подмножеств, например, массив граничных для заданной области вершин.

Следующим ключевым понятием, с которым сталкивается программист, являются *массивы значений*, связанных с элементами сетки (вершинами, ребрами, ячейками). Для хранения этих значений необходимо использовать массивы, полученные специальным библиотечным вызовом. Эти массивы в Кентавре представлены обычными указателями языка Си типа `void *`.

Третьим ключевым понятием является *трафаретный цикл* с прилагаемым к нему описанием трафарета. Цикл должен проходить по элементам сетки с некоторыми ограничениями. А именно — внутри цикла не допускаются обращения к массивам описания сетки и массивам значений по произвольному индексу. Допускаются обращения лишь по следующим индексам:

(1) по переменной цикла;
(2) по значениям, прочитанным из массивов описания сетки (по разрешенным индексам) на текущем шаге.

Например, трафаретный цикл метода Якоби следует записать как показано на рис. 4, где:

- `grid` — описание сетки;
- `grid.N` — общее количество узлов сетки;
- `grid.node` — массив описаний узлов сетки.

Каждое описание узла сетки содержит пять полей:

- `boundary` — флаг, показывающий, что узел граничный;
- `xp`, `xm`, `yp`, `ym` — поля, в которых хранятся номера соседних узлов в направлениях $(+1, 0)$, $(-1, 0)$, $(0, +1)$, $(0, -1)$ соответственно.

```
#pragma centaur cross(1;a_new;a)
/* Обход всех узлов */
for (n = 0; n < grid.N; n++) {
    /* Проверка, что узел внутренний */
    if (grid.node[n].boundary == -1) {
        a_new[n] = (a[grid.node[n].xp]+a[grid.node[n].xm]+
                    a[grid.node[n].yp]+a[grid.node[n].ym])/4;
    }
}
```

Рис. 4. Трафаретный цикл

Хотя ограничения на тело трафаретного цикла достаточно жесткие, многие программы им удовлетворяют или могут быть относительно просто переписаны под данные ограничения. Как будет рассказано в разделе 3, эти ограничения связаны как с распараллеливанием задачи между узлами суперкомпьютера, так и с исполнением на графических ускорителях.

В приведенном выше примере описание трафарета выглядит как `cross(1;a_new;a)`, что задает трафарет в форме креста размером 1, показанный на рис. 1 (т. е. трафарет состоит из соседних в направлении осей $X$ и $Y$ элементов; их номера хранятся в полях `xp`, `xm`, `yp`, `ym`). Указанные массивы `a_new` и `a` используются для записи и для чтения соответственно. Подчеркнем, что трафарет задает используемые элементы в терминах структур, описывающих сетку.

Отметим, что для разных проектов, возможно, необходимо использовать различные структуры для описания сетки и описания трафаретов. В вышеприведенном примере сетка была двумерная, и описание узла содержало пять полей. В других задачах может потребоваться другая размерность сетки, другие ее элементы, другие поля в описании элементов. В каждом случае необходимо создать свои проблемно-ориентированные реализации предлагаемого нами подхода. При необходимости язык описания трафаретов также необходимо изменять или расширять, чтобы на нем могли быть выражены трафареты, удовлетворяющие требованиям предметной области.

И, наконец, последним ключевым понятием, без которого также невозможно создание сколь-нибудь нетривиальных программ, является *редукция*. В системе Кентавр редукция реализована несколькими библиотечными вызовами; в качестве параметров указывается функция, с которой производится свертка (в текущей реализации поддерживаются функции вычисления максимума и минимума, а также сложение и умножение чисел с плавающей точкой), и редуцируемый массив. При реализации редукции для других предметно-ориентированных областей набор функций может быть расширен.

## 3. Принципы реализации

В данном разделе описаны основные принципы реализации инструментария Кентавр. На основе описанных принципов (или близких к ним) разработчики могут создавать аналогичные Кентавру наборы инструментов для параллельных вычислений в различных предметно-ориентированных областях. Снова отметим, что прикладному программисту, использующему инструментарий, не требуется ни знать эти детали реализации, ни иметь какие-либо представления о параллельном исполнении программы.

