ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №1(24), 2015, с. 175–188

УДК 519.682.3

В. А. Роганов, А. А. Кузнецов, Г. А. Матвеев, В. И. Осипов

Реализация T-системы с открытой архитектурой для CUDA-устройств с поддержкой динамического параллелизма и для гибридных суперЭВМ на их основе

Аннотация. В работе изложены основные принципы реализации расширения T-системы с открытой архитектурой (OpenTS), которое распространяет парадигму программирования T++ на аппаратуру современных спецвычислителей с поддержкой технологии CUDA. Специализированная версия T-надстройки, реализованная в микроядре системы OpenTS, способна работать автономно внутри CUDA-устройства, фактически превращая его в полноценный интеллектуальный T-узел гибридной суперЭВМ. В сочетании с поддержкой динамического параллелизма новейшими CUDA-устройствами это помогает существенно поднять процент утилизации графических ускорителей (GPU) без ручной балансировки статически распараллеленных блоков программы. Универсальная логика распараллеливания в T-системе теперь способна порождать и запускать легковесные счетные гранулы, избегая потерь, возникающих при интенсивном взаимодействии GPU с процессами базовой кластерной ОС. Спектр прикладных задач, которые подходят для новой модели вычислений, существенно пополняет привычные вычислительные ядра для GPU, привнося произвольную управляющую логику на уровень спецвычислителей. В качестве демонстрационного примера в статье рассматривается задача обращения криптостойких хэш-функций. Приведены базовые сведения, связанные с прикладными вопросами применения хэш-функций.

Ключевые слова и фразы: динамическое распараллеливание, T-система с открытой архитектурой, OpenTS, язык программирования T++, графические ускорители, гибридные кластерные системы, хэш-функции.

Введение

T-система [1, 2] является оригинальной российской разработкой, объединяющей в себе наиболее удачные черты функционального

программирования, dataflow-систем и традиционных языков и методов программирования.

OpenTS [3] представляет собой современную реализацию принципов T-системы, обладает открытой и масштабируемой архитектурой, легко адаптируемой к стремительно меняющимся аппаратным платформам современных суперкомпьютеров. Поддерживаемый системой OpenTS входной язык программирования T++ является синтаксически и семантически гладким расширением языка программирования C++, а среда исполнения T-приложений представляет собой ортогональную надстройку (T-суперструктуру) над стандартной последовательной средой программирования.

Подход к автоматическому динамическому распараллеливанию программ, предложенный в T-системе, позволяет получить хорошие результаты по утилизации вычислительных мощностей кластерных суперкомпьютерных установок, что связано с природой используемой модели вычислений. Однако сегодня все большую роль в высокопроизводительных вычислениях начинают играть гибридные кластерные установки, узлы которых оснащаются специализированными сопроцессорами, причем суммарная вычислительная мощность последних может заметно превосходить суммарную мощность центральных процессоров.

К сожалению, сложность внутреннего устройства ядра T-системы довольно резко контрастировала с достаточно простой моделью вычислений таких распространенных спецпроцессоров, как CUDA-устройства. Их использование в счетных задачах, реализованных на T++, ранее фактически ограничивалось простым вызовом вычислительных ядер на GPU.

У такого примитивного способа работы с GPU есть два весьма существенных недостатка. Во-первых, после вызова вычислительного ядра на спецвычислителе вызывающий поток ОС приостанавливал работу до получения результата. Во-вторых, взаимодействие T-логики и вычислительных ядер происходило посредством «бутылочного горлышка» программно-аппаратного интерфейса со спецвычислителем. Решение первого вопроса путем порождения дополнительных потоков еще более увеличивало накладные расходы, что делало вызовы небольших вычислительных ядер совершенно неэффективными. Это обстоятельство сделало актуальной задачей поддержку автономной работы T-суперструктуры внутри спецвычислителей.

В работе [5] авторами дано краткое описание методов адапта-

ции T-суперструктуры к простым спецвычислителям вообще, и к CUDA-устройствам с поддержкой Compute capability 2.1+ в частности. Благодаря усилиям NVIDIA ситуация с доступным для программиста арсеналом средств на стороне CUDA-устройств стремительно улучшается. В следующем разделе кратко рассматриваются новые возможности, которые более всего способствуют сближению T-парадигмы с технологией GPGPU. В реализации новой версии микроядра OpenTS были использованы самые современные возможности CUDA-устройств с Compute capability 3.5 и 5.0.

