

**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Ярославский государственный университет им. П.Г. Демидова»**

**Кафедра компьютерных сетей**

Сдано на кафедру  
«21» июня 2021 г.  
Заведующий кафедрой  
д.ф.-м.н., профессор  
\_\_\_\_\_Глызин С.Д.

Выпускная квалификационная работа  
**Развёртывание высокопроизводительного кластера  
с использованием менеджера ресурсов Slurm и контейнеров Singularity.**  
(Специальность 01.03.02 Прикладная математика и информатика)

Научный руководитель  
д.ф.-м.н., профессор  
Глызин С.Д. \_\_\_\_\_  
«21» июня 2021 г.  
Студент группы ИВТ-42БО  
Коляда М.М. \_\_\_\_\_  
«21» июня 2021 г.

Ярославль, 2021г.

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Теоретическая информация</b>	<b>6</b>
1.1 Описание архитектуры кластера	6
1.2 Менеджер конфигурации saltstack	8
1.2.1 Взаимодействие компонентов salt	8
1.2.2 Файлы состояний	9
<b>2 Практическая реализация</b>	<b>10</b>
2.1 Сборка пакетов из исходного кода	10
2.2 Подготовка конфигурации slurm	10
2.3 Подготовка конфигурации saltstack	12
2.4 Развёртывание кластера	16
2.4.1 Синхронизация пользователей ldap в slurm accounting	16
<b>3 Тестирование производительности</b>	<b>18</b>
3.1 Подготовка к тестированию	18
3.2 Проведение тестирования	19
<b>4 Решение уравнения теплопроводности</b>	<b>21</b>
4.1 Общие сведения	21
4.2 Пространственная дискретизация	21
4.3 Временная дискретизация	22
4.4 Критерий сходимости	22
4.5 MPI реализация	22
4.5.1 Декартова топология процесса	22
4.5.2 Межпроцессорное взаимодействие	23
4.6 Графические результаты	24
<b>Заключение</b>	<b>25</b>
<b>Список литературы</b>	<b>26</b>
<b>Приложение А — Скрипт сборки пакетов</b>	<b>27</b>
<b>Приложение Б — Файл конфигурации slurm</b>	<b>32</b>
<b>Приложение В — Файл конфигурации slurmdbd</b>	<b>33</b>
<b>Приложение Г — Скрипт синхронизации с ldap</b>	<b>34</b>
<b>Приложение Д — Файл конфигурации HPL.dat</b>	<b>36</b>
<b>Приложение Е — MPI реализация рекурсивной формулы</b>	<b>37</b>
<b>Приложение Ж — Функция межпроцессорного взаимодействия</b>	<b>38</b>

## Реферат

Объём 38 с. 4 гл. 11 источников.

Рассматривается один из возможных способов развёртывания кластера на базе менеджера ресурсов slurm и изолированных контейнеров singularity. Данная инсталляция будет использоваться сотрудниками факультета информатики и вычислительной техники, а также математического факультета Ярославского государственного университета для выполнения вычислительных задач (таких как решение дифференциальных уравнений в частных производных с запаздыванием).

Необходимо автоматизировать начальную установку стека программного обеспечения, а также максимально упростить дальнейшую эксплуатацию всего кластера с учётом возможного роста количества вычислительных узлов.

Для более эффективного распределения ресурсов между пользователями необходимо настроить ограничения доступа (используя Slurm Account Manager) согласно их принадлежности к различным группам в базе данных на основе OpenLDAP. По завершению произвести тестирование с использованием бенчмарка LINPACK, сравнив результаты полученные при использовании изолированных контейнеров с результатами запуска непосредственно на вычислительных узлах (bare metal). В качестве демонстрации методов параллельного программирования показать решение уравнения теплопроводности для двумерного случая с использованием дистрибутива OpenMPI.

## Введение

### Об организации управления ресурсами кластера

Для высокопроизводительных вычислений (High Performance Computing или HPC) характерно быстрое развитие, а также экспоненциальный рост потребляемых приложениями вычислительных мощностей (59,7 GFlops/s в Июне 1993 против 148,600 TFlops/s в Ноябре 2019[1]). За доставку и распределение ресурсов между приложениями (и пользователями) отвечает так называемый менеджер рабочих ресурсов (Resource and Job Management system). Он играет важную роль в управлении мощностями, так как он занимает стратегическое место во всем программном стеке, находясь между аппаратным и программным уровнями вычислительного кластера. Однако последние изменения в вышеупомянутых уровнях выявили новые сложности этого класса программного обеспечения. Таким вопросам, как масштабируемость, управление топологическими ограничениями, эффективность использования энергии и отказоустойчивость должно быть уделено особое внимание, дабы обеспечить лучшую эксплуатацию оборудования как с точки зрения системы, так и с точки зрения пользователя.

Наиболее известным (благодаря своей масштабируемости) и активно развивающимся продуктом в этой области на данный момент является slurm (Simple Linux Utility for Resource Management). При создании slurm авторы пытались избежать иерархического подхода к построению кластера, вместо этого была предложена система расширений и плагинов, что позволяет продумать архитектуру проекта любой сложности. Технология очередей (partitions) может быть использована для объединения узлов в группы по определённому набору свойств (например по техническим характеристикам). Взаимодействие между управляющим и вычислительными узлами происходит посредством сокетов, обычно через сеть Ethernet. Slurm поддерживает как однопоточное, так и параллельное выполнение задач. Поддерживаются наиболее распространённые реализации MPI (такие как MPICH, MVAPICH, OpenMPI, HP-MPI, IntelMPI). Такое понятие как массивы заданий (array jobs) позволяет планировать множественный запуск одного и того-же процесса с изменёнными параметрами одновременно. В целях безопасности все запускаемые задания могут быть запущены либо с использованием демона аутентификации munged, либо с использованием сертификата X509. В целях эффективности, slurm предоставляет различные политики планирования, такие как backfill, fairsharing и preemption. Slurm также является единственным планировщиком который поддерживает т.н. gang scheduling алгоритм (одновременный запуск связанных между собой процессов на разных процессорах).

### О Singularity

Unix-подобные операционные системы исторически разделены разработчиками на две основных составляющих — пространство компонентов ядра (kernel space) и пространство для работы пользовательских процессов (user space). Ядро поддерживает пространство пользователя взаимодействуя с оборудованием, предоставляя ключевые системные функции, а также создавая слой обратной совместимости программного обеспечения. User space пред-

ставляет собой наиболее привычное для конечного пользователя окружения, так как все приложения, библиотеки и сервисы запускаются именно там.

Контейнеры позволяют решить проблемы безопасности связанные с выполнением задачи непосредственно на вычислительном узле кластера (такие как например произвольный доступ к некоторым подсистемам ядра ОС), превращая пространство пользователя в набор взаимно заменяемых компонентов. На практике это означает, что вся пользовательская составляющая Linux, включая программы, собственные файлы конфигурации и окружение могут быть изменены или полностью заменены на новые в процессе выполнения. Singularity упрощает процесс использования и обслуживание контейнеров сводя всё окружение к одному и тому же проверяемому файлу.

Это позволяет пользователям создавать своё собственное окружение на основе той операционной системы (или дистрибутива), который полностью удовлетворяет их потребностям, получая при этом абсолютно изолированную среду для тестирования и/или разработки, естественным образом исчезает необходимость отслеживания каких-либо внешних факторов влияющих на рабочие процессы (например зависимостей для сборки проекта и т.д.).

Singularity предоставляет функциональность виртуальной машины без «тяжеловесной» реализации и, затрат связанных с производительностью и избыточностью.

В то время как в наши дни существует множество решений для контейнеризации (lxc, docker и т.д.), в singularity можно отметить следующие архитектурные решения которые выделяют его среди остальных:

**Воспроизводимый программный стек:** данные хранимые в singularity легко проверяются благодаря наличию контрольных сумм, а также с помощью криптографических подписей, всё это не требует никаких изменений со стороны пользователя (например предварительного разделения данных хранимых на диске на архивы). По умолчанию singularity использует так называемый формат image (с англ. image — снимок архива) для распространения готовыхборок. Присутствует совместимость с другими популярными форматами данных, например с image файлами docker.

**Мобильность вычислений:** singularity контейнеры легко переносятся с машины на машину при использовании стандартных средств операционной системы (rsync, scp, gridftp, http, NFS, и т.д.).

**Безопасная модель использования:** В отличие от множества других систем контейнеризации разработанных, для запуска доверенных контейнеров от доверенных пользователей, singularity в свою очередь был разработан с учётом ситуации, когда абсолютно любой пользователь имеет возможность использовать любое окружение без опасений негативно повлиять на хост систему.