### 3.1. Организация вычислений: стадии

Стадия — это участок программы (как правило, цикл), для которого заранее указано, какие данные и каким образом используются при исполнении данного участка. В рамках предлагаемой методики зависимости по данным могут указываться пользователем посредством трафарета, что является основным способом декларации стадий. Так, трафаретный цикл в исходной программе сам по себе является стадией с точки зрения реализации инструментария Кентавр. При этом зависимости по данным разрешаются средствами библиотеки Кентавр автоматически путем генерации вызовов нижележащих библиотек: MPI для межузловых обменов и CUDA для копирования данных между памятью ускорителя и системной памятью.

Схематично процесс исполнения стадии можно представить состоящим из двух фаз: фазы вычислений и фазы коммуникаций. Фаза вычисления задается явным образом — это собственно кусок кода, объявленный пользователем как стадия; фаза же коммуникаций описывается неявно — пользователь лишь указывает правила обращения

с данными (в зависимости от реализации инструментария это выражается с помощью синтаксических средств: макросов, псевдокомментариев, прагм[2] или даже выясняется автоматически анализатором кода), а библиотека по этим указаниям сама строит необходимый коммуникационный код. Впрочем, такое разделение на фазы условно: реализации уровня библиотек могут допускать совмещение счета и коммуникаций, но, поскольку система гарантирует удовлетворение зависимостей по данным, допустимо считать, что фазы разнесены во времени.

Поскольку основная цель нашей методики — перенести основную тяжесть вычислений на ускорители и распараллелить программу, подразумевается, что стадия может исполняться, во-первых, на ускорителях и, во-вторых, параллельно на многих узлах. Однако в общем случае не требуется, чтобы стадия обладала этими свойствами. Разработчикам проблемно-ориентированного инструментария, опирающегося на созданные в рамках Кентавра низкоуровневые библиотеки, может быть удобным выделение в виде стадии участка кода, который распараллеливать и переносить на ускорители затруднительно или невыгодно. Это может помочь элегантно выразить синхронизацию по данным с помощью внутренних библиотечных инструментов. В качестве примера можно привести операцию редукции: если оформить такой вызов в виде стадии, то подсистема времени исполнения автоматически проведет синхронизацию массива данных, по которому проводится редукция.

Другой особенностью стадии является то, что, в общем случае, можно выбирать устройство (центральный процессор, ускоритель) для выполнения стадии в динамике: вся необходимая для этого информация присутствует в описании стадии. Более того, допускаются реализации, в которых стадия может одновременно исполняться и на ускорителях, и на CPU, однако для того, чтобы это было оправданным, потребуется балансировка нагрузки между устройствами различных типов. Таким образом, к стадии может быть приписано несколько вариантов исходного кода для различных архитектур. В этом свете стадия аналогична понятию Codelet в системе

[2]Здесь прагма — это способ передать некое указание компилятору с помощью декларации `#pragma`. Набор поддерживаемых прагм не стандартизован в языке Си и варьируется для каждого компилятора от версии к версии. Этот механизм часто используется для быстрого добавления функциональности компилятору.

StarPU [17]. Ниже в разделе 3.4 будет показано, как с минимальными усилиями получить версию кода, предназначенную для графических ускорителей.

## 3.2. Распределение данных и обмены между узлами

Рассмотрим типичную трафаретную программу. В ней задана сетка и набор массивов значений по элементам сетки (например, по вершинам, ребрам, граням) или их подмножествам (например, по граничным вершинам). Массивы значений обычно обрабатываются в циклах по всем элементам.

Для распараллеливания задачи на несколько узлов суперкомпьютера будем использовать классический способ — разобьем сетку на области (для минимизации передач желательно с минимальным периметром, впрочем, это не влияет на корректность метода). Впоследствии каждая область будет обрабатываться в отдельном параллельном процессе. И в соответствии с этим разбиением распределим массивы значений.

