УДК 004.7

SKIFCH: ЭФФЕКТИВНЫЙ КОММУНИКАЦИОННЫЙ ИНТЕРФЕЙС

Ю.А. Климов, А.Ю. Орлов, А.Б. Шворин

SKIFCH: HIGH PERFORMANCE COMMUNICATION INTERFACE

Yu.A. Klimov, A.Yu. Orlov, A.B. Shvorin

В работе описывается SkifCh — низкоуровневый интерфейс передачи сообщений. Данный интерфейс эффективно поддерживается на уровне сетевого оборудования, которое, в свою очередь, может быть реализовано в ПЛИС (как сделано в суперкомпьютере СКИФ-Аврора) или в специализированных микросхемах. Интерфейс SkifCh может быть использован для высокоэффективных сетевых обменов непосредственно из прикладных программ, а также для реализации коммуникационных библиотек более высокого уровня. На данный момент поверх интерфейса SkifCh реализованы системы MPI, SHMEM, GASNet и ARMCI. В работе также приведено сравнение эффективности использования SkifCh и MPI на суперкомпьютере СКИФ-Аврора.

Ключевые слова: суперкомпьютер, коммуникационная сеть, интерфейс передачи сообщений, SkifCh.

We present highly efficient low level message passing interface named SkifCh. This interface is easily supported by network hardware which can be implemented in FPGA (as is in the case for SKIF-Aurora supercomputer) or in specific microchips. SkifCh interface is intended for use as highly efficient communication layer by applications or higher level communication libraries. For the moment there are several libraries implemented on top of SkifCh: MPI, SHMEM, GASNet, and ARMCI. We also provide a performance comparison for SkifCh versus MPI using the SKIF-Aurora supercomputer.

Keywords: supercomputer, HPC, interconnect, message passing interface, SkifCh.

Введение

На сегодняшний день интерфейс передачи сообщений MPI [1] является стандартом дефакто при написании программ для работы на машинах с распределенной памятью. Он отлично задокументирован, имеет ряд замечательных реализаций, и, главное, является достаточно выразительным средством как для написания множества прикладных программ, так и для создания более высокоуровневых библиотек.

Однако, как это обычно бывает, платой за универсальность является потеря эффективности там, где подошли бы более узкоспециализированные решения. В случае MPI, благодаря механизму переупорядочивания сообщений при приеме их из сети, программист получает определенное удобство структуризации своей программы. Но, независимо от того, пользуется ли преимуществами этого механизма, он в любом случае вынужден расплачиваться некоторой потерей эффективности.

В данной работе мы предлагаем интерфейс передачи сообщений SkifCh, в котором переупорядочивание сообщений на приемном конце значительно упрощено и может быть частично или полностью реализовано в аппаратуре. SkifCh является интерфейсом более низкого уровня чем MPI в том смысле, что возможности и ограничения аппаратуры выражены в нем более явно. Тем не менее, он допускает эффективную реализацию как полного стандарта MPI, так и других распространенных библиотек обмена данными. На момент написания этой статьи имеются (помимо MPI-2) реализации SHMEM [2], GASNet [3] и ARMCI [4] поверх SkifCh. Также планируется поддержка Charm++ [5] и других систем.

За счет упрощения и перехода на более низкий уровень SkifCh позволяет более эффективно использовать возможности коммуникационной сети. В первую очередь речь идет о темпе выдачи сообщений [6], и это будет продемонстрировано ниже. Заметим, что дисциплины коммуникаций на основе односторонних обменов данными, такие как SHMEM [2], могут быть реализованы на SkifCh без существенной потери эффективности. В связи с этим мы призываем авторов библиотек для параллельного программирования рассматривать SkifCh как возможную основу коммуникационной части.