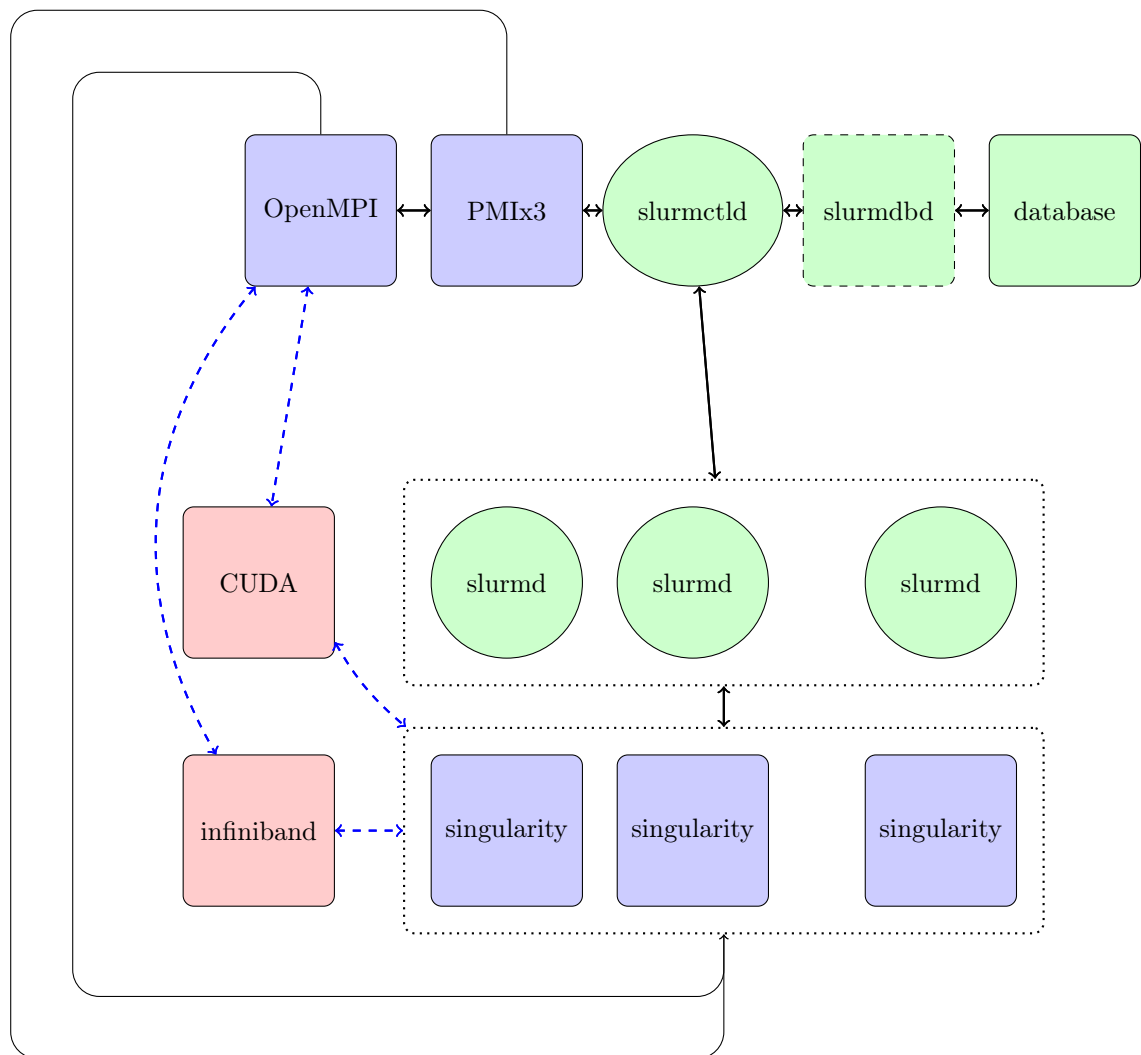
## Используемые программные средства

Кластер находится под управлением ОС GNU/Linux Ubuntu 18.04 (LTS). Сборка всех пакетов, необходимых для функционирования систем автоматизирована shell-скриптом, написанным на языке программирования bash. Компиляция производится стандартным для GNU/Linux набором утилит GCC с использованием системы сборки пакетов autotools. Сборка singularity производится бинарным компилятором golang-amd64. Инициализация, а также по-

следующая синхронизация пользователей из ldap в Slurm Accounting Manager происходит при помощи скрипта, написанного на python3 с использованием библиотеки python3-ldap. Для автоматизации управления инфраструктурой всего кластера используется менеджер конфигурации saltstack. Для проверки и обеспечения целостности данных передаваемых демонами slurm внутри сети, используется так называемый демон MUNGE, который производит авторизацию запросов, поступающих от пользователя по средствам проверки ключевых файлов на каждом узле. Далее необходимо сгенерировать

# 1 Теоретическая информация

## 1.1 Описание архитектуры кластера



**Рис. 1** Схема взаимодействия компонентов кластера

Согласно рис. 1 введём следующие обозначения:

- — компоненты являющиеся непосредственно частью slurm
- — компоненты являющиеся необходимыми для каждого узла
- — опциональные компоненты (наличие зависит от технического оснащения хоста)

### **slurmctld (slurm control daemon)**

Демон-контролёр. Отслеживает процессы и демоны slurm, управляет задачами (принимает, отклоняет и т.д), а также контролирует потребление всех ресурсов кластера.

### **slurmd (slurm daemon)**

Вычислительный демон. Производит периодический мониторинг всех задач, а также принимает и запускает задания от slurmctld или завершает задания по запросу от пользователя.

### **database**

Реляционная база данных, поддерживаемая slurmdbd и хранящая в себе все данные о пользователях из slurm accounting manager. В данной работе используется MySQL server 5.6.x.

### **slurmdbd (slurm database daemon)**

Многопоточный демон для передачи, обработки и хранения данных в базе данных slurm accounting manager.

### **OpenMPI**

Свободная реализация интерфейса обмена сообщениями (Message Passing Interface). Включает в себя также набор стандартных компиляторов для таких языков как C/C++ или FORTRAN. Позволяет писать переносимый код для выполнения параллельных вычислений на узлах кластера.

### **PMIx3**

Представляет собой программный интерфейс (API) для запуска параллельных программ написанных с использованием MPI через slurm srun. Подробная спецификация доступна в [2] и [3].

### **Infiniband (ofed/rdma-core)**

Высокоскоростной стандарт передачи данных в сетях HPC. Физически интерфейс представляет из себя карту расширения шины PCI. Интерфейс может быть также объединён в коммутируемую компьютерную сеть. Программно управляется с помощью отдельного модуля ядра Linux, загружаемого с помощью фреймворка dkms. На данный момент может работать с дистрибутивами rdma-core или Mellanox OpenFabrics Enterprise Distribution. Подробнее можно узнать в [4] и [5].

### **CUDA**

Дистрибутив Nvidia CUDA, позволяющий запускать параллельные программы с использованием ядер видеокарты (GPU). Включает в себя компилятор nvcc.

### **singularity**

Непосредственно singularity. Для изолированного запуска приложений, контейнер должен содержать в себе копию всего программного стека установленного на хосте. Подробнее о способах запуска можно узнать в [6].

Всего в кластере как видно из табл. 1 находится 22 узла. control1 является управляющим хостом с запущенными на нём демонами slurmdbd и slurmctld, остальные же являются подчинёнными хостами с запущенным на них демоном slurmd. Выбор версии cuda (8 или 10), обусловлен моделью GPU.



Распределение опциональных компонентов по узлам кластера

Хост	cuda 8	cuda 10	infiniband
cnode[1-7]	+	-	-
dnode[01-08]	-	+	+
dnode[09-14]	-	-	+
control1	-	+	+

## 1.2 Менеджер конфигурации saltstack

Salt, или SaltStack — это инструмент удалённого выполнения и система управления конфигурацией. Позволяет администраторам запускать команды на разных машинах. Функция управления конфигурацией создаёт модель клиент-сервер для быстрого, лёгкого и безопасного подключения компонентов инфраструктуры в соответствии с заданной политикой. Структура управления Salt довольно проста. В типичной установке есть только два разных класса машин.

**Salt master** — это машина, которая управляет кластером и определяет политики для подчинённых серверов. Мастер работает как репозиторий данных конфигурации и центр управления, который иницирует удалённые команды и приводит другие машины в требуемое состояние. Для обеспечения этой функциональности на мастер-сервер устанавливается демон salt-master. Управлять инфраструктурой можно и без мастера, но большинство настроек используют расширенные функции salt master. Для управления большими инфраструктурами salt может делегировать определённые компоненты и задачи, обычно связанные с мастером, на выделенные серверы. Он также может работать в многоуровневой конфигурации, где команды могут быть переданы через мастер-машины более низкого уровня.

**Salt minion** — ведомые серверы. На каждый ведомый сервер для связи с мастером устанавливается демон salt-minion. Миньон отвечает за выполнение команд, отправленных мастером, сообщает о результате заданий и предоставляет данные о базовом хосте.

### 1.2.1 Взаимодействие компонентов salt

Мастер и миньоны salt по умолчанию взаимодействуют через библиотеку обмена сообщениями zeroMQ. Она обеспечивает чрезвычайно высокую пропускную способность сети между сторонами, позволяя salt отправлять сообщения и данные с высокой скоростью. Поскольку zeroMQ является библиотекой, а не независимым сервисом, эта функциональность встроена в демоны salt-master и salt-minion.

При использовании zeroMQ salt поддерживает систему открытых ключей для аутентификации мастеров и миньонов. При первой загрузке миньон генерирует пару ключей и отправляет свои учётные данные на мастер-сервер, от которого он зависит. Затем мастер может принять этот ключ после проверки

миньона. После этого обе стороны могут быстро и безопасно обмениваться данными с помощью zeroMQ, зашифрованной ключами.

### 1.2.2 Файлы состояний

Salt предлагает способ управления узлами, с помощью которого необходимо описать состояние, в котором миньон должен находиться. Этот вид конфигурации называется salt state (состояние), а методология обычно называется управлением конфигурацией. Состояния определяются в файлах состояний. После того, как состояния миньонов описаны, они применяются к миньону.