1. Нововведения в CUDA-архитектуре

Ниже приводится перечисление основных и, с точки зрения разработчиков системы OpenTS, «прорывных» возможностей технологии CUDA, доступных с появлением версии CUDA SDK 6.0 и устройств с Compute capability 3.5+.

1.1. Динамический параллелизм

Ничто не сближает парадигму T-системы с технологией CUDA так решительно, как поддержка динамического параллелизма, представленная на рис. 1. Эта технология позволяет ядрам/потокам GPU динамически и асинхронно запускать новые вычислительные ядра. В контексте переноса T-системы на CUDAустройства данная возможность радикально увеличивает эффективность и упрощает сопряжение высокоуровневой логики порождения и анализа результатов промежуточных подзадач и базовой SIMT-логики, свойственной технологии CUDA.

Принципиально новым качеством, таким образом, становится возможность писать высокоуровневый код, запускающий традиционные ядра (примитивы алгоритмов) и обрабатывающий их результаты, причем сам высокоуровневый код является высокопараллельным взаимодействием T-функций, обменивающихся в рамках подхода поставщик-потребитель посредством так называемых неготовых значений.

1.2. Унифицированная память

Поддержка общего адресного пространства между CPU и GPU упрощает программирование, обеспечивая приложениям доступ к

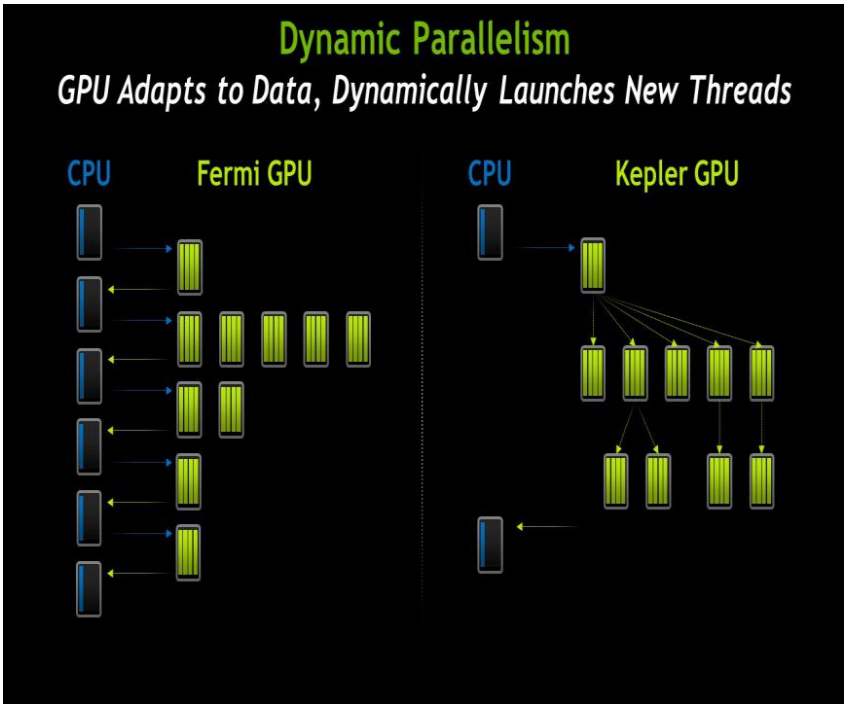


Рис. 1. Динамический параллелизм в NVidia GPU

памяти без необходимости вручную копировать данные с одной памяти в другую как показано на рис. 2.

Для T-системы это упрощает реализацию прозрачного взаимодействия между T-логикой кластерного уровня и спецвычислителем посредством доступа к T-переменным. Это хорошо согласовано с реализацией суперпамяти в OpenTS, которая также базируется на принципах введения единого адресного пространства для взаимодействующих T-процессов.