Трафаретные циклы могут использовать нетривиальные трафареты: для вычисления нового значения в некоторой ячейке необходимы значения в соседних ячейках. Для эффективного обращения к таким данным заведем теневые грани — копии данных, распределенных на другие узлы, но необходимых для вычислений на данном узле.

При реализации вышеописанной процедуры будем использовать способ, показанный в [19]. Этот способ основан на изменении массивов описания сетки при распараллеливании: каждый узел хранит описание только своей части сетки с теневыми гранями и соответствующие части массивов значений. При таком подходе исходный код циклов из последовательной версии программы может быть без изменений использован и в параллельной версии. Однако это возможно лишь при том условии, что обращения к массивам-описаниям и массивам-значениям происходят только по индексам, полученным из описания сетки. В этом случае все элементы будут иметь новые номера — локальные для данного процесса. Чтобы обращения к элементам массивов были корректны, эти номера должны быть изменены при распараллеливании. Из этого факта вытекают ограничения на тела трафаретных циклов, о которых говорилось выше.

Итак, Кентавр должен управлять описанием сетки и массивами значений. Поэтому программист, создавший описание сетки, должен

```
void * CENTAUR_AllocArray (kind_t kind, size_t elemsize);
```

Рис. 5. Функция заведения массива значений

передать свое описание системе. Кентавр будет использовать эту сетку для распределения данных и организации обменов. Задание массивов значений, связанных с элементами сетки, также происходит внутри Кентавра, а пользователь, вместо того чтобы просто выделить память под массив вызовом `malloc()`, создает массив с помощью специальной функции (рис. 5).

Эта функция принимает размер элемента массива (параметр `elemsize`), а также «тип» массива (параметр `kind`), определяющий, с какими именно элементами сетки связан массив — это могут быть ее узлы, ребра, пограничные элементы и т. п. Набор возможных «типов» всегда предопределен и фиксирован для заданного класса сеток. Размер массива библиотека вычисляет сама, поскольку после инициализации ей уже известны параметры сетки.

Отметим, что пользователь получает обычный указатель `void *`, как если бы массив создавался функцией `malloc()`. Благодаря этой особенности весь дальнейший (изначально последовательный) код практически без изменений может быть использован в параллельной версии.

Для определения необходимых теневых граней используются трафареты, записанные для каждого трафаретного цикла, — трафареты явно указываются пользователем, а система сама определяет структуру теневых граней и размещает их в памяти. По трафарету выясняется, какие данные на «тех» узлах могут потребоваться для вычисления на «этом» узле; именно эти данные и образуют теневую грань.

Трафареты также задают, какие массивы изменяются в трафаретных циклах. Используя это знание, система Кентавр организует передачу теневых граней — передача происходит после завершения трафаретного цикла, изменявшего эти данные, но до начала другого трафаретного цикла, использующего эти данные. При некоторых условиях такая передача может быть совмещена с вычислениями других циклов.

При распределении данных система Кентавр проанализирует все трафареты в программе и заведет необходимые теневые грани для

всех трафаретных циклов. Каждый трафаретный цикл будет преобразован в стадию, для которой (с помощью трафарета же) будет задано, какие данные (точнее, какая часть теневых граней) будут использоваться для чтения, а какие будут изменены при выполнении стадии.

При параллельном исполнении из одного описания стадии будет порождено несколько экземпляров стадий — по одной на каждый вычислительный процесс. Каждый экземпляр будет обрабатывать только свои части массивов. На основе информации из трафаретов будут выполнены необходимые передачи данных для синхронизации параллельных процессов.

Описанный выше подход позволяет решать различные задачи по оптимальному распределению данных между узлами без изменений в исходной программе.