Файлы состояний — это просто наборы словарей, списков, строк и чисел описанных в файлах с расширением .sls, которые затем подвергаются интерпретации в salt. По умолчанию для представления используется синтаксис yaml, но возможны описания с использованием json или python. Файлы состояний обычно хранятся в файловой системе мастера, но они также могут храниться в других местах файлового сервера, например, в репозитории git. В дополнение к ручному применению состояний к миньонам, salt предоставляет возможность автоматически отобразить, какие состояния должны применяться к различным миньонам. Это называется top-файлом. Примеры как top-файла, так и состояний описаны далее в пункте 2.3.

## 2 Практическая реализация

### 2.1 Сборка пакетов из исходного кода

Сборка готовых к установке программ производится скриптом из приложения А. В качестве единственного аргумента скрипт принимает путь к директории содержащей в себе архивы исходных кодов всех необходимых пакетов (в формате tar). Далее над каждым архивом производятся следующие операции:

1. распаковка во временную директорию /tmp
2. передача параметров конфигурации скрипту configure
3. компиляция
4. создание deb пакета

В скрипте соблюдена строгая последовательность сборки, так как некоторые собираемые пакеты, являются необходимыми зависимостями для сборки последующих. После завершения работы скрипта, готовые к распространению пакеты находятся в директории /tmp/deb\_packages, откуда их можно поместить в заранее настроенный РРА<sup>1</sup> - репозиторий для последующей установки на узлы кластера средствами slatstack.

### 2.2 Подготовка конфигурации slurm

Для конфигурации slurm необходимо создать два файла: slurm.conf (одинаков и находится на всех узлах кластера без исключения) и slurmdbd.conf (находится только на управляющем узле). Примеры реальных конфигураций находятся в приложении Б и приложении В соответственно. В файле конфигурации slurm.conf наибольший интерес представляют следующие директивы:

1. **SlurmctlHost** — Адрес узла, который является мастером (в данном случае достаточно просто указать имя хоста, так как разрешение имён происходит в локальной сети)
2. **MpiDefault** — Тип используемого MPI по умолчанию, согласно рис. 1 PMIxV3
3. **ClusterName** — Имя всего кластера в целом, данное имя должно быть указано при инициализации slurm accounting manager, этот момент является принципиально важным, так как в противном случае демоны slurmd, slurmdbd и slurmctld не синхронизируются между собой
4. **NodeName** — Список узлов с определёнными у них техническими характеристиками
  - 4.1. **RealMemory** — Реальный объём оперативной памяти каждого узла в мегабайтах
  - 4.2. **Sockets** — Количество CPU

---

<sup>1</sup>Personal Package Archive или персональный архив пакетов (англ.)

- 4.3. **CoresPerSocket** — Количество «ядер» для каждого процессора
- 4.4. **ThreadsPerCore** — Количество hyper-threading потоков для каждого ядра процессора
- 4.5. **State** — состояния узлов по умолчанию при запуске (UNKNOWN рекомендован по умолчанию, и означает что демон slurmd ожидает принятия команд)
- 5. **PartitionName** — Общее имя для группы хостов, позволяющее объединять несколько узлов в единое целое для выполнения задачи
  - 5.1. **Nodes** — Список узлов входящих в группу
  - 5.2. **MaxTime** — Временной лимит выполнения задач на группе (INFINITE — без ограничения)
  - 5.3. **State** — Состояние группы по умолчанию

В файле конфигурации slurmdbd.conf наибольший интерес в свою очередь представляют следующие директивы:

- 1. **StorageType** — Тип хранилища данных для slurm accounting manager
- 2. **StoragePass** — Пароль базы данных
- 3. **StorageUser** — Пользователь, имеющий доступ к базе данных
- 4. **StorageLoc** — Имя базы данных

## 2.3 Подготовка конфигурации saltstack

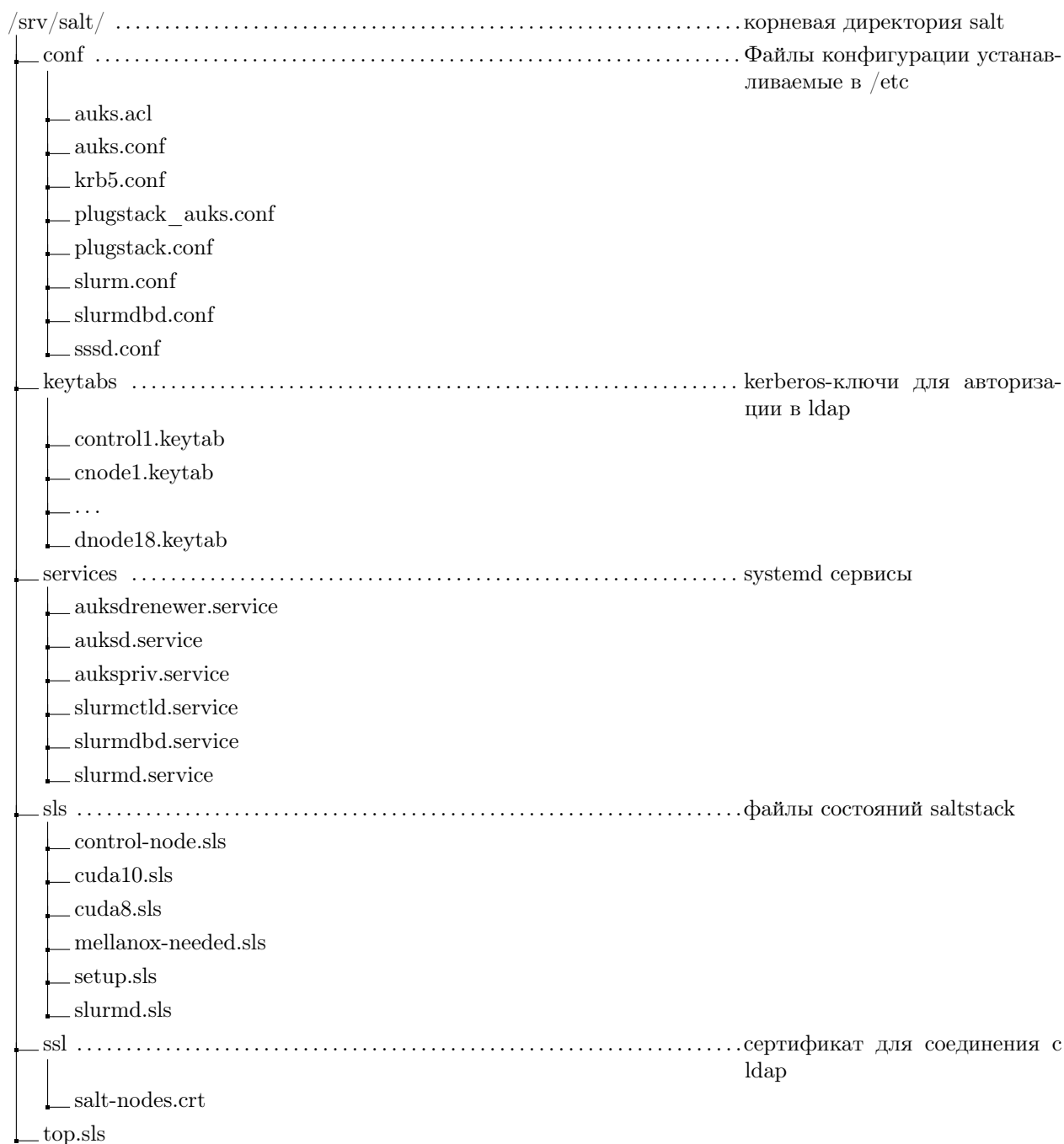


Рис. 1 Структура корня saltstack master (control1)

На рис. 1 изображена иерархия корня saltstack. В частности остановимся на необходимости наличия в ней директорий services и keytabs. services содержит в себе файлы необходимые для автоматического старта программ в режиме демона. Дабы избежать конфликтов с уже существующими в репозитории операционной системы пакетами, все программы собранные пользователем самостоятельно должны быть установлены в директорию /usr/local [7, стр. 21] (директория чаще всего указывается с помощью директивы `--prefix` во время конфигурации). Наиболее часто в service файлах поставляемых вместе с исходными кодами встречается так называемый общесистемный путь (а именно /usr/bin). Таким образом возникает необходимость предварительного внесения изменений в директиву `ExecStart` для актуализации путей к исполняемым файлам. Директория keytabs хранит в себе так называемые keytab файлы, необходимые для автоматической удалённой авторизации узлов в системе MIT Kerberos [8]. В данном случае kerberos используется для предотвращения несакционированного доступа к личным файлам в домашних директориях пользователей (все домашние директории хранятся на отдельном сетевом диске и подключаются уже непосредственно к кластеру средствами saltstack).

