Введение в технологию фрагментированного программирования и знакомство с системой LuNA

В.А. Перепелкин

Летняя школа по параллельному программированию, 2016 г.

Использование компьютеров

Компьютеры — хороший инструмент

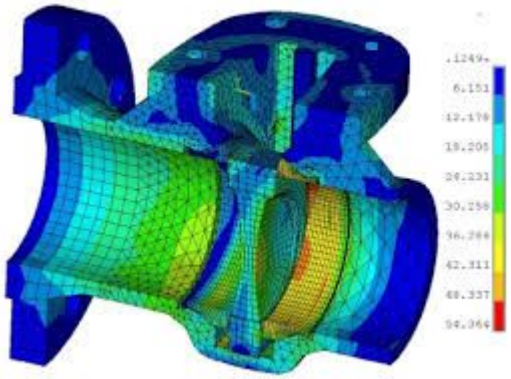
Сила компьютеров — в скорости работы и запоминающей способности

Реализовать потенциал использования компьютеров сложно

Это является научной проблемой

HPC (high performance computing) — это область, требующая проработки

Прикладное и системное программирование



Компиляторы,
операционная система,
IDE, профилировщики,
отладчики,
интерпретаторы,
верификаторы,
генераторы, фреймворки,
...

Система фрагментированного программирования LuNA

LuNA — Language for Numerical Algorithms

Система LuNA

- Разрабатывается в ИВМиМГ СО РАН
- Предназначена для поддержки параллельной реализации больших численных моделей для суперкомпьютеров
- LuNA-программа описывается на высоком уровне абстракции
- Система LuNA исполняет программу, автоматизированно настраивая её на доступные ресурсы и обеспечивая динамические свойства её

Вопросы

- Зачем нужно изучать другой язык программирования?
- Что представляют собой технология фрагментированного программирования и система LuNA?
- Почему они устроены так, как устроены?

Проблема

— На решение какой проблемы направлена технология фрагментированного программирования?

Параллельная реализация крупномасштабных численных моделей для суперкомпьютеров — сложная задача системного параллельного программирования

Крупномасштабные численные модели

Численные модели научного моделирования, характеризующиеся:

- Большим объемом данных (терабайты и более)
- Большим объемом вычислений (терафлопсы и более)

Программирование на суперкомпьютерах

- Параллельное, т.к. все суперкомпьютеры — параллельные
- Системно сложное, т.к. требуется:
  - представить алгоритм решения задачи в параллельной форме
  - обеспечить эффективную работу суперкомпьютера (равномерная и полная загрузка полезными вычислениями)

Параллельное программирование и параллельное программирование на суперкомпьютерах

Программирование для суперкомпьютеров (пета-, эксафлопсных и далее) имеет важные особенности:

- Асинхронность вычислений
- Гетерогенность вычислителя
- Локальность взаимодействий
- Децентрализованность вычислений
- Ограниченность знаний о состоянии программы
- Ограничения в декомпозиции данных
- Необходимость поддержки динамических свойств

Проблема

— На решение какой проблемы направлена технология фрагментированного программирования?

Параллельная реализация крупномасштабных численных моделей для суперкомпьютеров — сложная задача системного параллельного программирования

Цель технологии фрагментированного программирования (ТФП)

— В чем цель технологии фрагментированного программирования?

Автоматизация эффективной параллельной реализации численного алгоритма на суперкомпьютере

Автоматизация программирования

Формулировка задачи

Алгоритм решения

Параллельная программа

Формулировка задачи

- Функциональные требования, т.е. функция, которая должна быть вычислена
  - Пример: сортировка массива
- Нефункциональные требования (эффективность)

Алгоритм решения задачи

- Алгоритм задаёт способ вычисления функции.
- Для вычисления одной и той же функции существует счётное множество алгоритмов
- Разные алгоритмы обладают разными нефункциональными свойствами

Программа, реализующая алгоритм

Один и тот же алгоритм можно запрограммировать разными программами

Программа:

- реализует алгоритм
- управляет ресурсами
- содержит управление
- менее переносима