1.3. Hyper-Q

Технология Hyper-Q позволяет задействовать все вычислительные возможности центрального процессора и GPU через использование графического ускорителя сразу всеми ядрами CPU (см. рис. 3). Устранение «бутылочного горлышка» по задействованию множества потоков

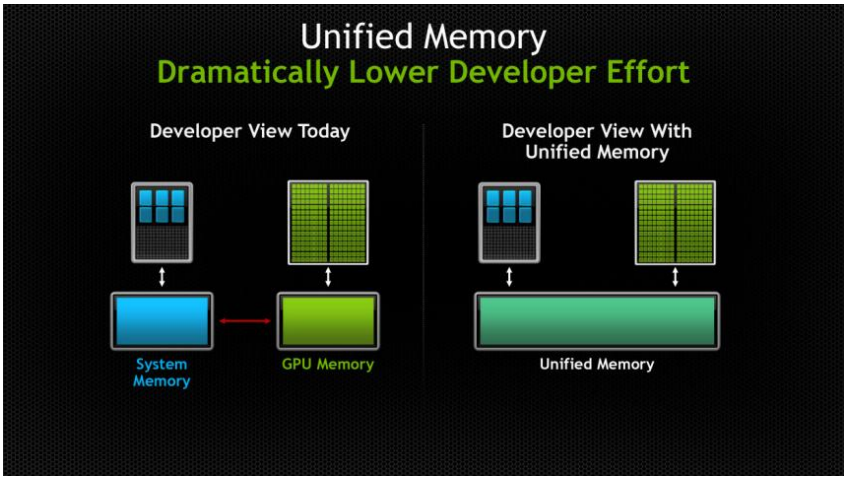


Рис. 2. Технология унифицированной памяти на CUDA-устройствах

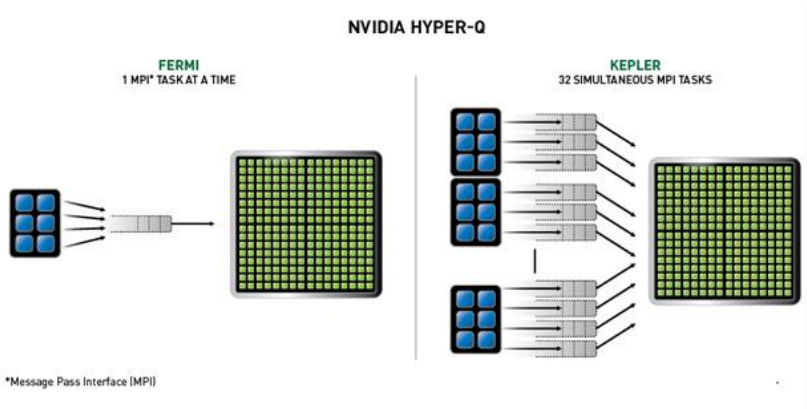


Рис. 3. Технология NVidia Hyper-Q

внутри GPU, присущего предыдущим поколениям CUDA-устройств, позволяет максимально утилизировать вычислительную мощность при порождении значительного количества параллельно работающих вычислительных ядер.

По всей видимости, данная возможность вообще является необходимой в условии внедрения динамического параллелизма, так как

в противном случае между ним и многопоточностью (CUDA streams) возникают серьезные препятствия. Интеллектуальный планировщик гранул внутри CUDA-устройства является, на наш взгляд, совершенно логичным продолжением этой возможности.

2. Современные CUDA-устройства с новыми возможностями

В настоящее время на рынке доступен уже некоторый ассортимент CUDA-устройств с вышеперечисленными новыми возможностями, как профессиональные (семейство Tesla), так и бюджетные (семейство GeForce) что позволяет приобщиться к их использованию как организациям, так и широким массам рядовых пользователей. Ниже приведены основные характеристики двух наиболее доступных на настоящий момент ускорителей.

2.1. Ключевые характеристики ускорителя NVidia Tesla K20

- Пиковая производительность для вычислений двойной точности с плавающей точкой: 1.17 Tflops;
- пиковая производительность для вычислений одинарной точности с плавающей точкой: 3.52 Tflops;
- полоса пропускания памяти (без ECC): 208 GB/s;
- объем памяти (GDDR5): 5 GB;
- ядер CUDA: 2496.

2.2. Ключевые характеристики ускорителя NVidia GeForce GTX 750 Ti

- Ядер CUDA: 640;
- базовая тактовая частота: 1020 MHz;
- тактовая частота с ускорением: 1085 MHz;
- быстродействие памяти: 5,4 Gbps;
- объем памяти: 2 GB;
- интерфейс памяти: 128-bit GDDR5;
- максимальная полоса пропускания памяти: 86,4 GB/sec.