1. Особенности MPI

Интерфейс MPI предоставляет пользователю возможность посылать сообщения в одном порядке, а принимать — в другом.

```
int MPI_Recv (void* buf,
              int count,
              MPI_Datatype datatype,
              int source,
              int tag,
              MPI_Comm comm,
              MPI_Status *status);
```

Рис. 1. Прототип функции MPI_Recv()

Для этого при вызове функции MPI_Recv() (см. рис. 1) пользователь указывает коммуникатор comm, ранг отправителя source (либо константу MPI_ANY_SOURCE) и тег сообщения tag (либо константу MPI_ANY_TAG). Если по сети пришло несколько сообщений, то библиотека MPI выберет из них первое, которое удовлетворяет указанным параметрам.

Таким образом, реализация библиотеки MPI должна принимать сообщения в том порядке, в котором они приходят, а выдавать их пользователю в том порядке, в котором он просит. Это означает, что приходящие сообщения необходимо хранить и переупорядочивать.

Переупорядочивание вносит ощутимый накладной расход (что будет показано в разделе 4), даже если пользователь не использует переупорядочивание явно (то есть вызывает MPI_Recv() с параметрами MPI_ANY_SOURCE и MPI_ANY_TAG). Дело в том, что библиотека не может заранее знать, потребуется ли хранение входящего сообщения, и поэтому стек функций, обеспечивающих механизм переупорядочивания, в любом случае должен отработать. «Схлопнув» этот стек (убрав его совсем или поместив в аппаратуру), возможно добиться существенного повышения темпа обработки сообщений.

Отказ от накладных расходов на переупорядочивание входящих сообщений предполагает возложение контроля за переносом сообщений из сети в рабочие процессы на пользователя интерфейса. Таким образом, для эффективного использования коммуникационной среды необходимо сформулировать интерфейс более низкого, чем MPI, уровня.

2. Интерфейс SkifCh

2.1. Краткое описание интерфейса SkifCh

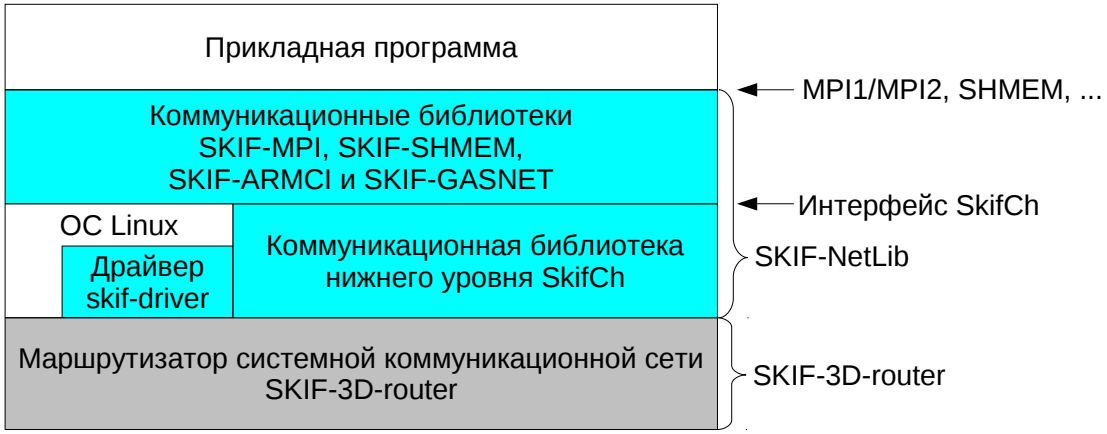

Рис. 2. Стек протоколов

Для сети топологии 3D-тор суперкомпьютера СКИФ-Аврора [7] был разработан низкоуровневый интерфейс SkifCh. Его положение в стеке коммуникационных протоколов показано на рис. 2.

```
// struct iovec from sys/uio.h:
struct iovec {
    void * iov_base;
    size_t iov_len;
};

#define SKIFCH_CONT_SIZE 3
typedef uint32_t netaddr_t;
struct SkifCh;

ssize_t SkifCh_Send (SkifCh * skifch,
                     netaddr_t dst_netaddr,
                     const struct iovec * iov,
                     int iov_count);

ssize_t SkifCh_Recv (SkifCh * skifch,
                     struct iovec cont[SKIFCH_CONT_SIZE],
                     int * cont_count);

int SkifCh_RecvComp (SkifCh * skifch);
```

Рис. 3. Описание ядра интерфейса SkifCh

Ядро интерфейса SkifCh представлено на рис. 3. Остановимся на нем более подробно, чтобы сравнить с интерфейсом MPI.

