A computational model for interactive visualization of high-performance computations

ВЫЧИСЛИТЕЛЬНАЯ МОДЕЛЬ ДЛЯ ИНТЕРАКТИВНОЙ ВИЗУАЛИЗАЦИИ ВЫСОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Павел Васёв

Институт математики и механики имени Н.Н. Красовского УрО РАН, г. Екатеринбург

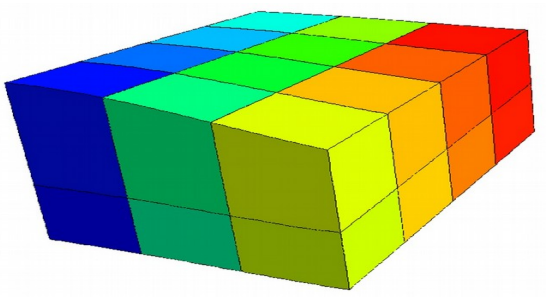
Интерактивная визуализация суперкомпьютерных вычислений

Термины:

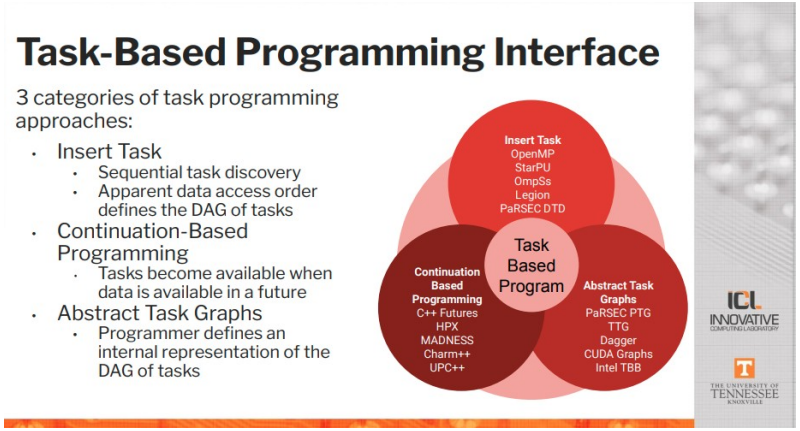
- Онлайн-визуализация, insitu-визуализация

Современное состояние:

- Для реализации самой визуализации нужен суперкомпьютер



Вычислительная модель



Мотивация и критерии:

- Задача визуализации существенно интерактивна, необходима “гибкая” система параллельных вычислений.
- Необходим способ стыковки этой модели с существующими счетными кодами, чтобы “получать” из них данные и передавать управляющие воздействия.
- **Идея**: используем метод графа вычислений.

Обещание

- **Обещание (promise, future) — примитив синхронизации**.
- Находится в одном из состояний – ожидает, выполнено, и д.р.
- Можно узнать, выполнено ли обещание.
- Можно заказать вызов callback, когда обещание будет выполнено.
- Можно выполнить обещание и задать связанное с ним значение.

- Обещание находится в одном из состояний. Обычно делают такие:
  Pending | Resolved | Rejected | Cancelled
- Из pending может перейти в любое другое, и больше не переходит.
- Можно выяснить состояние и связанное значение, заказать callback на смену состояния.
- Можно перевести обещание в то или иное состояние и задать связанное значение.

- **Добавить_данные**: данные → promise

Добавляет данные.
Возвращает объект обещания, значение которого связано с добавленными данными.

Объект обещания маленький, и его можно передавать по сети между разными программами.

- Добавить_данные: данные → promise
- **Выполнить**: действие, аргументы → promise

Добавляет задачу, то есть заявку на выполнение действия.

Возвращает объект обещания, которое будет выполнено по завершению выполнения действия. Значение обещания – это будет результат выполнения действия.

Действие это функция (чистая, грязная). Аргументы передаются действию.

В значениях аргументов могут быть обещания.
В этом случае действие будет доступно к выполнению только когда все обещания перечисленные в аргументах — выполнены.

- Добавить данные: данные → promise
- Выполнить: действие, аргументы → promise
- **Получить данные**: promise → данные

Ждёт выполнение обещания, и по выполнению загружает данные, связанные с этим обещанием.

- Добавить-данные: данные → promise
- Выполнить: действие, аргументы → promise
- Получить-данные: promise → данные

Дополнительные операции

- **Когда все**: список promise → promise
- **Когда одно**: список promise → promise

Когда все обещания из списка выполнены — итоговое обещание выполняется. Связанное значение — массив значений обещаний.

Когда любое одно обещание из списка выполнено — итоговое обещание выполняется. Связанное значение — значение выполненного обещания.