3. Механизмы реализации T-суперструктуры для технологии CUDA

В данном разделе кратко изложено описание механизмов реализации T-ядра для вычислительных устройств CUDA, а также наиболее интересные детали реализации расширения T-ядра.

(1) Целью работы была реализация автономного (то есть без вмешательства CPU) распараллеливания, когда каждое CUDA-устройство рассматривается как независимый узел, который способен вычислять соответствующие функции. Поэтому вызовы T-функций автоматически доставляются на те узлы кластера, где присутствует хотя бы одно CUDA-устройство.
(2) Каждое T-ядро внутри CUDA-устройства выполняется независимо от остальных ядер и динамически формирует и производит вызовы для вложенных T-CUDA-функций, распределением нагрузки которых занимается подсистема динамической балансировки CUDA-устройства.
(3) Выделенный поток CPU формирует структуры в памяти GPU и вызывает набор ядер, соответствующий вызовам T-CUDA-функций, после чего ожидает их завершения и копирует данные из памяти GPU в T-значения результатов.

4. Основные детали реализации

Добавление нового модуля ATSS (Accelerated T-Super-Structure) в существующую инфраструктуру OpenTS было произведено в соответствии с существующей идеологией разработки T-расширений. Таким образом, новые механизмы по динамическому распараллеливанию счета внутри GPU сосуществуют с уже имеющимися механизмами для более «крупноблочного» распараллеливания программ на кластерном и метакластерном уровнях.

CUDA-ускорители рассматриваются как дополнительный нижний уровень для вычислений, как счетные узлы, попав на которые T-функция может динамически порождать новые T-функции для CUDA, а также «листовые» T-функции, которые уже не порождают более новых T-функций. Взаимодействие между T-функциями соответствует модели «поставщик-потребитель» и происходит посредством обращения к T-значениям, реализованным при помощи шаблонов языка T++.

«Листовые» T-функции могут либо проводить массированные вычисления самостоятельно, либо вызывать вычислительные ядра любых CUDA-библиотек. Листовая T-функция реализуется при помощи определения класса C++, в конструкторе которого указывается количество потоков, которые необходимы для эффективного выполнения. Число потоков, таким образом, вычисляется динамически, что придает модели вычислений дополнительную гибкость.

Если сторонние CUDA-библиотеки не используются в листовых T-функциях, то код T++ в новой гибридной модели является аппаратнонезависимым, то есть отсутствие на вычислительном узле CUDA-устройства не скажется на работоспособности программы, а только лишь на скорости счета. Это достигается следующим приемом: написанный программистом код T++ компонуется независимо в двух пространствах имен (понимаемых в терминах языка C++): для CPU и для GPU. Если при запуске программы T-микроядро не обнаружит устройств GPU, то будет вызываться версия кода, скомпилированного для CPU. Данная возможность позволяет прозрачно и параллельно задействовать произвольное количество GPU-устройств на узле, поскольку с точки зрения макропланировщика T-системы GPU выглядит просто как отдельный вычислительный поток, который инициирует запуск T-функций внутри CUDA-устройства. Это ключевая возможность, позволяющая эффективно производить параллельный расчет на связке CPU+GPU.

5. Примеры задач и некоторые прикладные вопросы криптоанализа

Парадигма T-системы хорошо подходит для достаточно широкого класса прикладных задач, в которых параллельные гранулы естественным образом появляются непосредственно в процессе счета.

В данной работе мы рассмотрим одну из этого множества задач, имеющую отношение к прикладным вопросам криптоанализа, таким как стойкость шифрования паролей, и, более широко, стойкость однонаправленных хэш-функций к различного рода атакам.

Для демонстрации актуальности данного вопроса обратимся к следующей истории.