Далее приступаем к созданию корневого файла saltstack. Наличие данного файла строго обязательно, так как по сути своей он ставит в соответствие список узлов, и те файлы состояний которые к ним необходимо применить.

```
base:
  '*':
    - sls.setup
  'cnode(01|02|03|04|05|06|07).int.accelcomp.org':
    - match: pcre
    - sls.cuda8
  'control1.int.accelcomp.org':
    - match: pcre
    - sls.control-node
    - sls.cuda10
    - sls.mellanox-needed
  'dnode(01|02|03|04|05|06|07|08).int.accelcomp':
    - match: pcre
    - sls.cuda10
    - sls.mellanox-needed
  'dnode(09|10|11|12|13|14).int.accelcomp':
    - match: pcre
    - sls.mellanox-needed
  'not control1.int.accelcomp.org':
    - match: compound
    - sls.slurmd
```

Рис. 2 Структура файла top.sls

top.sls (рис. 2) поддерживает различные уровни описательной логики, в частности мы можем видеть pcre-совместимые регулярные выражения с применением логического «ИЛИ», а также применение логического «НЕ» для исключения узла master. Из приведённого выше файла можно заключить следующее:

- Ко всем узлам без исключения будет применён setup.sls (содержит настройки, необходимые для всех узлов кластера)
- К узлам cnode[01-07] будет применён cuda8.sls (установка CUDA 8)
- К узлам dnode[01-08] cuda10.sls и mellanox-needed.sls (установка CUDA 10 и Mellanox ofed для работы с infiniband соответственно)

- К узлам dnode[09-14] будет применён `mellanox-needed.sls`
- К узлу `control1` будут применены `cuda10.sls`, `mellanox-needed.sls` и `control-node.sls` (последний содержит в себе описание настроек базы данных для `slurmctld`)
- Ко всем узлам *кроме* `control1` будет применён `slurmd.sls` (содержит в себе описание установки `slurm` из локального репозитория, а также описание установки `slurmd.service`)

Для наглядности приведём и пошагово опишем некоторые примеры состояний из различных `.sls` файлов.

```
slurm_user:
  user.present:
    - name: slurm
    - fullname: Slurm
    - shell: /bin/false
    - home: /home/slurm
    - uid: 64030
    - gid_from_name: True
    - require:
      - group: slurm
```

**Рис. 3** Создание системного пользователя и группы `slurm` (`setup.sls`)

На рис. 3 представлен пример создания системного пользователя `slurm` средствами `saltstack`. Именно данный пользователь является владельцем прав на запуск процессов `slurmctld`, `slurmd` и `slurmdbd`. Также на основе пользователя создаётся группа с идентичным именем. И группе и пользователю присваиваются уникальные номерные идентификаторы (`user id` и `group id`) с номером 64030.

```
slurm accounting grants:
  mysql_grants.present:
    - grant: all privileges
    - database: slurm_acct_db.*
    - user: slurm
    - connection_user: root
    - connection_pass: RanDoMPassWord
    - connection_charset: utf8
```

**Рис. 4** Автоматическое назначение прав доступа в `mysql` (`control-node.sls`)

На рис. 4 показан пример состояния для назначения прав полного доступа `mysql` пользователю `slurm` к базе данных `slurm_acct_db` с использованием прав суперпользователя.

```
{% set hostname = salt.grains.get('host') %}

distribute krb5.keytab files:
  file.managed:
    - name: /etc/krb5.keytab
    - source: salt://keytabs/{{ hostname }}.keytab
```

**Рис. 5** Установка файлов `keytab` (`setup.sls`)

На рис. 5 описано состояние для установки keytab файлов. Так как имя каждого файла совпадает с именем хоста, на котором этот файл должен быть установлен, здесь вводится динамическая переменная `hostname`, которая предоставляет имя узла с именем файла.

```
mellanox specified packages:
  pkg.installed:
    - pkgs:
      - m4
      - swig
      - autoconf
      - quilt
      - libltdl-dev
      - gfortran
      - autotools-dev
      - libgfortran3
      - flex
      - automake
      - graphviz
      - tk
      - bison
      - tcl
      - chrpath
      - debhelper
      - dpatch

unpack mlx_ofed:
  archive.extracted:
    - name: /usr/local
    - source:
      ↪ /clusterhome/install/saltcluster/MLNX_OFED_LINUX-4.6-1.0.1.1-ubuntu16.04-x86_64.tgz

install mlx_ofed:
  cmd.run:
    - name: /usr/bin/perl
    ↪ /usr/local/MLNX_OFED_LINUX-4.6-1.0.1.1-ubuntu16.04-x86_64/mlnxofedinstall
    ↪ --force
    - runas: root
    - unless:
      - ls /usr/sbin/iblinkinfo
```

**Рис. 6** Структура файла `mellanox-needed.sls`

На рис. 6 представлен полный цикл установки `mellanox ofed` состоящий из трёх состояний.

- `mellanox specified packages` — Производит установку пакетов из указанного списка путём вызова функции `pkg.installed`
- `unpack mlx_ofed` — Распаковывает `.tgz` дистрибутив в директорию `/usr/local`
- `install mlx_ofed` — Производит запуск скрипта-установщика от имени суперпользователя при условии *отсутствия* файла `/usr/sbin/iblinkinfo` (в противном случае считается, что установка уже была произведена ранее и не требуется выполнять какие-либо действия)



## 2.4 Развёртывание кластера

Для развёртывания кластера необходимо выполнить ряд действий на узле `controll` (который является мастером как для `slurm`, так и для `saltstack`). Для развёртывания достаточно выполнить команду `salt '*' state.apply` от имени суперпользователя. Данную команду стоит понимать как «применить все файлы состояний ко всем узлам согласно файлу `top.sls`». Однако существуют и более гибкие способы применения состояний, например команда `salt 'cnod*' state.apply sls.cuda8` произведёт установку CUDA 8 на все узлы, имена которых содержат в себе шаблон `cnode`. Далее необходимо сгенерировать и распространить по всем узлам кластера ключ MUNGE для проверки валидности передаваемых данных, для этого выполняются следующие команды:

- `salt '*' file.remove /etc/munge/munge.key` — удаляем все индивидуальные ключи, созданные каждым узлом при первом запуске, так как они не идентичны
- `echo -n "foo" | sha512sum | cut -d' ' -f1 >/etc/munge/munge.key` — генерируем новый ключ, вместо `foo` может быть строка любой длины из любых символов (включая UTF-8)
- `salt-cp '*' /etc/munge/munge.key /etc/munge` — копируем новый ключ в директорию `/etc/munge` каждого узла
- `salt '*' file.chown /etc/munge/munge.key munge munge` — назначаем владельцем ключа пользователя и группу `munge`
- `salt '*' cmd.run 'chmod 400 /etc/munge/munge.key'` — редактируем права на файл в целях безопасности (пользователь может читать, группа и остальные не имеют доступа)

Перезагрузка всего кластера производится командой `salt '*' system.reboot`, после выполнения перезагрузки все службы кластера стартуют в автоматическом режиме.

### 2.4.1 Синхронизация пользователей ldap в slurm accounting

Синхронизация производится с помощью скрипта, представленного в приложении Г. При первом запуске необходимо указать в качестве единственного опционального аргумента имя кластера, которое должно совпадать с именем определённым ранее в файле конфигурации `slurm.conf` директивой `ClusterName`. Также при первом запуске `slurm accounting manager` создаёт в базе данных `mysql` две группы `regular` и `power`. На ряду с этим производится инициализация так называемого QoS (Quality of Service) правила с названием `short_term` и параметром `WallTime`. Данное правило может быть применено к пользователю или группе и задаёт максимально возможное время выполнения задачи (в нашем случае 10 минут). Скрипт подключается напрямую к базе `OpenLDAP`, в которой существует 4 группы пользователей: `uni`, `асс`, `power` и `regular` (к последней принадлежат все пользователи без исключения, любой пользователь может находиться более чем в одной группе одновременно). Согласно своей принадлежности к группам `ldap`, пользователи получают следующие права доступа в `slurm`

- Пользователи группы uni получают доступ к очереди debug, а также short\_term QoS
- Пользователи группы acc получают short\_term QoS, доступ к любой очереди без ограничений
- Пользователи группы power не имеют каких-либо ограничений

User	Account	Partition	QoS
sglyzin	regular	debug	normal,short_term,standby
zlogene	power		normal,standby
zlogene	regular	debug	normal,short_term,standby
zlogene	regular		normal,short_term,standby
zlogene	regular	uni	normal,short_term,standby

**Рис. 7** Пример вывода команды `sacctmgr`

Далее убедимся в правильности работы slurm accounting выполнив команду `sacctmgr show user zlogene,sglyzin -s format=User,Account,Partition`

Анализируя данные с рис. 7 можно заключить следующее:

- Пользователь `sglyzin` принадлежит к группе `regular`, имеет доступ к очереди `debug`, также для него установлено временное ограничение на выполнение задач
- пользователь `zlogene` принадлежит к группам `regular` и `power`, имеет формальный доступ к очередям `debug` и `uni` с временным ограничением

Стоит отметить, что при принадлежности одного пользователя к нескольким группам в slurm accounting, пользователь в праве сам выбирать какую из групп ему использовать для выполнения задачи. Таким образом, хоть пользователь `zlogene` и имеет ограничения, наложенные на него группой `regular`, они могут быть проигнорированы при использовании им группы `power`.

При запуске скрипта без аргумента происходит синхронизация пользователей в существующую структуру slurm accounting (синхронизация без повторной инициализации демона `slurmdbd`). После завершения работы скрипта необходимо перезагрузить кластер командой `salt '*' system.reboot` и удостовериться в правильности работы `slurmctld` проверив вывод команды `systemctl status slurmctld` (в выводе должны отсутствовать какие-либо ошибки).