Переносимость

- Переносимость на уровне исполняемых файлов
- Переносимость на уровне исходных кодов
- Переносимость с точки зрения сохранения нефункциональных свойств (эффективности)

Эффективность

- Экономия памяти
- Экономия времени вычислений
- Малая доля накладных расходов

Частичная автоматизация

Эффективная реализация численного алгоритма — алгоритмически труднорешаемая задача

Аналог: задача о рюкзаке, задача комивояжера

Вывод: автоматизация должна быть частичной, система — специализированной

Представление алгоритма

Разные представления алгоритма обладают разными свойствами:

- явность параллелизма
- гранулярность алгоритма
- автономность частей алгоритма
- управление ресурсами

Для разных целей удобны разные представления

Масштабируемость исполнения алгоритма

Виды масштабируемости: сильная и слабая

Требования масштабируемости:

- Децентрализация по данным, вычислениям и коммуникациям
- Локальность коммуникаций

Цель технологии фрагментированного программирования (ТФП)

— В чем цель технологии фрагментированного программирования?

Автоматизация эффективной параллельной реализации численного алгоритма на суперкомпьютере

Фрагментированный алгоритм

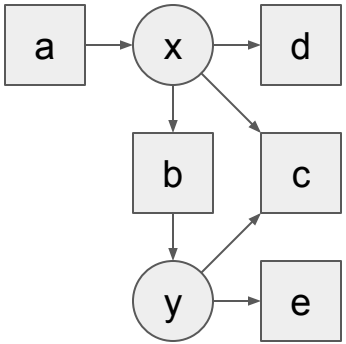
— Каково представление алгоритма в ТФП и каковы его особенности?

Алгоритм представляется в явно-параллельной форме, ориентированной на автоматизацию обеспечения нефункциональных свойств

Определение фрагментированного алгоритма (ФА)

```
df x, y;

cf a: func1(out: x)
cf b: func2(in: x, out: y)
cf c: func3(in: x, y)
cf d: func4(in: x)
cf e: func4(in: y)
```



ФА — это конечное множество фрагментов данных и вычислений (ФД и ФВ), связанных отношениями in и out

Исполнительная семантика — data flow, то есть, по готовности данных. Выполнение ФВ означает, что значение выходных ФД будет вычислено

ФД может быть вычислен или не вычислен

ФД — единтственного присваивания

Порядок выполнения может быть разным, в т.ч. возможно параллельное выполнение

Особенности ФА

Порядок выполнения ФА влияет на расход памяти и доступный параллелизм в разные моменты времени

Допустим любой порядок выполнения ФВ, не противоречащий информационным зависимостям

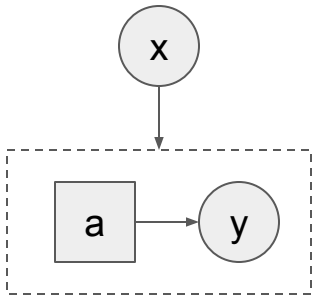
ФД возможно передавать по сети, обеспечивая подачу входных аргументов для ФВ

ФВ возможно передавать по сети, обеспечивая динамическую балансировку нагрузки на вычислительные узлы (*)

Эффективная реализация ФА является труднорешаемой задачей

Определение ФА (продолжение)

```
if (x>0) {
     df y;
     cf a: func1(out: y)
     ...
}
```



if — это *структурированный ФВ*

Входной параметр: выражение

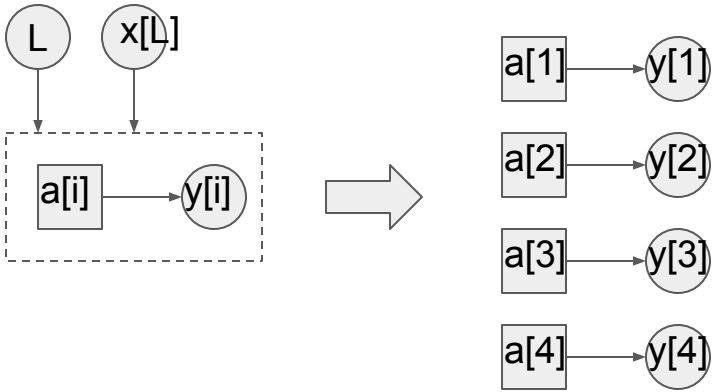
Входные ФД: необходимые для вычисления выражения