Полный список операций

- Добавить-данные: данные → promise
- Выполнить: действие, аргументы → promise
- Получить-данные: promise → данные
- Когда все: список promise → promise
- Когда одно: список promise → promise

Утверждается, что этих операций достаточно для описания разнообразных вычислений, в том числе и визуализации, и математических моделирующих.

Использование модели

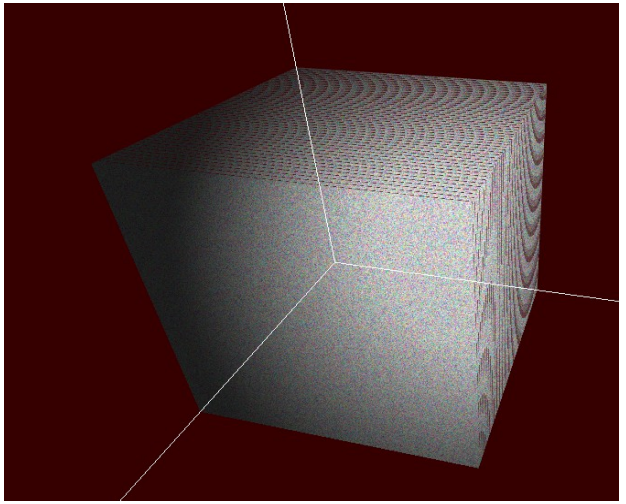
- Создаётся обычный **последовательный алгоритм**, который формирует схему параллельных вычислений (граф вычислений).
- Система выполняет добавленные задачи на узлах вычислителя в некотором порядке, ищет оптимальные распределения задач.

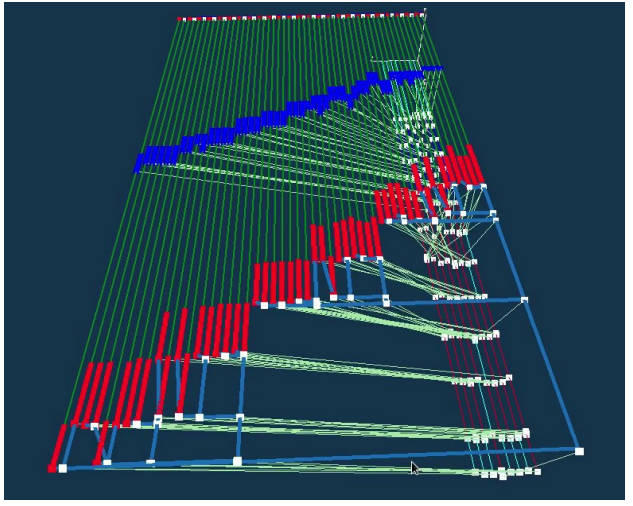
- **Алгоритм** может быть реагирующий, взаимодействующий с пользователем.
- Алгоритм может пользоваться всей мощью структурного программирования (например абстрагирование SICP).

```
let K = 50
let filenames = ["1.dat","2.dat",...,K+".dat"]
let blocks = filenames.map( __load )
```

```
rapi.query( "render",(m) => {
 let images = blocks.map( b =>
                __render( b, m.camera_position, m.w, m.h) )
 let final_image = recursive_merge( images )
 rapi.msg( {label:"image", final_image } )
})
```

```
function __load( filepath ) {
  return rapi.exec( arg =>
      read_file_as_floats(arg.filepath),{filepath} )
}
function __render( block, camera_position, w, h ) {
  return rapi.exec( arg =>  arg.render_fn(arg.camera_position),
      {render_fn: {code: "cell_render_func", need: true,
                arg: {block,w,h}}})
}
function recursive_merge( images ) {
  if (images.length <= 1) return images[0]
  let acc = []; for (let i=0; i<images.length; i+=2 )
     acc.push( __merge_2( images[i], images[i+1] ) )
  return recursive_merge( acc )
}
function __merge_2( image1, image2 ) {
  return rapi.exec( arg => merge_2_zbuf( arg.image1, arg.image2 ), { image1, image2 })
}
```





https://youtu.be/XnV3l8hw8QE

Тонкости модели и реализации

- Структура системы и распределение ролей
- Учёт состояния исполнителей
- “Обезжиревание” данных, выгрузка данных
- Передача владения данными
- Счётчик использования обещаний
- Распределение отслеживания готовности задач
- Основа реализации: сообщения и реакции.