5.1. Универсальный способ DoS-атаки, затрагивающий Web-платформы с некриптостойкими хэш-функциями

В 2011 году был обнаружен и обнародован новый способ нарушения работоспособности Web-сервисов, основанный на особенности реализации структур хэш-таблиц, используемых для организации хранения наборов данных в представлении ключ/значение в широком спектре языков программирования (Java, JRuby, PHP, Python, Ruby 1.8.7, V8 JavaScript Engine, ASP.NET). Большинство Web-фреймворков и Web-приложений на тот момент на этапе разбора

HTTP POST-запроса автоматически размещало передаваемые пользователем параметры в хэш-таблице, что позволяло атакующему контролировать наполнение этой таблицы при недостаточной криптостойкости используемой хэш-функции. В зависимости от языка программирования для достижения 100% загрузки CPU достаточно было сформировать поток запросов интенсивностью всего в несколько килобит в секунду. Например, при лимите на размер POST-запроса в 1 МБ, Python-фреймворк Plone тратил около 7 минут процессорного времени на запись 1 МБ данных (набор специально оформленных ключей), вызывающих коллизии (для успешной DoS-атаки на CPU Intel Core Duo достаточно потока в 20 Kbit/s). Для фреймворков на языке Ruby 1.8.7 при лимите на размер POST-запроса в 2 Мб на разбор двухмегабайтного блока данных на CPU Intel Core i7 тратилось около 6 часов, т.е. для поддержания постоянной 100% нагрузки достаточно потока в 850 бит/с. Для PHP-сценариев разбор POST-запроса размером 8 Мб занимал около 5 часов.

Понятно, что при такой распространенности уязвимых систем ситуация была совершенно критической. Причиной же послужило отсутствие стойкости используемой хэш-функции к коллизиям.

Хэш-функции также широко применяются для шифрования паролей. При этом, их стойкость к обращению, очевидно, играет ключевую роль. Арсенал простых методов криптоанализа в данном случае для хороших хэш-функций обычно ограничен использованием технологии радужных таблиц [6] и (в случае использования так называемой «соли») методом «грубой силы», то есть прямому перебору паролей, входящих в так называемый «хороший словарь». И в первом и во втором случае эффективность криптоанализа прямо пропорциональна быстродействию вычисления хэшей от генерируемого набором специальных правил множества паролей.

Примером использования криптоустойчивых хэш-функций может быть защита от несанкционированного доступа региональной сети «Ботик» в г. Переславль-Залесский [4]. Для аутентификации пользовательских компьютеров и роутеров используется простой протокол, в котором труднообратимая хэш-функция, совместно со случайно сгенерированной «солью» используется для формирования периодических запросов на аутентификацию клиентского оборудования. Для корректного ответа на эти запросы аутентификации необходимо знать секретный ключ. Не имея информации о секретном ключе клиент сети не может за разумное время сгенерировать корректный ответ, что

приводит к автоматическому блокированию клиентского IP-адреса.

Как в случае отсутствия «соли», так и при использовании фиксированной «соли» ситуация с защищенностью при использовании эффективно вычислимых хэш-функций резко меняется в худшую сторону: декодирование становится возможным с использованием так называемых радужных таблиц. Однако, и в случае нефиксированной соли ситуация в последнее время все более усугубляется, поскольку вычислительная мощность суперкомпьютерных установок сегодня позволяет очень быстро обращать хэш-функции типа MD5.

В качестве демонстрационной задачи для T-системы с открытой архитектурой, производящей динамическое распараллеливание вычислений внутри CUDA-устройств, была выбрана именно такая задача: обращение функции MD5, краткое описание решения которой приводится в последующем разделе.

5.2. Динамическое распараллеливание в задачах криптоанализа

Функционал демонстрационного приложения следующий: входными данными является набод хэш-кодов (результатов применения функции MD5+соль к паролям), на выходе через некоторый интервал времени (в случае использования видеоадаптера Nvidia GTX 750 — около одного часа) — список успешно дешифрованных паролей:

Output:

```
Reading hash info
    [0]: 69996ae2e822cc1f18404d4c66cc4932
    [1]: 7990359d7165e0765cbef00ce512ddaa
    [2]: fe197a0b29c576a667b0e4a25d54dc4a
    [3]: 1e40c35f623714ee1d32f91268614995
    [4]: 31b0bd0d97e64b907a5e4a4e6b119d47
    [5]: edb26e484a81b5941d3276c1cf4d8e57
    [6]: af3c0bb9f798a3253416e683b7113407
    [7]: 7b16519ad9768f87245296471cded6db
    [8]: c1545f31091b680d70796cdbc25e532b
    [9]: a6ba610820c7c6cffd5b0a698709e4d4
    [10]: abd2d5ed9e5c56a1429a1b42f851b7a0
    [11]: ea332c272f7e6498fbbdfe538feb9a1d
    Got 12 hashes
    PASSWORD[0]: adamovich
    PASSWORD[2]: baranessa
    PASSWORD[3]: biliberda
    PASSWORD[9]: inikogda
    PASSWORD[11]: moyparol
```

```
    PASSWORD[8]: nizachto
    PASSWORD[5]: parolchik
    PASSWORD[7]: vslomati
real 65m37.867s
user 21m33.047s
sys 44m4.871s
```

Программа находит почти любые {6,7,8}-буквенные пароли, а также фонетически изящные {9,10}-буквенные. Данная версия учитывает только латинские символы в нижнем регистре. В дальнейшем эту функциональность планируется наращивать.