## 3 Тестирование производительности

### 3.1 Подготовка к тестированию

Для тестирования производительности кластера, будем использовать набор тестов производительности адаптированных для запуска на распределённых вычислительных системах — HPL (High Performance Linpack) для тестирования производительности CPU. Данный инструмент основан на широко известном наборе тестов производительности LINPACK. Суть тестирования заключается в решении системы линейных алгебраических уравнений путём так называемой LU факторизации, подробный алгоритм решения подобных систем описан в [9].

Стандартный архив HPL содержит в себе ряд заранее подготовленных файлов make, любой из них необходимо привести к виду показанному на рис. 1.

```
ARCH      = amd64
TOPdir    = $(HOME)/hpl
MPdir     = /usr/mpi/gcc/openmpi-4.0.3
MPinc     = -I $(MPdir)/include
MPlib     = $(MPdir)/lib/libmpi.so
LAdir     = /usr/lib/x86_64-linux-gnu/blas/
LAlib     = $(LAdir)/libblas.a
```

Рис. 1 пример настройки директив для файла make

- ARCH — Архитектура, под которую производится сборка
- TOPdir — Путь к директории с исходным кодом hpl (в данном случае абсолютный, т.к. мы находимся внутри данной директории непосредственно во время сборки)
- MPdir — Путь к корню установки OpenMPI
- MPinc — Путь к заголовочным файлам
- MPlib — Путь и имя основной библиотеки MPI
- LAdir — Путь к библиотеке алгебры для работы с векторными и матричными операциями (пакет libblas-dev был установлен заранее)
- LAlib — Имя библиотеки BLAS

Далее достаточно запустить команду `make arch=amd64`, бенчмарк будет скомпилирован с использованием компилятора `mpicc` и размещён в директории директории `$(TOPdir)/bin` под именем `xhpl` вместе с файлом настроек `HPL.dat`. Данный файл настраивается индивидуально в соответствии с параметрами тестируемых узлов. В нашем случае тестирование производительности будет производиться на узлах `dnode01-dnode08`, так как там установлено наиболее новое оборудование для поддержки infiniband. Файл используемый для тестирования представлен в приложении Д. Наибольший интерес для модификации пользователем представляют параметры `Ps`, `Qs` и `Ns`. Произведение `Ps` и `Qs` должно давать количество процессоров, участвующих в запуске, в нашем случае 8 узлов 2 сокета в каждом, соответственно допустимые значения  $Ps \times Ns = 4 \times 4 = 16 \times 1 = 16$  (могут задаваться пары, но

не тройки), значение  $N_s$  задаёт размерность решаемой матрицы, в данном случае (и чаще всего на практике) оно подбирается эмпирическим путём.

### 3.2 Проведение тестирования

Для проведения тестирования был заранее подготовлен образ Docker, включающий в себя дистрибутивы OpenMPI и Mellanox OFED с версиями, идентичными версиям установленным на узле control1. По сути всё тестирование сводится к запуску двух команд (оценка производительности происходит автоматически по результатам тестирования):

```
- srun -p all -N 8 -w dnode[01-08] ./xhpl
```

Данная команда запустит файл ./xhpl непосредственно на узлах dnode[01-08]. Результат тестирования представлен на рис. 2

```
- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
  ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be          1.110223e-16
- Computational tests pass if scaled residuals are less than    16.0

=====
T/V              N    NB    P    Q              Time              Gflops
-----
WR12R2R4        115584  192    4    4          1681.87          1.943e+3
HPL_pdgesv() start time Sat Jun 06 15:20:17 2020

HPL_pdgesv() end time   Sun Jun 07 19:31:31 2020

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=          0.0028942 ..... PASSED
=====

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.

-----

End of Tests.
```

**Рис. 2** результат тестирования вне контейнера (bare metal)

```
- srun -p all -N 8 singularity exec benchamrking-ubuntu-image ./xhpl
```

Данная команда запустит файл ./xhpl непосредственно на узлах dnode01 — dnode08, но процесс обчёта задачи будет выполняться непосредственно внутри контейнера. Результат тестирования представлен на рис. 3.

```

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
  ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be 1.110223e-16
- Computational tests pass if scaled residuals are less than 16.0

=====
T/V              N      NB      P      Q              Time              Gflops
-----
WR12R2R4        115584   192      4      4              1800.21              1.910e+3
HPL_pdgesv() start time Sat Jun 07 16:15:17 2020

HPL_pdgesv() end time   Sun Jun 08 22:30:00 2020

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 0.0028942 ..... PASSED
=====

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.

-----

End of Tests.

```

**Рис. 3** результат тестирования вне контейнера (singularity ubuntu)

Сопоставив оба результата можно заметить, что производительность (не смотря на время выполнения) различается несущественно (1943 Gflops в условиях реального окружения против 1910 Gflops в контейнере).

## 4 Решение уравнения теплопроводности

### 4.1 Общие сведения

$$\frac{\partial T}{\partial t} = \kappa \Delta T \quad \text{где} \quad \Delta = \sum_{i=1}^n \frac{\partial^2}{\partial x_i^2} \quad (1)$$

$\kappa$  является коэффициентом теплопроводности. Таким образом уравнение теплопроводности для двумерного случая может быть записано как

$$\frac{\partial \theta}{\partial t} = \kappa \left( \frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} \right) \quad (2)$$

Мы воспользуемся FTCS<sup>2</sup>[10] схемой расчёта, которая является частью метода конечных разностей и зачастую используется для численного решения уравнения теплопроводности (наряду с решением параболических уравнений в частных производных). Эта схема основана на Чобы упростить нотацию, для записи двумерной дискретизации по  $j$ ,  $k$  в момент времени  $n$  обозначим её как  $\theta_{i,j}^n$ .

### 4.2 Пространственная дискретизация

$$\theta_{m+1} = \theta(x_m + h) = \theta(x_m) + h\theta'(x_m) + \frac{h^2}{2}\theta''(x_m) + \frac{h^3}{6}\theta'''(x_m) + O(h^4) \quad (3)$$

$$\theta_{m+1} = \theta(x_m - h) = \theta(x_m) - h\theta'(x_m) + \frac{h^2}{2}\theta''(x_m) + \frac{h^3}{6}\theta'''(x_m) + O(h^4) \quad (4)$$

Складывая (3) и (4) получим выражение для второй производной в точке  $x_m$ :

$$\theta''(x_m) = \frac{\theta_{m+1} - 2\theta_m + \theta_{m-1}}{h^2} + O(h^2) \quad (5)$$

В двумерном случае мы получаем те же отношения для переменной  $y$  взяв  $h_x = \text{size}_x / N_x$ ,  $h_y = \text{size}_y / N_y$  и учитывая соглашение  $\theta(x_i, y_j) = \theta_{i,j}$  имеем:

$$\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} = \frac{\theta_{i+1,j} - 2\theta_{i,j} + \theta_{i-1,j}}{h_x^2} + \frac{\theta_{i,j+1} - 2\theta_{i,j} + \theta_{i,j-1}}{h_y^2} \quad (6)$$

---

<sup>2</sup>Forward-Time Central-Space

### 4.3 Временная дискретизация

Правый член (2) может быть записан как:

$$\frac{\partial \theta_{i,j}^n}{\partial t} = \frac{\theta_{i,j}^{n+1} - \theta_{i,j}^n}{\Delta t} \quad (7)$$

Таким образом комбинируя (2) и (6) рекуррентная формула примет вид:

$$\theta_{i,j}^{n+1} = \theta_{i,j}^n + \kappa \Delta t \left[ \frac{\theta_{i+1,j}^n - 2\theta_{i,j}^n + \theta_{i-1,j}^n}{h_x^2} + \frac{\theta_{i,j+1}^n - 2\theta_{i,j}^n + \theta_{i,j-1}^n}{h_y^2} \right] \quad (8)$$

### 4.4 Критерий сходимости

Критерий сходимости связывает точное решение уравнений с численно вычисленным решением. В случае когда у нас имеется  $h_x = h_y = h$  уравнение (8) устойчиво если:

$$\Delta t \leq \frac{h^2}{4\kappa} \quad (9)$$

С учётом шага пространственной дискретизации это условие устойчивости даёт оценку выбора шага по времени  $\Delta t$ :

$$\Delta t = \frac{1}{4\kappa} \min(h_x, h_y)^2 \quad (10)$$

### 4.5 MPI реализация

#### 4.5.1 Декартова топология процесса

Мы разобьём каждую область (также называемую сеткой) на подобласти, назначая каждой свой процесс (с помощью стандартной функции `MPI_Cart_create`[11]). Выигрыш во времени выполнения достигается за счёт того, что итеративная схема вычислений выполняется параллельно, а обновления для всех 4х сторон подобласти выполняются с взаимодействием окружающих процессов: мы назвали это разбиение "декартовой топологией" потому что подобласти имеют квадратную или прямоугольную форму. Ниже можно видеть пример разбиения области с использованием 8 процессами ( $x\_domains \times y\_domains = 2 \times 4 = 8$ ).