Структура системы

Необходимо:

- Вести список обещаний и их состояний, рассылать интересующимся уведомления по мере готовности.
- Хранить данные связанные с обещаниями, и особо – массивы данных.
- Вести список задач и их зависимостей от обещаний, вычислять готовность задач.
- Выполнять задачи (а точнее действия).

Структура системы

- **Сервис обещаний** – ведёт список обещаний и хранит их данные.
- **Сервис задач** – принимает заявки на задачи.
- **Исполнители** – решают задачи и их готовность, хранят массивы данных.
- Дополнительный **сервис хранения** массивов данных.
- В основе: сервис обмена реакциями.

Учёт состояния исполнителей

При выполнении действий выделяются под-действия, выражающиеся в подготовке состояния исполнителя.

- Загрузить библиотеку..
- Загрузить массив данных..
- Сформировать структуру данных...

Проблема: эти под-действия затратны по времени, если выполнять их для каждого действия, то это существенное дублирование вычислений.

- **Идея**: учитывать наличие результатов действий при назначении задач, и использовать их.

Пример

Действие по рендерингу массива данных.

Вычислить:

- mu1 – массив данных
- camera – положение взгляда.
- renderlib – среда рендеринга.

- Здесь имеет смысл кешировать **mu1** и **renderlib** в памяти исполнител И учитывать при назначении задач.

Обезжиревание данных

Проблема: при операции добавить-данные (а также при сохранении результатов действий) на самом деле нереально куда либо их добавить.

- Вводятся операции

выгрузить-массив: массив → url

получить-массив: url → массив

- Объект данных становится пучком небольших записей, среди которых – межузловые указатели на массивы данных находящихся в памяти устройств (ram, gpu, …)
- Такой объект можно легко передавать по сети.

Автоматизация обезжиревания

- Если аргумент операции “**добавить данные**” или **результат действий** является словарём и содержит поле **payload**, то оно интерпретируется как список массивов данных.
- Эти массивы выгружаются. Поле payload заменяется на ссылки на массивы. Таким образом объект data остаётся **небольшим**.
- При запуске **действий** проводится обратное преобразование, и алгоритм действия получает исходное data.
- При этом есть возможность отказаться от этого преобразования и загружать массивы вручную.

Реализация обезжиревания

- Операция **выгрузить-массив** оставляет данные там где они есть, но запоминает адрес и описание массива.
- Возвращает URL доступа, по которым эти данные можно получить, как в этом же процессе так и во внешних процессах.
- Для обеспечения внешнего доступа к URL запускается сетевой сервер с расчётом на скоростные сетевые протоколы доступа.

- При недостатке памяти исполнителя данные выгружаются во “внешнее” хранилище (компонент которого как правило находится на этом же узле).

Передача владения данными

Проблема: аллокация памяти для массивов данных каждым действием это дорогая медленная операция!

- Модификатор “**Передать владение**”: promise → promise.
- Массивы данных, связанные с входным обещанием, передаются под управление последующему действию. Исходное обещание их “теряет”.
- Таким образом, снижается количество аллокаций памяти.
- Это же касается и массивов на устройствах (GPU).

Распределение отслеживания готовности задач

Проблема: если отслеживать готовность задач централизованно, то возникает задержка от момента готовности до фактического старта выполнения действия.

- **Причины**: время назначения задачи, время передачи сигнала с требованием старта действия.
- **Решение**: задачи отслеживаются и назначаются в процессах исполнителей. Следовательно – надо заранее решать на каком исполнителе будет выполняться действие. Но это можно делать в окне очереди задач..

Счётчик использования обещания

Проблема: память системы заполняется!

- Значение счётчика задаётся при каждой операции создающей обещание (добавить данные, выполнить). По умолчанию – единица (или лучше бесконечность?).
- Значении счётчика снижается при использовании обещания, т.е. в операциях “выполнить” и “получить данные”.
- По достижению нуля, обещание и связанные с ним массивы данных стираются.

Сообщения и реакции

- Всё вышеперечисленное реализовано на основе модели сообщений и реакций. Которая реализована следующим образом.

- Сообщения размещаются в топиках (topic-driven).
- При размещении выполняются все реакции, привязанные к топику (в процессе отправителя).
- Это позволяет реализовать, в частности, передачу сообщений клиентам во внешних процессах.

A computational model for interactive visualization of high-performance computations

ВЫЧИСЛИТЕЛЬНАЯ МОДЕЛЬ ДЛЯ ИНТЕРАКТИВНОЙ ВИЗУАЛИЗАЦИИ ВЫСОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Павел Васёв

Институт математики и механики имени Н.Н. Красовского УрО РАН, г. Екатеринбург

https://youtu.be/XnV3l8hw8QE