Если выражение истинно, то ФВ перобразуется в можество фрагментов, описанных в теле оператора

Множество ФД и ФВ фрагментированного алгоритма зависит от значений ФД

Определение ФА (продолжение)

```
for i=1..x[L] {
     cf a[i]: func1(in: i; out: y[i]);
}
```



for — это *структурированный ФВ*

Входные параметры: выражения для первого и последнего значений параметра цикла — счётчика

При исполнении ФВ раскрывается на несколько копий тел, в каждом из которых значение счётчика имеет разное значение

Счётчик не является ФД, он — константное выражение

Особенности раскрутки оператора FOR

Оператор FOR можно “раскручивать” постепенно в целях экономии памяти

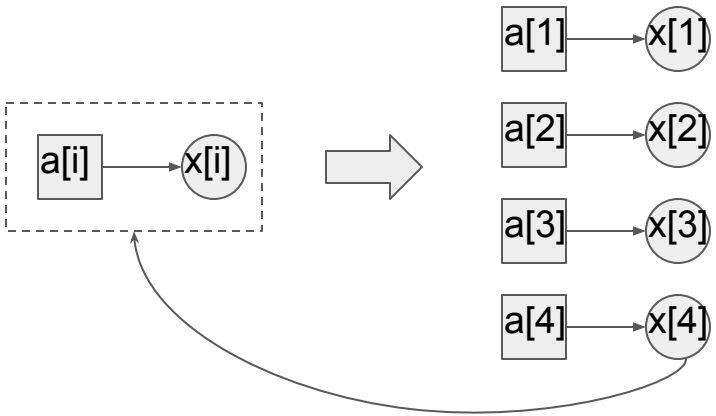
В этом случае система должна определить с какого конца следует раскручивать FOR

В общем случае эффективная раскрутка FOR является сложной задачей

На практике для численных алгоритмов существуют частные эффективные алгоритмы раскрутки FOR

Определение ФА (продолжение)

```
while x[i-1]>0, i=1..out N {
     cf a[i]: func1(in: i; out: x[i]);
}
```



while — это *структурированный ФВ*

Входные параметры:
1. выражение для первого значения параметра цикла — счётчика
2. значение выражения для текущего значения счётчика

При исполнении ФВ раскрывается на одну копию тела с заданным значением счётчика и аналогичный оператор WHILE, где начальное значение счётчика на единицу больше предыдущего

Определение ФА (продолжение)

```
sub A(in: x; out: y)
{
     df z;
     cf a: func1(in: x; out: z);
     cf b: func1(in: z; out: y);
}

...

cf a: A(in: x[0], out: x[1]);
```

Фрагментированные подпрограммы — это структурированные ФВ.

Входные и выходные значения определяются аналогично атомарным ФВ

Весь ФА — это тоже подпрограмма (аналог main в C/C++)

Фрагменты данных

Описание фрагментов данных, фактически, является избыточным и определяется их употреблением

Декларирование ФД служит целям проверки на ошибки

Есть реализационные особенности, связанные с определением ФД, но они не принципиальные и выходят за рамки материала лекции

Фрагментированный алгоритм

Понятие ФА определено, в модели больше нет ничего другого

Исполнение ФА на мультикомпьютере

ФВ и ФД — реализуются объектами, расположенными на различных узлах (способ реализации рассматривается позже)

Исходно один ФВ (“main”) располагается на одном из узлов

В дальнейшем происходит исполнение/раскрытие ФВ по готовности входных ФД

Система обеспечивает миграцию ФД и ФВ по узлам для выполнения всех ФВ

Требования к реализации ФА

Необходимо: обеспечить доставку всех входных ФД ко всем ФВ. Эта задача не может быть невыполнена

Важно (но не необходимо): обеспечить нефункциональные свойства. Эта задача может быть выполнена лучше или хуже. В частности, система старается:

— Уменьшить время выполнения программы
— Увеличить равномерность нагрузки на вычислительные узлы
— Экономить сетевой траффик
— Своевременно поставлять ФД к ФВ, их потребляющим

(*)

Основные компоненты системы LuNA

Компилятор LuNA:

- Синтаксис языка LuNA
- Транслятор LuNA —> внутреннее JSON-представление ФА
  - Тривиальное преобразование из одного синтаксиса в другой
- Оптимизации и проверки на ошибки (JSON—>JSON)
- API для определения пользовательских фрагментов кода (компилируется традиционным компилятором)

Исполнительная система LuNA (далее)

Дополнительные компоненты (профилировщик, визуализатор, ...)

Организация исполнительной системы LuNA

Некоторые вещи рассматриваются упрощённо, но не в ущерб принципиальному пониманию организации системы

Рассмотрим основные компоненты исполнительной системы (ИС) LuNA

Коммуникационный менеджер

Модуль, предназначенный для осуществления коммуникаций между вычислительными узлами

Использует MPI для фактической передачи сообщений (можно сказать, что он является “обёрткой” над MPI). Коммуникационная библиотека может быть заменена на другую без изменения остальной системы

Функционирует в отдельном потоке, фактически — непрерывно ждёт сообщений с помощью MPI_Probe

Отправка сообщений осуществляется асинхронно

Менеджер фрагментов данных (МФД)

Фрагмент данных — это пара (указатель, размер)

МФД в пределах узла — это коллекция таких пар с интерфейсом, позволяющим добавлять или удалять ФД

Передача ФД по сети осуществляется как передача обычных данных. В отправляемый пакет упаковывается идентификатор ФД и его содержание (блок памяти)

Для отправки ФД по сети используется коммуникационный менеджер

Доставка ФД

Одна из основных функций МФД — это доставка ФД по требованию

1. Некоторый узел посылает запрос на ФД
2. Запрос пересылается на нужный узел по маршруту (модуль PathFinder)
    a. Запрос — это сообщение, содержащее имя запрашиваемого ФД и номер запросившего узла
3. На нужном узле запрос ожидает ФД (если тот ещё не поступил)
4. Затем копия ФД отправляется на запросивший узел

Реализация ФА

Исполнительной системе в разных ситуациях нужна информация о ФА (например, для определения входных ФД для ФВ)

Существует сервисный класс, который считывает JSON-файл и затем может давать информацию о нём (какие есть подпрограммы, какие у них параметры, что находится в теле операторов и т.д.)

Менеджер ФВ (МФВ)

ФВ реализуется как идентификатор соответствующего оператора в описании ФА

По этому идентификатору можно определить какие входы и выходы есть у ФВ, какой именно это оператор (exec, if, for, ...)

Наличие ФВ на узле означает, что этот ФВ необходимо исполнить (так отслеживается фронт выполнения ФВ)

В начальный момент на одном узле имеется единственный ФВ — main.

МФВ: Первичная миграция

При создании ФВ он мигрирует на узел, на котором он будет исполняться (т. к. порождение и исполнение ФВ, вообще говоря, могут происходить на различных узлах)

Миграция ФВ, фактически, сводится к передачи по сети его идентификатора. Когда коммуникационный менеджер получает сообщение с пометкой “ФВ”, он передаёт это сообщение на обработку МФВ.

Первичная миграция может проходить по нескольким узлам прежде, чем достигнет своего назначения (определяется через PathFinder)

МФВ: Ожидание входных ФД

Когда ФВ достиг узла исполнения, он отправляет запросы на входные ФД. Иногда это проходит в несколько этапов (когда в индексных выражениях встречаются ФД)

МФД осуществляет доставку запрошенных ФД на узел. Полученные ФД прикрепляются к ФВ.

Процесс продолжается до тех пор, пока все входные ФД для ФВ не окажутся к нему прикрепленными

МФВ: Исполнение в пуле потоков

Пул потоков (многопоточный портфель задач) содержит готовые к исполнению ФВ (с полным комплектом входных ФД).