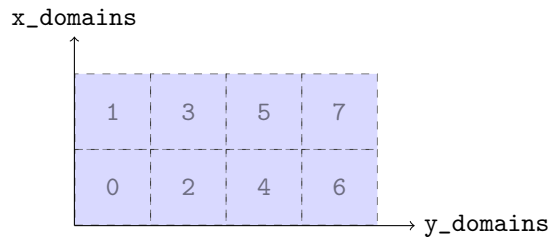


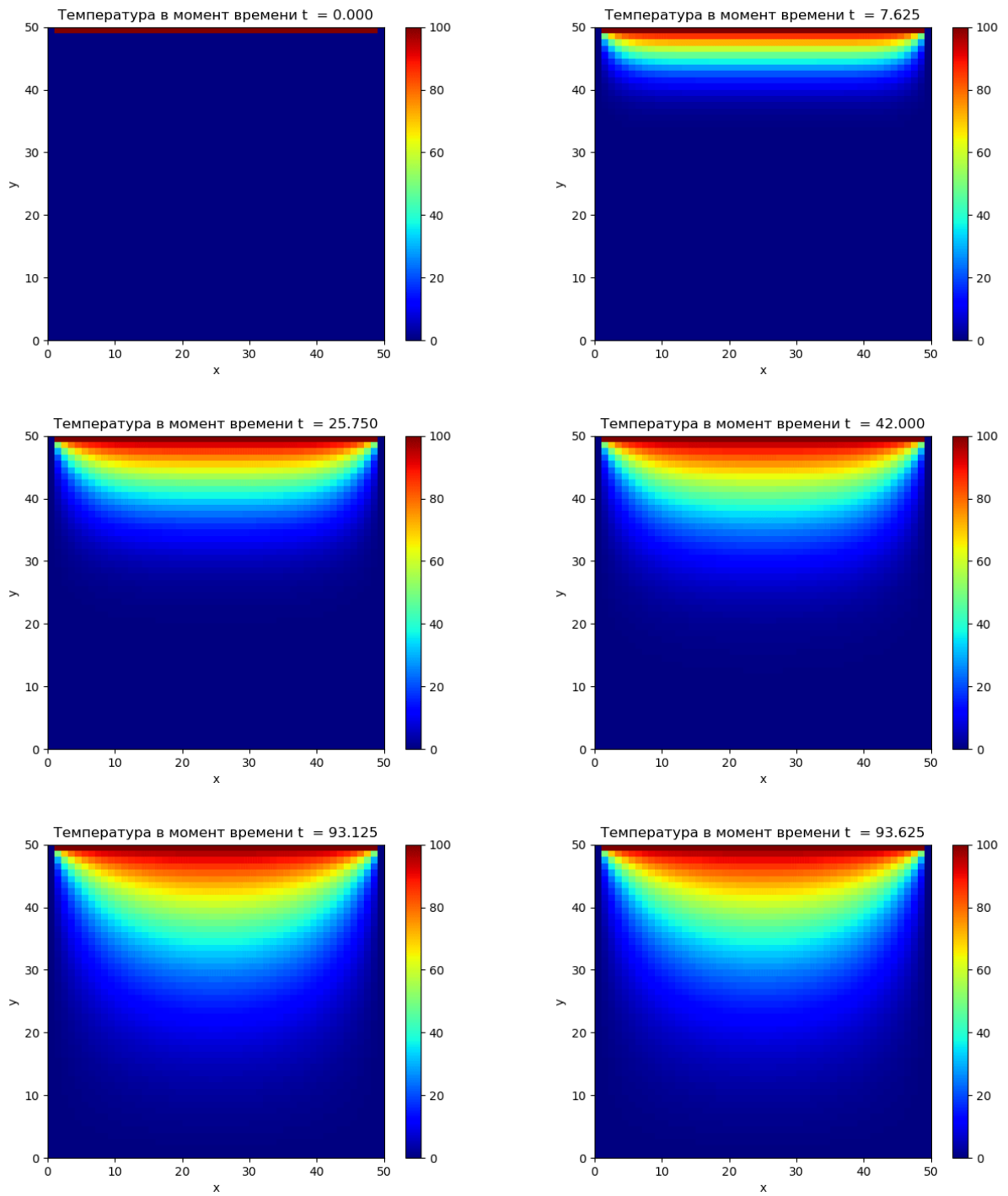
Рис. 1 Пример картезианского разбиения области

#### 4.5.2 Межпроцессорное взаимодействие

Параллелизация подразумевает под собой коммуникацию между процессами внутри каждой подобласти. Как можно видеть в приложении Ж, для каждого итеративного с рангом `ime` мы вычисляем следующее значение для соответствующей подобласти с использованием функции `computeNext`. Затем мы используем функцию `updateBound` для обновления значений строк и столбцов соответствующих процессов находящихся вокруг текущего процесса `ime`.



## 4.6 Графические результаты



**Рис. 2** Графическое отображение решения в момент времени  $t$

В качестве тела для нагрева была взята тонкая квадратная пластина со стороной в 50 единиц длины. Температура внутри пластины равна 0 (в момент времени  $t = 0$ ). Для данной модели были также взяты  $h = 1$  и  $\kappa = 2.0$

## Заключение

В данной работе был продемонстрирован один из методов развёртывания вычислительного кластера. В ходе работы была полностью решена проблема автоматизации процесса развёртывания с возможностью дальнейшего масштабирования, разработана документация для облегчения дальнейшего обслуживания, а также ряд инструментов облегчающих выполнение задач администрирования в будущем. Был произведён сравнительный анализ результатов тестирования и показан один из способов решения уравнения теплопроводности. Кластер был запущен в эксплуатацию и используется сотрудниками Ярославского государственного университета.

## Список литературы

- [1] Erich Strohmaier Jack Dongarra Horst Simon Martin Meuer, The TOP500 project, URL: <https://www.top500.org/lists/> (дата обращения: 29.04.21)
- [2] Artem Y. Polyakov, Joshua S. Ladd, Boris I. Karasev, Slurm PMIx support, URL: [https://slurm.schedmd.com/SC17/Mellanox\\_Slurm\\_pmix\\_UCX\\_backend\\_v4.pdf](https://slurm.schedmd.com/SC17/Mellanox_Slurm_pmix_UCX_backend_v4.pdf) (дата обращения: 27.04.21)
- [3] Morris Jette, Artem Y. Polyakov, Tim Wickberg. MPI and UPC Users Guide, URL: [https://slurm.schedmd.com/mpi\\_guide.html](https://slurm.schedmd.com/mpi_guide.html) (дата обращения: 27.04.21)
- [4] Gil Bloch, MPI over InfiniBand, URL: [https://www.open-mpi.org/papers/workshop-2006/thu\\_01\\_mpi\\_on\\_infiniband.pdf](https://www.open-mpi.org/papers/workshop-2006/thu_01_mpi_on_infiniband.pdf) (дата обращения: 28.04.21)
- [5] Виктор Гурылев, Infiniband: матрица для данных, URL: <https://habr.com/ru/company/intel/blog/154339> (дата обращения: 28.04.21)
- [6] Felip Moll, Tim Wickberg. Singularity and MPI applications, URL: <https://sylabs.io/guides/3.3/user-guide/mpi.html> (дата обращения: 27.04.21)
- [7] Rusty Russell, Daniel Quinlan, Linux Foundation inc.. Filesystem Hierarchy Standard [2004 — ], URL: [https://refspecs.linuxfoundation.org/FHS\\_3.0/fhs-3.0.pdf](https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf) (дата обращения 17.05.21)
- [8] MIT Kerberos Consortium, MIT Kerberos Documentation/keytab, URL: [https://web.mit.edu/kerberos/krb5-devel/doc/basic/keytab\\_def.html](https://web.mit.edu/kerberos/krb5-devel/doc/basic/keytab_def.html) (дата обращения 20.05.21)
- [9] Jack J. Dongarra, Piotr Luszczek, Antoine Petit, The LINPACK Benchmark: Past, Present, and Future / 2002 , URL: <http://www.netlib.org/utk/people/JackDongarra/PAPERS/hplpaper.pdf> (дата обращения 25.05.21)
- [10] Gerald Recktenwald. FTCS Solution to the Heat Equation, URL: [http://web.cecs.pdx.edu/~gerry/class/ME448/lecture/pdf/FTCS\\_slides.pdf](http://web.cecs.pdx.edu/~gerry/class/ME448/lecture/pdf/FTCS_slides.pdf) (дата обращения 23.05.21)
- [11] The OpenMPI Project, Open MPI v4.1.1 documentation, URL: <https://www.open-mpi.org/doc/v4.1> (дата обращения 23.05.21)

## Скрипт сборки пакетов

```

1  #!/usr/bin/env bash
2
3  # The script accepts a directory
4  # with needed tarballs as an argument
5  # so make sure that you put everything you need there
6  # before running the script.
7  # Do not also forget to track down
8  # the $mpi3_prefix and $mpi4_prefix on OpenMPI version changes
9
10 if [[ ${UID} -ne 0 ]]; then
11     echo "You must be logged in as root to run this script!"
12     exit 1
13 fi
14
15 # temporary directory for deb files
16
17 mkdir -p /tmp/deb_packages
18
19 # shut up perl nagging
20 export LC_ALL=C
21 export LANGUAGE=C
22
23 # shut up slurm mysql nagging
24
25 export HAVEMYSQLCONFIG=/usr/bin/mysql_config
26 export ac_have_mysql=yes
27
28 apt-get install -y libkrb5-dev bison \
29     libevent-dev munge libmunge-dev flex libssl-dev uuid-dev \
30     checkinstall m4 libnuma-dev gcc-multilib gfortran > /dev/null 2>&1
31
32 tmpdir=$(mktemp -d /tmp/slurm-build-XXX --suffix=-(date +%m-%d-%y))
33
34 basic_prefix="/usr/local"
35 mpi4_prefix="/usr/mpi/gcc/openmpi-4.0.1"
36 mpi3_prefix="/usr/mpi/gcc/openmpi-3.1.3"
37 mpi1_prefix="/usr/mpi/gcc/openmpi-1.6.5-32"
38 toolchain_prefix="/usr/local/autotools"
39
40 if [[ -z "$1" ]]; then
41     echo "Oops... you seem to forget to point a dir!"
42     exit 1
43 fi
44
45 run_binary_build() {
46     make -j$(nproc)
47     checkinstall -y --pakdir=/tmp/deb_packages --install=no \
48         --pkgname=$1 --pkgversion=$2
49 }
50
51 # The run_binary_build() function
52 # accepts two optional arguments
53 # that can be used when we need
54 # few versions of the same package
55 # installed at the same time
56 # or neither version nor name of
57 # a package is not clear for checkinstall
58 # look at either singularity or hwloc or openmpi examples