2.1.1. Функция SkifCh_Send()

Функция SkifCh_Send() похожа на функцию MPI_Send(), а также на библиотечную функцию writev() из sys/uio.h (см., например, [8]). Она имеет следующий прототип:

- skifch — указатель на структуру канала SkifCh;
- dst_netaddr — адрес получателя (аналог ранга в MPI);
- iov и iov_count — задают данные в формате аналогичном тому, что используется в функциях readv() и writev() [8];
- возвращаемое значение — количество успешно отправленных байтов, то есть размер отправленного сообщения.

Параметры iov и iov_count задают массив указателей на данные пользователя. Разбиение на части при передаче не учитывается: можно считать, что посылается один сплошной массив данных. Разбиение используется лишь для того, чтобы избежать дополнительного копирования при отправке данных.

Функция SkifCh_Send() формирует сообщение из данных пользователя и отправляет его в сеть, причем размер сообщения определяется динамически в зависимости от состояния сети. Таким образом SkifCh_Send() имеет право отправить не все данные пользователя, а только их часть, что аналогично поведению функции writev().

Чтобы отправить оставшиеся данные, необходимо соответствующим образом изменить параметры iov и iov_count (чтобы они описывали оставшиеся данные) и, возможно, изменить сами данные, а затем снова вызвать функцию посылки SkifCh_Send(). Данную процедуру, возможно, придется повторять многократно.

2.1.2. Функция SkifCh_Recv()

Функция SkifCh_Recv() значительно отличается и от MPI_Recv(), и от readv(). Она имеет следующие параметры:

- skifch — указатель на структуру канала SkifCh;
- cont — указатель на массив, размера не менее, чем SKIFCH_CONT_SIZE;
- cont_count — указатель на место, где будет размещен размер заполненной части массива cont после завершения функции;
- возвращаемое значение — количество успешно полученных байтов, размер полученного сообщения.

Массив cont размера SKIFCH_CONT_SIZE должен быть заведен пользователем перед вызовом, например, на стеке. Элементы массива не должны быть проинициализированы.

После вызова SkifCh_Recv() в возвращаемом значении пользователю будет выдан размер принятого сообщения в байтах. Массив cont будет заполнен указателями на части сообщения, которые хранятся во внутренней памяти реализации SkifCh. Реальное количество заполненных элементов массива cont будет указано в cont_count. Разбиение сообщения на части несущественно, можно считать, что сообщение — это один сплошной массив.

За один вызов SkifCh_Recv() будет получено одно сообщение, которое было отправлено одним вызовом функции SkifCh_Send(). Как и при отправке, чтобы получить все необходимые данные, отправленные несколькими вызовами SkifCh_Send(), потребуется такое же количество вызовов функции SkifCh_Recv().

SkifCh гарантирует порядок сообщений между отправителем и получателем. Однако при приеме сообщения заранее неизвестно, от какого отправителя будет получено сообщение. Поэтому, даже если данные были отправлены несколькими подряд идущими вызовами функции SkifCh_Send(), то не обязательно они будут приняты подряд идущими вызовами функции SkifCh_Recv(). Между получением сообщений от данного отправителя могут быть получены сообщения от других отправителей.

После обработки сообщения пользователь обязан вызвать SkifCh_RecvComp(), которая уведомляет реализацию SkifCh о том, что сообщение прочитано, и соответствующее место в служебном буфере можно использовать повторно.

Такой механизм, в частности, позволяет избежать обязательного копирования на приеме. Если пользователь может обработать полученное сообщение на месте, то интерфейс позволяет ему это сделать.