Как уже было сказано выше, для эффективного исполнения желательно, чтобы размер пересылаемых данных между узлами был как можно меньше. Этого можно добиться, уменьшая периметры частей, на которые разбивается сетка. Для уменьшения нагрузки на сеть желательно, чтобы обменивающиеся теневыми гранями «соседние» процессы были близки в терминах коммуникационной сети. Данная оптимизация особенно важна для сетей большого диаметра (например, с топологией типа решетка или тор).

Для эффективной посылки и приема данных желательно, чтобы они естественным образом агрегировались при обменах. Этого можно добиться, расположив теневые грани в памяти непрерывным образом.

Отдельно необходимо отметить такую важную проблему, как балансировка нагрузки на узлы. Относительно легко поделить данные так, чтобы на каждый узел системы приходилось примерно одинаковое количество элементов сетки, но в вычислительном смысле элементы сетки неодинаковы — для одних элементов может оказаться необходимо проводить больше вычислений, чем для других. Да и вычислительные узлы могут иметь разную производительность. В результате, при равномерном разбиении сетки возможна ситуация, когда на один узел приходится больше вычислений, чем на остальные, что снижает общую производительность системы. Для решения этой

проблемы необходимо динамически балансировать нагрузку — перераспределять элементы сетки между узлами. Подход Кентавра позволяет делать это на уровне системы, опираясь на информацию о программе, специфицированную в трафаретах.

### 3.3. Обмены с ускорителями

Предлагаемая методика подразумевает перенос основной тяжести вычислений на ускорители. Прежде всего речь идет о графических ускорителях как о наиболее распространенном в настоящее время типе ускорителей вычислений. Тем не менее данная методика применима и к другим типам ускорителей, таким как ПЛИС[3].

Одним из ключевых моментов методики является автоматизация обменов между системной памятью и памятью ускорителей. Этим, в частности, определяются свойства понятия стадии. А именно, программист должен в декларативном виде описать то подмножество глобальных данных, которое используется в рамках стадии, и указать, каким именно образом оно используется (для чтения и/или записи). Не исключается также возможность автоматизации сбора такого рода информации: для этого могут применяться анализаторы кода, позволяющие отследить использование данных в коде стадии. Обсуждение этой проблемы, однако, выходит за рамки данной статьи.

Важно, что наличие информации об использовании данных допускает реализацию рантайма, осуществляющего обмен между системной памятью и памятью ускорителей автоматически, неявно для пользователя.

Наивная реализация рантайма может выглядеть следующим образом. Будем считать, что основное место хранения данных — системная память, и потребуем, чтобы после исполнения каждой стадии корректным было представление данных именно там. Далее, если исполнение стадии запланировано на центральном процессоре, то исполняем ее без дополнительных действий. Если же ее необходимо исполнить на ускорителе, то вначале осуществляем копирование на ускоритель всех данных, отмеченных как «для чтения», а после исполнения — копирование из ускорителя всех данных, отмеченных как

[3]ПЛИС (программируемая логическая интегральная схема) — интегральная схема, логика которой определяется не при изготовлении, а в результате программирования. В качестве укорителей вычислений обычно используется разновидность ПЛИС, называемая FPGA (field-programmable gate array).

«для записи». Таким образом будет обеспечена корректность и актуальность представления данных в системной памяти.

Конечно, вышеописанная наивная реализация может быть значительна улучшена — не обязательно каждый раз копировать все отмеченные данные. Например, в реализации системы Кентавр при инициализации проводится простой анализ размещения и мест использования данных, и в соответствии с этим экономично планируются вызовы операций копирования. В случае, когда все стадии программы исполняются на ускорителе, операций копирования между стадиями вообще не будет. Лишь в самом начале выполнения программы будут скопированы все данные на ускоритель, а по завершении счета — обратно в системную память для вывода результатов. Отметим, что анализируя зависимости между стадиями по данным, можно добиться совмещения передачи данных и счета.