```

```

59
60 run_source_build() {
61     make -j$(nproc)
62     make install
63
64 }
65
66 # The run_source_build() function is only needed for
67 # packages that we do not want/need to convert to
68 # the binary packages but need them to build
69 # binaries (libtool/autoconf/automake/m4 and so on)
70
71 cd $1
72
73 for item in *tar*; do
74     tar xf ${item} -C ${tempdir}
75 done
76
77 # install cuda repositories
78 dpkg -i --force-overwrite cuda-repo-ubuntu1604-8-0-local-ga2_8.0.61-1_amd64.deb
79 dpkg -i --force-overwrite cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
80
81 apt-key add /var/cuda-repo-10-2-local-10.2.89-440.33.01/7fa2af80.pub
82 apt-key add /var/cuda-repo-8-0-local-ga2/7fa2af80.pub
83
84 apt-get update
85
86 cd ${tempdir}
87
88 files=( hwloc-1*
89         hwloc-2*
90         libevent*
91         m4*
92         autoconf*
93         automake*
94         libtool*
95         pmix*
96         slurm*
97         auks*
98         openmpi-1*
99         openmpi-3*
100        openmpi-4*
101        go*
102        singularity* )
103
104 echo "Just make sure that you have right packages to install"
105
106 echo ${files[@]}
107
108 read -n 1 -s -r -p "And then press any key to continue "
109
110 # copy go properly
111 mv -f ${tempdir}/go ${basic_prefix}/
112
113 # hwloc-1
114 if [[ -e "${files[0]}" ]]; then
115     pushd "${files[0]}"
116     ./configure --prefix="${basic_prefix}/hwloc-1" && run_binary_build "hwloc1" && popd
117 fi
118
119 # hwloc-2

```

```

120 if [[ -e "${files[1]}" ]]; then
121     pushd "${files[1]}"
122     ./configure --prefix="${basic_prefix}/hwloc-2" && run_binary_build "hwloc2" && popd
123 fi
124
125 # libevent
126 if [[ -e "${files[2]}" ]]; then
127     pushd "${files[2]}"
128     ./configure --prefix="${basic_prefix}/libevent" && run_binary_build "libevent" "2.1.10" && popd
129 fi
130
131 export PATH=${PATH}:/usr/local/autotools/bin
132
133 # m4
134 if [[ -e "${files[3]}" ]]; then
135     pushd "${files[3]}"
136     ./configure --prefix=${toolchain_prefix} && run_source_build && popd
137 fi
138
139 # autoconf
140 if [[ -e "${files[4]}" ]]; then
141     pushd "${files[4]}"
142     ./configure --prefix=${toolchain_prefix} && run_source_build && popd
143 fi
144
145 # automake
146 if [[ -e "${files[5]}" ]]; then
147     pushd "${files[5]}"
148     ./configure --prefix=${toolchain_prefix} && run_source_build && popd
149 fi
150
151 # libtool
152 if [[ -e "${files[6]}" ]]; then
153     pushd "${files[6]}"
154     ./configure --prefix=${toolchain_prefix} && run_source_build && popd
155 fi
156
157 # pmix
158 if [[ -e "${files[7]}" ]]; then
159     pushd "${files[7]}"
160     ./autogen.pl
161     ./configure --prefix="${basic_prefix}/pmix" --with-platform=optimized --with-libevent=/usr/local/libevent
162     ↪ --disable-pmi-backward-compatibility \
163     && run_binary_build "pmix" "3.1.4" && popd
164 fi
165
166 # slurm
167 if [[ -e "${files[8]}" ]]; then
168     pushd "${files[8]}"
169     ./configure --prefix="${basic_prefix}/slurm" --sysconfdir=/etc/slurm --without-ucx --without-freeipmi
170     ↪ --with-pmix="${basic_prefix}/pmix" \
171     --with-mysql_config=/usr/bin \
172     && run_binary_build && popd
173 fi
174
175 # auks
176 if [[ -e "${files[9]}" ]]; then
177     pushd "${files[9]}"
178     # temporary hack, makefiles may be changed in the future
179     sed -i 's/-lkrb5 -pthread/-lkrb5 -lpthread/g' src/api/auks/Makefile.am

```

```

179     sed -i 's/-lkrb5 -pthread/-lkrb5 -lpthread/g' src/api/auks/Makefile.in
180     sed -i 's/MANS = ${man1_MANS} ${man5_MANS} ${man8_MANS}/MANS = ${man1_MANS} ${man5_MANS}/g'
        ↪ doc/man/Makefile.in
181     sed -i 's/install-man: install-man1 install-man5 install-man8/install-man: install-man1 install-man5/g'
        ↪ doc/man/Makefile.in
182     ./configure --prefix="${basic_prefix}/auks" --sysconfdir=/etc/auks --with-slurm="${basic_prefix}/slurm" \
183     && run_binary_build && popd
184 fi
185
186
187 # install cuda
188 apt-get --reinstall install -y cuda-8-0
189 apt-get --reinstall install -y cuda
190
191
192 # openmpi-1
193 if [[ ! -d "${file[10]}" ]]; then
194     pushd "${files[10]}"
195     ./configure --prefix=${mpi1_prefix} \
196     --disable-vt \
197     --build=i686-pc-linux-gnu \
198     --with-slurm="${basic_prefix}/slurm" \
199     && run_binary_build "openmpi1" && popd
200 fi
201
202 # openmpi-3 and openmpi-4 with cuda
203 if [[ -e "${files[11]}" ]]; then
204     pushd "${files[11]}"
205     ./configure --prefix=${mpi3_prefix} \
206     --with-slurm="${basic_prefix}/slurm" \
207     --with-libevent="${basic_prefix}/libevent" \
208     --with-hwloc="${basic_prefix}/hwloc-1" \
209     --with-pmix="${basic_prefix}/pmix" \
210     --with-cuda=/usr/local/cuda \
211     --without-ucx \
212     && run_binary_build "openmpi3" && popd
213 fi
214
215 if [[ -e "${files[12]}" ]]; then
216     pushd "${files[12]}"
217     ./configure --prefix=${mpi4_prefix} \
218     --with-slurm=${basic_prefix} \
219     --with-libevent="${basic_prefix}/libevent" \
220     --with-hwloc="${basic_prefix}/hwloc-2" \
221     --with-pmix="${basic_prefix}/pmix" \
222     --with-cuda=/usr/local/cuda \
223     --without-ucx \
224     && run_binary_build "openmpi4" && popd
225 fi
226
227 # openmpi-3 and openmpi-4 without cuda
228 if [[ -e "${files[11]}" ]]; then
229     pushd "${files[11]}"
230     ./configure --prefix=${mpi3_prefix}_without_cuda \
231     --with-slurm="${basic_prefix}/slurm" \
232     --with-libevent="${basic_prefix}/libevent" \
233     --with-hwloc="${basic_prefix}/hwloc-1" \
234     --with-pmix="${basic_prefix}/pmix" \
235     --without-cuda \
236     --without-ucx \
237     && run_binary_build "openmpi3_without_cuda" && popd

```

```

238 fi
239
240 if [[ -e "${files[12]}" ]]; then
241     pushd "${files[12]}"
242     ./configure --prefix="${mpi4_prefix}_without_cuda" \
243         --with-slurm="${basic_prefix}/slurm" \
244         --with-libevent="${basic_prefix}/libevent" \
245         --with-hwloc="${basic_prefix}/hwloc-2" \
246         --with-pmix="${basic_prefix}/pmix" \
247         --without-cuda \
248         --without-ucx \
249         && run_binary_build "openmpi4_without_cuda" && popd
250 fi
251
252 export PATH=${basic_prefix}/go/bin:${PATH}:${GOPATH}/bin
253
254 # singularity
255 if [[ -e "${files[14]}" ]]; then
256     pushd "${files[14]}"
257     ./mconfig --prefix=/usr/local/singularity && \
258     cd ./builddir && run_binary_build "singularity" "3.5.3" && popd
259 fi
260
261 chmod 4755 ${basic_prefix}/singularity/libexec/singularity/bin/starter-suid
262
263
264 mkdir -p /tmp/clusterbuild
265
266 mv -f ${basic_prefix}/* /tmp/clusterbuild > /dev/null 2>&1
267 # we install cuda per node using saltstack
268 rm -fr /tmp/clusterbuild/cuda*
269 rm -fr /var/cuda*
270 rm -fr /tmp/clusterbuild/autotools
271 rm -fr /etc/apt/sources.list.d/*cuda*
272
273 echo "-----"
274 echo "compilation finished!"
275 echo "BINARY packages here: (add them to the repository)"
276 echo "/tmp/deb_packages"
277 echo "-----"
278
279 rm -fr ${tempdir}
280
281 apt-get remove -y --purge cuda* > /dev/null 2>&1