2.2. Каналы SkifCh

Следует отметить, что теги и/или коммуникаторы в MPI используются для разделения пространства сообщений между различными подсистемами — например, между библиотекой и программой. Совсем выбросить такой полезный механизм было бы опрометчиво, поэтому разделение пространства сообщений возможно и с использованием интерфейса SkifCh. Для этого каждая часть программы (например, библиотека), которая должна использовать сеть независимо от других систем, получает свой канал SkifCh и свой сетевой адрес dst_netaddr. Это обеспечивает получение лишь тех сообщений, которые предназначены именно ей. В некотором смысле каналы SkifCh аналогичны портам протокола TCP/IP.

При помощи каналов интерфейс SkifCh абстрагирует для пользователя разделение сообщений между получателями. Это может обеспечиваться непосредственно аппаратурой, либо программной прослойкой. В случае сети 3D-тор суперкомпьютера СКИФ-Аврора сетевые адаптеры реализованы в ПЛИС, и разделение пространства сообщений происходит именно там. Использование ПЛИС позволяет гибко описывать, какая часть сортировки и переупорядочивания сообщений будет реализована в аппаратуре (при ее ограниченных возможностях), а какая будет выполняться на уровне библиотеки (если это реально нужно в приложении), например, в библиотеке MPI.

2.3. Особенности SkifCh и его место в разработке ПО

SkifCh позиционируется авторами в первую очередь как низкоуровневое средство, на котором следует реализовывать коммуникационные библиотеки более высокого уровня, такие как MPI, SHMEM, Charm++ и т. п. Однако это не умаляет возможности использования SkifCh непосредственно при разработке прикладных программ.

В чем же преимущество использования низкоуровневого интерфейса по сравнению, например, с MPI?

Прежде всего — в повышении эффективности за счет снижения накладных расходов. Например, функция SkifCh_Recv() (в отличие от MPI_Recv() и readv()) *не копирует* сообщение в буфер пользователя, а оставляет его в буферах библиотеки SkifCh и показывает, где именно оно расположено. Пользователь волен скопировать сообщение в свой буфер либо обработать его на месте. Основная идея такого подхода заключается в том, чтобы иметь возможность избежать лишнего копирования.

Интерфейс SkifCh подразумевает, что внутри библиотеки не происходит дополнительной буферизации. Например, если сообщение не может быть отправлено немедленно, то вызов SkifCh_Send() завершится неуспехом, и программист должен повторить вызов через некоторое время.

При вызове SkifCh_Send() с большой порцией данных, в сеть может быть отправлено сообщение, содержащее только начальную часть данных. Эта особенность вызвана ограничением аппаратных ресурсов сети и отсутствием буферизации в SkifCh. Поэтому, если программисту необходимо передать большую порцию данных, то при отправлении оно будет нарезано на отдельные сообщения. Это автоматически означает, что программисту необходимо озаботиться собиранием больших порцией данных из нескольких сообщений на приемном конце.

Заметим, что из-за отсутствия буферизации и переупорядочивания, SkifCh_Recv() получает сообщения от разных отправителей в произвольном порядке. Однако, как и в случае

с интерфейсом MPI, SkifCh гарантирует (и накладывает соответствующие ограничения на нижележащую реализацию сети), что сообщения от одного отправителя приходят в том же порядке, в котором были посланы.

Заботясь об эффективности, не стоит забывать и о продуктивности — трудоемкости создания программного обеспечения с точки зрения разработчика. Как правило, чем ниже уровень, на который опирается разработчик, чем ближе к аппаратуре, тем продуктивность ниже. (Этим фактом, в частности, оправдывается существование высокоуровневых средств программирования.) Описанные выше ограничения SkifCh, с одной стороны, действительно усложняют его использование. С другой стороны, усложнения касаются в основном обмена большими сообщениями. Во многих случаях (например, для библиотеки SHMEM) эти ограничения несущественны. И именно благодаря этим ограничениям удается не потерять эффективность сети.

3. Примеры использования SkifCh

В рамках работ по созданию сети 3D-тор суперкомпьютера СКИФ-Аврора авторами были реализованы библиотеки MPI и SHMEM [2], основанные на описанном выше интерфейсе SkifCh.