В приложении активно используются новейшие возможности библиотек CUDA версии 6.0, такие как унифицированный доступ к памяти CUDA-устройства и динамический параллелизм.

При переборе динамически генерируемого словаря происходит автоматическое динамическое распараллеливание счета по всем доступным вычислительным узлам сначала на кластерном уровне, а затем внутри каждого отдельного CUDA-устройства.

Быстрая генерация хорошего словаря — сама по себе непростая задача, для решения которой перед началом счета составляется специальная таблица. Этот (подготовительный этап) занимает обычно несколько секунд, и производится с использованием фрагмента текста, содержащего фразы на языке, который был использован как наиболее вероятная основа при формировании паролей. Пароли, разумеется, не обязаны быть словами или словоформами данного языка (практика показывает, что обученная на английском тексте программа прекрасно расшифровывает пароли на санскрите), так что исходный текст для обучения скорее представляет средство «тонкой настройки», а не базы генерируемых словоформ.

Формирование элементов словаря происходит по иерархической схеме, начиная с верхнего уровня (префикса слова). Каждый префикс инициирует вызов T-функции для CUDA-устройства. Внутри CUDA-устройства выбираются продолжения слов, формирующие вложенные T-вызовы, приостанавливающие свой счет до завершения формируемых ими динамических вызовов листовых T-функций. Происходящее при этом «переключение контекста» T-функций внутри CUDA-устройства обеспечивается встроенными возможностями современных CUDA-адаптеров и в дальнейшем может быть существенно усовершенствовано в случае необходимости с использованием новых возможностей PTX-ассемблера.

В случае выполнения программы на вычислительном кластере время, необходимое на декодирование, будет уменьшаться пропорционально количеству вычислительных узлов (или, более точно, пропорционально количеству CUDA-устройств). Динамическое распараллеливание по узлам кластера при этом выполняется ядром T-системы, распределяющей вычислительные подзадачи, возникающие на верхнем уровне вычислений (то есть при разбиении словаря паролей по префиксу).

6. Заключение

В ядре системы OpenTS реализован подход автономного динамического распараллеливания [5], что позволяет во время счета приложений T++ параллельно задействовать ресурсы как CPU, так и GPGPU. При этом синтаксис T-программ существенно не меняется, что позволяет без значимых усилий адаптировать унаследованный код под современные высокопроизводительные вычислительные установки.

Важно отметить, что данный подход универсален в части поддержки произвольных аппаратных ускорителей, многие из которых сегодня можно программировать на языке OpenCL.

Благодарности. *Работы, положенные в основу данной статьи, были выполнены в рамках НИР «Методы и программные средства разработки параллельных приложений и обеспечения функционирования вычислительных комплексов и сетей нового поколения». Безусловной благодарности заслуживают также Институт программных систем им. А. К. Айламазяна РАН за приобретение профессионального оборудования NVIDIA Tesla K20, а также фирма NVIDIA за предоставление массы новых интересных возможностей в современных CUDA-устройствах.*

Список литературы

[1] С. М. Абрамов, В. А. Васенин, Е. Е. Мамчиц, В. А. Роганов, А. Ф. Слепухин, «Динамическое распараллеливание программ на базе параллельной редукции графов. Архитектура программного обеспечения новой версии T-системы», *Научная сессия МИФИ-2001*, Сборник научных трудов. Т. 2 (Москва, 22–26 января 2001 г.), с. 234 ↑ 175.

[2] С. М. Абрамов, А. А. Кузнецов, В. А. Роганов, «Кроссплатформенная версия T-системы с открытой архитектурой», *Труды Международной научной конференции «Параллельные вычислительные технологии*

(ПаВТ'2007)». Т. 1 (Челябинск, 29 января–2 февраля 2007 г.), изд. ЮУрГУ, Челябинск, 2007, с. 115–121 ↑ 175.