Для прикладного программиста, который использует инструментарий Кентавр, все такие обмены задаются неявно — они строятся на основе трафаретов, описывающих трафаретные циклы в исходной программе.

### 3.4. Перенос кода на графические ускорители

Язык CUDA, предназначенный для программирования графических ускорителей, реализован как надстройка над языком Си. Эту его особенность можно использовать для облегчения переноса последовательной программы, изначально написанной на Си/Си++, на многоузловую машину с графическими ускорителями. Предлагаемая методика переноса заключается в создании специальным образом оформленного исходного кода, который допускает компиляцию как в CUDA (с помощью компилятора nvcc из набора инструментов фирмы Nvidia), так и в обычный объектный код для центрального процессора (с помощью любого компилятора Си). Исходный код, обладающий таким свойством, назовем *двуязычным*. Использование двуязычного кода упрощает перенос задачи на CUDA и может существенно помочь при дальнейшей отладке.

Основная идея того, как это сделать, заключается в том, чтобы вынести тело цикла в отдельную функцию, которая должна поразному вызываться в зависимости от того, хотим мы ее исполнить на центральном процессоре или же на ускорителе.

В настоящее время появляются и другие проекты, предлагающие использовать двуязычный код. Примером является упомянутый выше OpenACC [16], который позволяет легко переносить и совмещать в одном исходном файле код для классического процессора и графического ускорителя.

Конечно, язык CUDA накладывает дополнительные ограничения на код, который мы можем вынести на устройство. В частности, запрещены рекурсивные вызовы, не работает косвенная адресация. Поэтому в общем случае может потребоваться дополнительное преобразование кода. В проекте Кентавр предлагается задавать трафаретные циклы в определенном стиле, описанном в разделе 2.3, а также в [10], причем стиль этот переносится на язык CUDA с незначительными изменениями или вовсе без изменений.

Разбиение программы на стадии по описанной выше методике позволяет задавать различный код для различных архитектур: классических процессоров, графических ускорителей или ПЛИС. Таким образом, не обязательно стремиться выразить алгоритм двуязычным кодом (тем более, что в случае ПЛИС это вообще не представляется возможным). В любом случае система берет на себя все обмены между системной памятью и ускорителями (реализации этих обменов, разумеется, будут различаться для разных видов ускорителей). Задание специфичного кода позволяет добиться более высокой эффективности решения задачи на конкретной архитектуре.

## Заключение

В статье было дано описание принципов построения и реализации программного инструментария Кентавр, предназначенного для автоматизации переноса трафаретных программ на гибридные суперкомпьютеры.

Трафаретные программы обладают тем свойством, что полная информация о зависимостях между данными может быть специфицирована для них в компактном и естественном для человека виде. В рамках методики использования Кентавра эта информация задается программистом в виде специально определяемого трафарета для каждого цикла в программе, витки которого могут исполняться независимо, — трафаретного цикла.

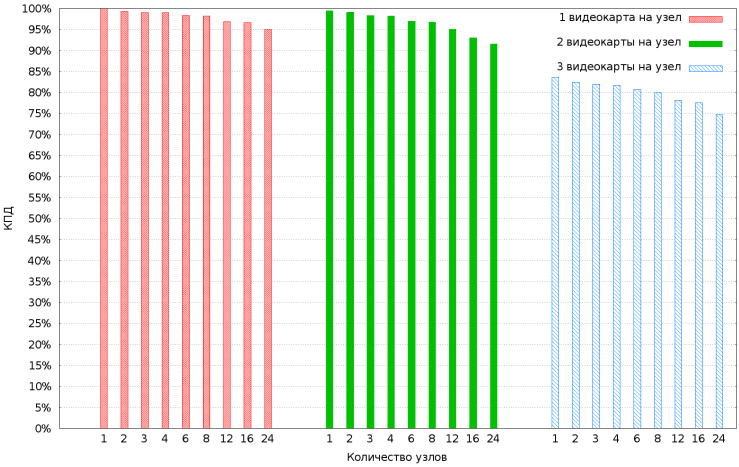


Рис. 6. Модельная задача распространения горения в предварительно перемешанной смеси. Отношение реального ускорения задачи к идеальному ускорению

Опираясь на эту информацию, инструментарий Кентавр автоматически строит распределение данных по вычислительным устройствам так, чтобы трафаретные циклы исполнялись на них параллельно. При этом обеспечивается передача необходимых данных между узлами суперкомпьютера и между процессорами и ускорителями внутри узлов.