Другими коллективами были разработаны системы GASNet [3] и ARMCI [4], что показывает достаточную функциональную наполненность и жизнеспособность интерфейса SkifCh.

SkifCh изначально разрабатывался для сети 3D-тор суперкомпьютера СКИФ-Аврора. Как уже было сказано, реализация коммуникационной среды в этой машине выполнена в ПЛИС, что дает достаточно большую свободу при выборе границы и методов разделения работы между программной и аппаратной частями. Впоследствии SkifCh был перенесен на сеть МВС-Экспресс [9], на основе которой в ИПМ РАН совместно с НИИ «Квант» были созданы суперкомпьютеры МВС-Экспресс и К-100. Эта сеть основана на прямой коммутации PCI-Express при помощи стандартных микросхем фирмы PLX.

Последнее является свидетельством того, что интерфейс SkifCh обладает достаточной гибкостью и может быть адаптирован к разным типам коммуникационного оборудования.

4. Эффективность SkifCh

В работе [6] была продемонстрирована важность такого показателя качества коммуникационной сети как темп выдачи сообщений.

Для сравнения эффективности работы с использованием интерфейсов SkifCh и MPI будем использовать стандартный тест MsgRate[2] [10]. Ядро его реализации на MPI приведено на рис. 4. С использованием SkifCh тест записывается аналогично.

Здесь один процесс (с номером 0) только отправляет сообщения, а другой (с номером 1) только принимает сообщения. Измеряется общее время выполнение теста. Количество итераций, деленное на полученное значение — это темп выдачи сообщений.

На графике на рис. 5 приведены результаты теста в зависимости от размера сообщения для реализаций на MPI и на SkifCh. Хорошо видно, что для коротких сообщений использование SkifCh дает громадное преимущество по сравнению с MPI на сети 3D-тор, реализованным поверх SkifCh.

В данном тесте никак не используются преимущества переупорядочивания сообщений на приемном конце. Есть сообщения только одного вида, и все они должны быть получены

[2]Этот тест входит в поставку ScaliMPI под именем «Ping-Ping», что может ввести в заблуждение, поскольку в наборе IMB [11] есть более известный тест с тем же именем Ping-Ping, который устроен совершенно иначе. Во избежание путаницы мы решили дать свое название тесту.

```
MPI_Barrier(MPI_COMM_WORLD);
for (i = 0; i < N /*количество итераций*/; i++)
  if (my_rank == 0)
    MPI_Send(buf, len, MPI_BYTE, 1 /*dst*/, 0, MPI_COMM_WORLD);
  else if (my_rank == 1)
    MPI_Recv(buf, len, MPI_BYTE, 0 /*src*/, 0, MPI_COMM_WORLD, &sts);
MPI_Barrier(MPI_COMM_WORLD);
```