```



## Файл конфигурации slurm

```

1 SlurmctldHost=control1
2 DisableRootJobs=YES
3 MpiDefault=pmix_v3
4 PluginDir=/usr/local/slurm/lib/slurm
5 PlugStackConfig=/etc/slurm/plugstack.conf
6 ProctrackType=proctrack/linuxproc
7 ReturnToService=1
8 SlurmctldPidFile=/var/run/slurmctld.pid
9 SlurmctldPort=6817
10 SlurmdPidFile=/var/run/slurmd.pid
11 SlurmdPort=6818
12 SlurmdSpoolDir=/home/slurm/slurmd
13 SlurmUser=slurm
14 StateSaveLocation=/home/slurm
15 SwitchType=switch/none
16 TaskPlugin=task/affinity
17 TaskPluginParam=Sched
18 InactiveLimit=0
19 KillWait=30
20 MinJobAge=300
21 SlurmctldTimeout=120
22 SlurmdTimeout=300
23 Waittime=0
24 FastSchedule=1
25 SchedulerType=sched/builtin
26 SelectType=select/cons_res
27 SelectTypeParameters=CR_Core
28 AccountingStorageEnforce=limits
29 AccountingStoragePort=7031
30 AccountingStorageType=accounting_storage/slurmdbd
31 AccountingStoreJobComment=YES
32 ClusterName=cluster
33 JobCompType=jobcomp/none
34 JobContainerType=job_container/none
35 JobAcctGatherFrequency=30
36 JobAcctGatherType=jobacct_gather/none
37 SlurmctldDebug=debug5
38 SlurmctldLogFile=/tmp/slurmctld.log
39 SlurmdDebug=debug5
40 SlurmdLogFile=/tmp/slurmd.log
41 NodeName=dnode[01,02,03,04] RealMemory=64000 Sockets=2 CoresPerSocket=10 ThreadsPerCore=1 State=UNKNOWN
42 NodeName=dnode[05,06,07,08] RealMemory=128000 Sockets=2 CoresPerSocket=10 ThreadsPerCore=1 State=UNKNOWN
43 NodeName=dnode[09,10,11,12,13,14] RealMemory=16000 Sockets=2 CoresPerSocket=6 ThreadsPerCore=1 State=UNKNOWN
44 NodeName=cnode[1,2,3,4,5,6,7] RealMemory=64000 Sockets=2 CoresPerSocket=8 ThreadsPerCore=1 State=UNKNOWN
45 PartitionName=all Nodes=cnode[1,2,3,4,5,6,7],dnode[01,02,03,04,05,06,07,08,09,10,11,12,13,14] MaxTime=INFINITE
   ↪ State=UP
46 PartitionName=uni Nodes=cnode[1,2,3,4,5,6,7] MaxTime=INFINITE State=UP

```

## Файл конфигурации slurmdbd

```
1 AuthType=auth/munge
2 AuthInfo=/var/run/munge/munge.socket.2
3 StorageType=accounting_storage/mysql
4 DbdAddr=localhost
5 DbdPort=7031
6 DbdHost=localhost
7 MessageTimeout=15
8 DebugLevel=verbose
9 DefaultQOS=normal,standby
10 LogFile=/tmp/slurmdbd.log
11 PidFile=/var/run/slurmdbd.pid
12 StorageHost=localhost
13 StoragePort=3306
14 StoragePass=x4qGD*KNHDb9-Bu>u8N{
15 StorageUser=slurm
16 StorageLoc=slurm_acct_db
17 SlurmUser=slurm
```

## Скрипт синхронизации с ldap

```

1  #!/usr/bin/env python3
2
3  from ldap3 import Server, Connection, ALL
4  from subprocess import run
5  import sys
6
7
8  def main():
9      server = Server(
10         'network1.int.accelcomp.org',
11         get_info=ALL)
12
13     conn = Connection(
14         server,
15         auto_bind=True)
16
17     conn.search(
18         search_base='ou=groups,dc=int,dc=accelcomp,dc=org',
19         search_filter='(|(gidNumber=5001)(gidNumber=5002)(gidNumber=5003))',
20         paged_size=100,
21         attributes=['*'])
22 )
23
24 groups_handler = conn.entries
25
26 conn.search(
27     search_base='ou=people,dc=int,dc=accelcomp,dc=org',
28     search_filter='(objectClass=inetOrgPerson)',
29     paged_size=100,
30     attributes=['*'])
31 )
32
33 regular_users = conn.entries
34
35 if len(sys.argv) == 2:
36
37     # initialize cluster with a given name
38     # if it did not exist before
39
40     run("sacctmgr -i add cluster {}".format(sys.argv[1]), shell=True)
41     run("sacctmgr -i add account regular", shell=True)
42     run("sacctmgr -i add account power", shell=True)
43     run("sacctmgr -i add qos short_term MaxWall=10:00", shell=True)
44
45     # just cleanup everything on each call
46
47     run("sacctmgr -i remove User where DefaultAccount=regular", shell=True)
48
49     # add accounts and give permissions (parse groups)
50
51     for uni_member in groups_handler[1].memberUids:
52         run("sacctmgr -i add user {} DefaultAccount=regular Partition=uni".format(uni_member), shell=True)
53         run("sacctmgr -i modify user where name={} account=regular set qos+=short_term defaultqos=short_term"
54             .format(uni_member), shell=True)
55
56     for acc_member in groups_handler[2].memberUids:
57         run("sacctmgr -i add user {} DefaultAccount=regular".format(acc_member), shell=True)
58         run("sacctmgr -i modify user where name={} account=regular set qos+=short_term defaultqos=short_term"

```

```

59         .format(acc_member), shell=True)
60
61     # parse all ldap users
62
63     for user in regular_users:
64         run("sacctmgr -i add user {} DefaultAccount=regular Partition=debug".format(user.uid), shell=True)
65         run("sacctmgr -i modify user where name={} account=regular set qos+=short_term defaultqos=short_term"
66             .format(user.uid), shell=True)
67
68     for power_member in groups_handler[0].memberUid:
69         run("sacctmgr -i add user {} DefaultAccount=power".format(power_member), shell=True)
70         run("sacctmgr -i modify user where name={} account=power set defaultqos=normal"
71             .format(power_member), shell=True)
72
73
74 if __name__ == "__main__":
75     main()

```

## Файл конфигурации HPL.dat

```

1 HPLinpack benchmark input file
2 Innovative Computing Laboratory, University of Tennessee
3 HPL.out      output file name (if any)
4 6            device out (6=stdout,7=stderr,file)
5 1            # of problems sizes (N)
6 115584      Ns
7 1            # of NBs
8 192          NBs
9 0            PMAP process mapping (0=Row-,1=Column-major)
10 1           # of process grids (P x Q)
11 4           Ps
12 4           Qs
13 16.0        threshold
14 1           # of panel fact
15 2           PFACTs (0=left, 1=Crout, 2=Right)
16 1           # of recursive stopping criterium
17 4           NBMINs (>= 1)
18 1           # of panels in recursion
19 2           NDIVs
20 1           # of recursive panel fact.
21 1           RFACTs (0=left, 1=Crout, 2=Right)
22 1           # of broadcast
23 1           BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
24 1           # of lookahead depth
25 1           DEPTHs (>=0)
26 2           SWAP (0=bin-exch,1=long,2=mix)
27 64          swapping threshold
28 0           L1 in (0=transposed,1=no-transposed) form
29 0           U  in (0=transposed,1=no-transposed) form
30 1           Equilibration (0=no,1=yes)
31 8           memory alignment in double (> 0)
32 ##### This line (no. 32) is ignored (it serves as a separator). #####
33 0           Number of additional problem sizes for PTRANS
34 1200 10000 30000      values of N
35 0           number of additional blocking sizes for PTRANS
36 40 9 8 13 13 20 16 32 64      values of NB

```

## MPI реализация рекурсивной формулы

```
// явно задаем переменные для вычисления
diagx = -2.0 + hx*hx/(2*k0*dt);
diagy = -2.0 + hy*hy/(2*k0*dt);
weightx = k0 * dt/(hx*hx);
weighty = k0 * dt/(hy*hy);

// Обновляем значение подобластей в сетке
for (i=xs[me]; i<=xe[me]; i++)
for (j=ys[me]; j<=ye[me]; j++)
x[i][j] = weightx*(x0[i-1][j] + x0[i+1][j] + x0[i][j]*diagx)
+ weighty*(x0[i][j-1] + x0[i][j+1] + x0[i][j]*diagy)
```

**Функция межпроцессорного взаимодействия**

```
while (!converge) {  
    // увеличиваем шаг и время  
    t = t + dt;  
    step = step + 1;  
    // Прогоняем FTCS схему ровно один раз  
    computeNext(x0, x, dt, hx, hy, &localDiff, me, xsi, ysi, xei, yei, k0);  
    updateBound(x0, neighBor, comm2d, column_type, ime, xsi, ysi, xei, yei, yce);  
    MPI_Allreduce(&localDiff, &result, 1, MPI_DOUBLE, MPI_SUM, comm);  
    result = sqrt(result);  
    // Выходим из цикла если считать уже нечего  
    if ((result < epsilon) || (step > maxStep)) break;  
}
```