Отличительной особенностью инструментария Кентавр является способ задания пространства ячеек (сетки), к которому применяется трафарет, позволяющий:

(1) не изменять тела трафаретных циклов по сравнению с последовательной программой;

(2) работать с областями произвольной формы, а также многоблочными, неструктурированными и адаптивными сетками.

В настоящее время проект Кентавр активно развивается. Текущая версия состоит из специальных библиотечных вызовов. Планируется разработка компилятора для автоматического преобразования исходного кода на основе аннотаций программиста. Первые эксперименты показывают КПД (реальное ускорение задачи на нескольких

графических ускорителях по сравнению с идеальным) около 80–90% (рис. 6).

В современных условиях, когда увеличение уровней иерархии памяти в гибридных машинах значительно усложнило разработку прикладных программ, создание проблемно-ориентированных инструментариев для отдельных классов программ стало особенно актуально. Проект Кентавр, направленный на трафаретные вычисления, показывает, что можно значительно автоматизировать перенос последовательных программ на гибридные суперкомпьютеры, равно как и создание новых параллельных программ с нуля.

Авторы выражают благодарность Андрею Климову за поддержку при работе над проектом Кентавр и за ценные замечания по тексту данной статьи.

## Список литературы

[1] Nvidia CUDA, http://www.nvidia.com/object/cuda_home_new.html. ↑[]

[2] OpenCL, http://www.khronos.org/opencl/. ↑[]

[3] Проект Кентавр, http://centaur.botik.ru/. ↑[]

[4] Datta K., Murphy M., Volkov V., Williams S., Carter J., Oliker L., Patterson D., Shalf J., Yelick K. *Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures* // Proceedings of the 2008 ACM/IEEE Conference on Supercomputing // SC '08. — Piscataway, NJ, USA : IEEE Press, 2008, p. 4:1–4:12 ↑[], 1.3

[5] Климов Ю. А., Орлов А. Ю., Шворин А. Б. *Перспективные подходы к созданию масштабируемых приложений для суперкомпьютеров гибридной архитектуры* // Программные системы: теория и приложения: электронный научный журнал, 2011, № 4 (8), с. 45–59, http://psta.psiras.ru/read/psta2011_4_45-59.pdf ↑[]

[6] OpenMP, http://openmp.org/. ↑[]

[7] Intel Cluster OpenMP User's Guide, http://software.intel.com/file/6330. ↑[]

[8] Message Passing Interface, http://www.mpi-forum.org/. ↑[]

[9] Unified Parallel C, http://upc.gwu.edu/. ↑[]

[10] Проект Кентавр: методические материалы, http://centaur.botik.ru/metodicheskie-materialy. ↑[], 3.4

[11] Dean J., Ghemawat S. *MapReduce: simplified data processing on large clusters* // Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation // OSDI'04. — Berkeley, CA, USA : USENIX Association, 2004. Vol. **6**, p. 137–150 ↑1.1

[12] Schäfer A., Fey D. *LibGeoDecomp: A Grid-Enabled Library for Geometric Decomposition Codes* // Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. — Berlin, Heidelberg : Springer-Verlag, 2008, p. 285–294 ↑1.3, 2.1

[13] Maruyama N., Nomura T., Sato K., Matsuoka S. *Physis: an implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers* // Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis // SC '11. — New York, NY, USA : ACM, 2011, p. 11:1–11:12 ↑2