Рис. 4. Тест MsgRate

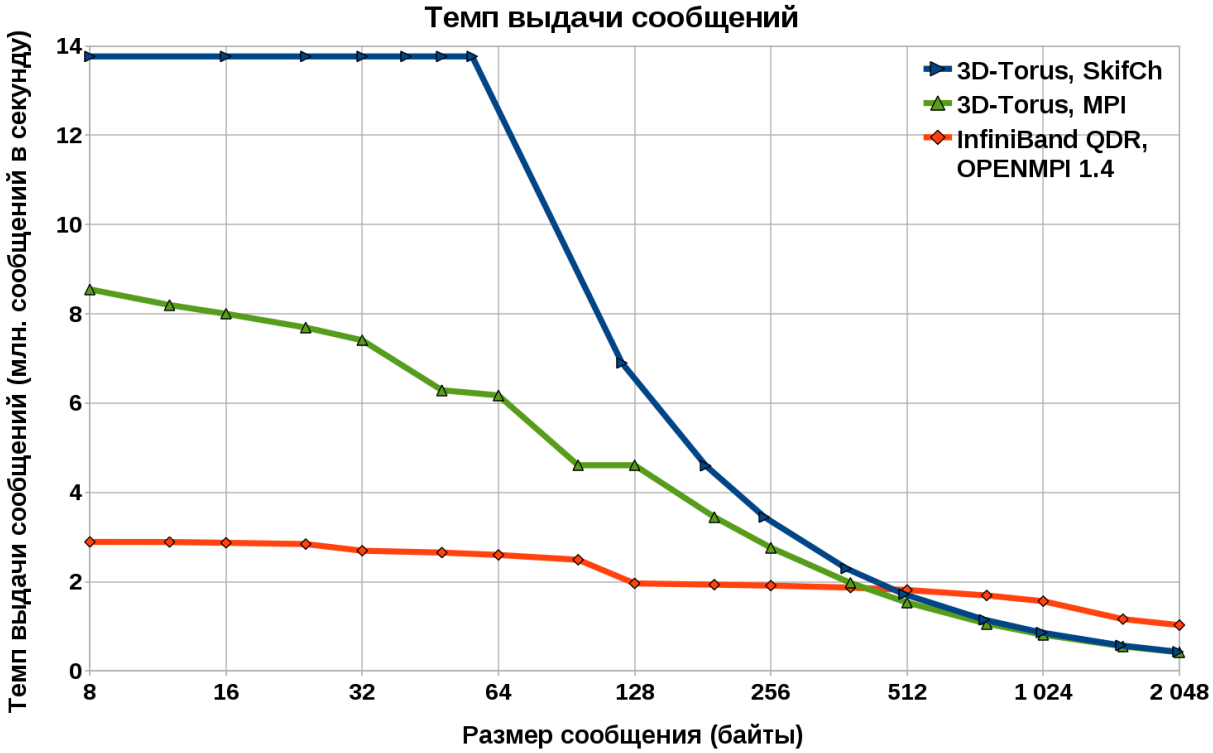

Рис. 5. Темп выдачи сообщений (больше — лучше)

в порядке их появления. Таким образом, разница в скорости работы теста на SkifCh и на MPI полностью обусловлена теми накладным расходами на переупорядочивание сообщений, которые присутствуют всегда — независимо от того, используется ли данный механизм или нет.

Для сравнения приведены также значения для того же самого кода на MPI, при использовании реализации Open MPI [12] на сети InfiniBand QDR. Все замеры (и на сети 3D-тор, и на сети InfiniBand) производились на узлах суперкомпьютера СКИФ-Аврора.

5. Заключение

В статье предложен низкоуровневый интерфейс передачи сообщений SkifCh. Были показаны преимущества и недостатки его использования по сравнению с интерфейсом MPI. Продемонстрировано, что SkifCh позволяет эффективно использовать возможности коммуникационной сети, достигая высокого темпа выдачи сообщений за счет простоты интерфейса и переноса части работы в аппаратуру.

Следует отметить, что близкие по положению в стеке сетевых протоколов высокоэффективные интерфейсы естественным образом возникают у многих разработчиков сетевой аппаратуры и коммуникационных библиотек.

Например, близким аналогом SkifCh в библиотеке MPICH2 [13] является внутренний интерфейс CH3, не предназначенный, однако, для стороннего использования.

В библиотеке Intel-MPI [14] таким интерфейсом является TMI, который дает возможность разработчиками эффективно использовать различные коммуникационные среды, например, QLogic PSM и Myrinet MX.

Большой интерес вызывает QLogic PSM [15] — низкоуровневый высокоэффективный интерфейс, разработанный фирмой QLogic для своей сетевой аппаратуры. В отличие от CH3 и TMI этот интерфейс предназначен для реализации не только MPI, но различных коммуникационных библиотек — имеются реализации SHMEM и других. PSM является наиболее близким из известных авторам данной работы аналогов SkifCh. Интересно, что фирма QLogic заявляет о эффективности PSM именно в связи с высоким темпом выдачи сообщений. Однако до сих пор нам не представилась возможность увидеть заявленные цифры в реальности.

Основными преимуществами интерфейса SkifCh по сравнению с аналогами могут быть названы следующие:

- ориентация на создание различных библиотек с высокоэффективной коммуникационной частью;
- достаточная гибкость, позволяющая не потерять эффективность сети на разных классах сетевого оборудования;
- доступность на современных суперкомпьютерах СКИФ-Аврора и К-100, оба из которых являются гибридными.