Пул потоков — это стандартный шаблон многопоточного программирования

ФВ исполняется рабочими потоками по мере их освобождения. Информация о входах и выходах ФВ имеется в описании ФА, на основании которых формируется:
— вызов к библиотеке пользовательских кодов (*.so файл) с нужными параметрами (для атомарных ФВ)
— порождение новых ФВ, определённых в теле оператора (для структурированных ФВ)

По завершению вызова выходные ФД передаются в МФД для хранения

МФВ: Поиск объектов (Path finder)

Объекты (ФД и ФВ) должны быть распределены по вычислительным узлам. Это распределение может быть задано разными способами, в том числе корректироваться или определяться динамически. Так или иначе, должна быть обеспечена возможность нужный объект найти

Модуль Path finder по заданному имени объекта выдаёт номер соседнего узла, где следует продолжить поиск, либо сообщает о том, что объект должеен находиться на текущем узле.

Модуль Path finder должен гарантировать, что местоположение любого объекта будет найдено таким способом

МФВ: упрощённый пример Path finder

Пусть задано статическое распределение ФВ и ФД по узлам в виде функции, которая по имени объекта возвращает номер его узла

Тогда Path finder по имени объекта будет определять номер узла назначения и выдавать номер следующего узла на пути к искомому в соответствии с сетевой топологией

Динамическая балансировка нагрузки

При обнаружении дисбаланса вычислительной нагрузки ФВ при первичной миграции передаются соседним недогруженным вычислительным узлам

В более сложных случаях недогруженные узлы определяются иначе, но в остальном схема такая же

Другой вариант динамической балансировки нагрузки — это job stealing, часто применяющийся для распределённых портфелей задач (не реализован в текущей версии системы LuNA)

Сборка мусора

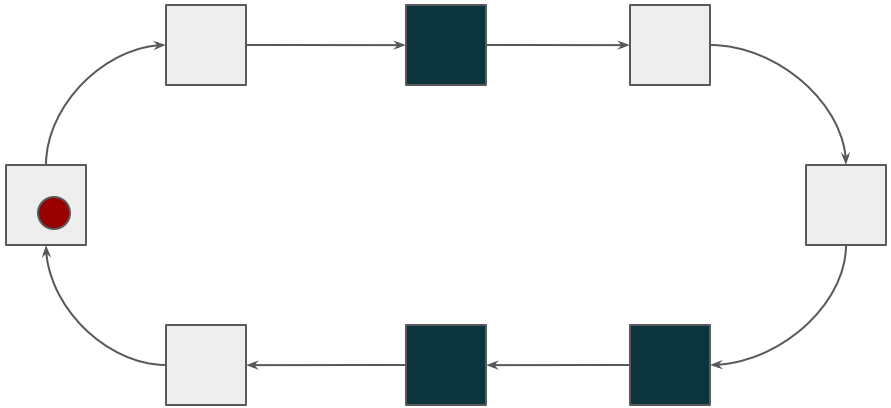
Сборка мусора осуществляется по завершению области видимости, в которой объявлен ФД. МФД отслеживает ФД, порождённые внутри области видимости (тела оператора — {...}). МФВ отслеживает ФВ, порождённые и исполненные в области видимости. Когда все ФВ исполнены, то удаляются все ФД в данной области видимости.

Такой алгоритм сборки мусора является малоэффективным, поэтому в системе LuNA существуют другие способы обеспечивать сборку мусора.

Остановка системы

Обнаружение ситуации, когда ни на одном узле уже нет работы, является сложной задачей.

В системе LuNA используется распределённый алгоритм с малыми накладными расходами и локальными взаимодействиями.



Исполнительная система LuNA — итоги

Исполнительная система представляет собой распределённую среду, в которой существуют ФД и ФВ. Система обеспечивает выполнение ФВ в соответствии с семантикой ФА. Рассмотрены все основные составляющие этих процессов:

— порождение, хранение, миграция и исполнение ФВ

— порождение хранение, миграция и удаление ФД

Перспективы проекта

— Совершенствование языка

— Создание интеллектуальных системных алгоритмов

— Решение прикладных задач

— Доведение системы до пользователей

Спасибо за внимание!