[14] Balay S., Brown J., Buschelman K., Gropp W. D., Kaushik D., Knepley M. G., McInnes L.C., Smith B. F., Zhang H. PETSc Web page, 2012, http://www.mcs.anl.gov/petsc. ↑1.3, 2.1

[15] Singh S. *Computing without processors* // Communications of the ACM, 2011. **Vol. 54**, no. 8, p. 46–54 ↑2.1

[16] OpenACC, http://www.openacc-standard.org/. ↑2.1, 3.4

[17] Augonnet C., Thibault S., Namyst R., Wacrenier P.–A. *StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures* // Proceedings of the 15th International Euro-Par Conference. Lecture Notes in Computer Science — Delft, The Netherlands : Springer, 2009. Vol. **5704**, p. 863–874 ↑2.1, 3.1

[18] Charm++, http://charm.cs.uiuc.edu/research/charm/. ↑2.2

[19] Андрианов А. Н., Ефимкин К. Н. *Подход к параллельной реализации численных методов на неструктурированных сетках* // Вычислительные методы и программирование: новые вычислительные технологии, 2007. **Т. 8**, № 2, с. 6–17 ↑3.2

Рекомендовал к публикации *д.ф.-м.н. С. М. Абрамов*

*Об авторах:*

**Юрий Андреевич Климов**

Старший научный сотрудник ИПМ им. М.В. Келдыша РАН, ведущий инженер-программист ИПС им. А.К. Айламазяна РАН, к.ф.-м.н. Разработчик метода специализации на основе частичных вычислений для программ на объектно-ориентированных языках, принимал активное участие в разработке коммуникационного программного обеспечения для сетей SCI, 3D-тор суперкомпьютера СКИФ-Аврора и «МВС-Экспресс» суперкомпьютера К-100. Область научных интересов: суперкомпьютеры, оптимизация и преобразование программ, функциональное программирование.

*e-mail:* yuklimov@keldysh.ru

**Антон Юрьевич Орлов**

Инженер-исследователь ИПС им. А.К. Айламазяна РАН. Разработчик новой версии языка Рефал+ и интегрированной среды разработки для него, принимал активное участие в разработке коммуникационной сети 3D-тор суперкомпьютера СКИФ-Аврора. Область научных интересов: высокопроизводительные вычисления, архитектура вычислительных комплексов, анализ и преобразование программ, функциональное программирование.

*e-mail:* orlov@mccme.ru

**Артем Борисович Шворин**

Инженер-программист ИПС имени А.К. Айламазяна РАН. Принимал активное участие в разработке коммуникационной сети 3D-тор суперкомпьютера СКИФ-Аврора. Область научных интересов: метавычисления, моделирование, функциональное программирование.

*e-mail:* shvorin@gmail.com

*Образец ссылки на эту публикацию:*

Ю. А. Климов, А. Ю. Орлов, А. Б. Шворин. *Программный инструментарий для трафаретных вычислений на гибридных суперкомпьютерах* // Программные системы: теория и приложения : электрон. научн. журн. 2012. Т. 3, № 2(11), с. 23–49.

*URL:* http://psta.psiras.ru/read/psta2012_2_23-49.pdf

Yu. A. Klimov, A.Yu. Orlov, A. B. Shvorin. *Software Toolkit for Implementing Stencil Codes on Hybrid Supercomputers.*

Abstract. The problem of porting programs to hybrid (heterogeneous) supercomputers is considered. The process of porting is known to be difficult and error prone and generally requires a lot of efforts. Nevertheless, for some restricted classes of programs it can be automated. One of such classes is stencil codes, which are used widely in scientific computations.

The ideas and implementation of a Centaur toolkit aimed at automatic parallelization of stencil programs for running on hybrid supercomputers are described. The main task performed by Centaur is to organize automatic data interchange between supercomputer nodes as well as interchange between accelerator and CPU inside each hybrid node.

*Key Words and Phrases:* stencil codes, hybrid supercomputing, domain-specific languages, Centaur project.