[3] С. М. Абрамов, А. А. Кузнецов, В. А. Роганов. «Кроссплатформенная версия T-системы с открытой архитектурой», *Вычислительные методы и программирование*, **8**:1(2) (2007), с. 175–180, URL http://num-meth.srcc.msu.ru/zhurnal/tom_2007/v8r203.html ↑ 176.

[4] А. А. Кузнецов.. «Исследование криптостойкости протокола аутентификации Botikkey к компрометации уязвимостей алгоритма хеширования MD5», *Программные системы: теория и приложения*, **6**:1(24) (2015), с. 135–144, URL http://psta.psiras.ru/read/psta2015_1_135-145.pdf ↑ 183.

[5] В. А. Роганов, А. А. Кузнецов, Г. А. Матвеев, В. И. Осипов. «Методы адаптации системы параллельного программирования OpenTS для поддержки работы T-приложений на гибридных вычислительных кластерах», *Программные системы: теория и приложения*, **4**:4(18) (2013), с. 17–31, URL http://psta.psiras.ru/read/psta2013_4_17-31.pdf ↑ 176, 186.

[6] URL http://ru.wikipedia.org/wiki/Радужная_таблица ↑ 183.

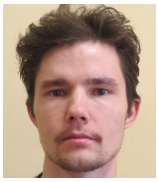
Рекомендовал к публикации *д.ф.-м.н. С. В. Знаменский*

Об авторах:

Владимир Александрович Роганов

Научный сотрудник ИПС им. А.К. Айламазяна РАН. Разработчик современных версий T-системы, ведущий разработчик системы OpenTS. Принимал активное участие в суперкомпьютерных проектах Союзного государства России и Беларуси, в том числе в проектах «СКИФ» и «СКИФ-ГРИД».

e-mail: var@pereslavl.ru



Антон Александрович Кузнецов

Научный сотрудник ИПС им. А.К. Айламазяна РАН. Один из разработчиков системы OpenTS, ведущий разработчик системы распределенных вычислений SkyTS. Область научных интересов: системы параллельного программирования, распределенные вычисления в гетерогенных средах, геоинформационные системы.

e-mail: tonic@pereslavl.ru

Герман Анатольевич Матвеев

Ведущий инженер-исследователь ИЦМС ИПС им. А.К. Айламазяна РАН. Один из разработчиков системы OpenTS. Принимал участие в суперкомпьютерных проектах Союзного государства России и Беларуси.

e-mail: gera@prime.botik.ru

Валерий Иванович Осипов

К.ф.-м.н., научный сотрудник ИПС им. А.К. Айламазяна РАН. Один из разработчиков системы OpenTS. Принимал участие в суперкомпьютерных проектах Союзного государства России и Беларуси.

e-mail: val@pereslavl.ru

Пример ссылки на эту публикацию:

В. А. Роганов, А. А. Кузнецов, Г. А. Матвеев, В. И. Осипов. «Реализация T-системы с открытой архитектурой для CUDA-устройств с поддержкой динамического параллелизма и для гибридных суперЭВМ на их основе», *Программные системы: теория и приложения*, 2015, **6**:1(24), с. 175–188.

URL http://psta.psiras.ru/read/psta2015_1_175-188.pdf

Vladimir Roganov, Anton Kuznetsov, German Matveev, Valerii Osipov. *Implementation of T-system with an open architecture for CUDA devices supporting dynamic parallelism and for hybrid computing clusters.*

Abstract. The article describes methods used to adapt the OpenTS parallel programming system and runtime for the optimal performance on hybrid clusters with computational nodes having accelerator hardware based on NVIDIA CUDA technology. (*In Russian*).

Key Words and Phrases: dynamic parallelization, T-system with an open architecture, OpenTS, T++ programming language, GPU accelerator, hybrid cluster systems, hash functions.

Sample citation of this publication

Vladimir Roganov, Anton Kuznetsov, German Matveev, Valerii Osipov. "Implementation of T-system with an open architecture for CUDA devices supporting dynamic parallelism and for hybrid computing clusters", *Program systems: theory and applications*, 2015, **6**:1(24), pp. 175–188. (*In Russian.*)

URL http://psta.psiras.ru/read/psta2015_1_175-188.pdf