Заметим напоследок, что использование гибридных машин предполагает обычно мелкозернистые обмены. Это обстоятельство значительно повышает требования к качеству коммуникационной сети в целом, и, в частности, достижение высокой эффективности использования машины становится невозможным без высокого темпа выдачи сообщений.

Работы выполняются по научно-технической программе Союзного государства «СКИФ-ГРИД» [16], а также при поддержке РФФИ по проекту № 09-07-13598-офи_ц.

Статья рекомендована к публикации программным комитетом международной научной конференции «Параллельные вычислительные технологии 2011».

Литература

1. Message Passing Interface (MPI) // URL: http://www.mpi−forum.org/ (дата обращения: 15.12.2010).
2. SHMEM application programming interface // URL: http://www.shmem.org/ (дата обращения: 15.12.2010).
3. GASNet communication system // URL: http://gasnet.cs.berkeley.edu/ (дата обращения: 15.12.2010).
4. Aggregate Remote Memory Copy (ARMCI) library // URL: http://www.emsl.pnl.gov/docs/parsoft/armci/ (дата обращения: 15.12.2010).
5. Charm++ programming language // URL: http://charm.cs.uiuc.edu/ (дата обращения: 15.12.2010).
6. Темп выдачи сообщений как мера качества коммуникационной сети / Ю.А. Климов, А.Ю. Орлов, А.Б. Шворин // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: тр. Междунар. суперкомпьютер. конф. (20 – 25 сентября 2010 г., г. Новороссийск). – М.: Изд-во МГУ, 2010. – С. 414 – 417.

7. Опыт разработки коммуникационной сети суперкомпьютера «СКИФ-Аврора» / И.А. Адамович, А.В. Климов, Ю.А. Климов, А.Ю. Орлов, А.Б. Шворин // Программные системы: теория и приложения: электрон. науч. журн. – 2010. – № 3 (3). – С. 107 – 123. – URL: http://psta.psiras.ru/read/psta2010_3_107−123.pdf (дата обращения: 15.12.2010).
8. Linux man-pages project // URL: http://www.kernel.org/doc/man−pages/online/pages/man2/readv.2.html (дата обращения: 15.12.2010).
9. Лацис А.О. Вычислительная система МВС-Экспресс // URL: http://www.kiam.ru/MVS/research/mvs_express.html (дата обращения: 15.12.2010).
10. Тест Bandwidth // URL: http://botik.ru/~klimov/bandwidth.tgz (дата обращения: 15.12.2010).
11. Набор тестов Intel MPI Benchmarks (IMB) // URL: http://software.intel.com/en−us/articles/intel−mpi−benchmarks/ (дата обращения: 15.12.2010).
12. Open MPI: Open Source High Performance Computing // URL: http://www.open−mpi.org/ (дата обращения: 15.12.2010).
13. MPICH2: High-performance and Widely Portable MPI // URL: http://www.mcs.anl.gov/research/projects/mpich2/ (дата обращения: 15.12.2010).
14. Intel MPI library // URL: http://software.intel.com/en−us/articles/intel−mpi−library/ (дата обращения: 15.12.2010).
15. Scaling IB Fabrics to Meet the Needs of a PetaFlop World // URL: http://www.cse.scitech.ac.uk/disco/mew20/presentations/QLogic.pdf (дата обращения: 15.12.2010).
16. Суперкомпьютерная программа Союзного государства «СКИФ-ГРИД» (2007 – 2010 гг.) // URL: http://skif−grid.botik.ru/ (дата обращения: 15.12.2010).

Юрий Андреевич Климов, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН, yuklimov@keldysh.ru.

Антон Юрьевич Орлов, Институт программных систем им. А.К. Айламазяна РАН, orlov@mccme.ru.

Артем Борисович Шворин, Институт программных систем им. А.К. Айламазяна РАН, shvorin@gmail.com.

Поступила в редакцию 4 марта 2011 г.