Особенности использования системы LuNA на примере реализации PIC-метода

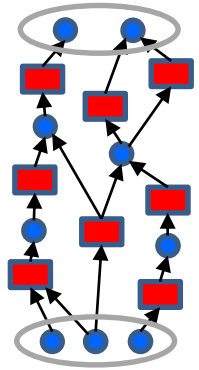
Киреев С.Е.
ИВМиМГ СО РАН

Введение

- Проблема: создание эффективных параллельных программ для решения больших численных задач
- Технология фрагментированного программирования (ТФП) разрабатывается с целью автоматизации решения этой проблемы
- На базе ТФП разрабатывается система фрагментированного программирования LuNA
- **Цель работы:** апробация системы LuNA на конкретной прикладной задаче:
  - Реализовать метод частиц-в-ячейках в системе LuNA
  - На основе полученного опыта отметить сильные и слабые стороны системы LuNA, предложить пути для её улучшения

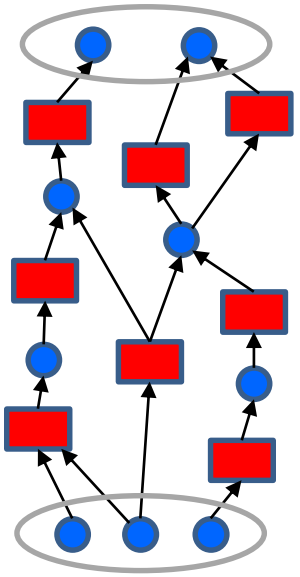
Технология фрагментированного программирования и система LuNA

- Особенности ТФП
  - Высокоуровневое представление алгоритма
    - В основе – граф информационных зависимостей
    - Явно выраженный параллелизм
    - Независимость от ресурсов
  - Исполнительная система
    - Нацелена обеспечивать эффективную реализацию алгоритма на доступных вычислительных ресурсах
    - Должна решать задачи распределения ресурсов, балансировки нагрузки, …



Представление алгоритма в ТФП

- Фрагментированный алгоритм (ФА)
  - Фрагменты данных (ФД)
  - Фрагменты вычислений (ФВ)
  - Отношения вход-выход



- Исполнение ФА
  - Каждый ФВ исполняется по готовности всех входных ФД и в результате исполнения вычисляет выходные ФД
  - Исполнение продолжается, пока все ФВ не окажутся исполненными

Язык LuNA

- Язык LuNA включает:
  - Средства текстовой записи ФА (ФД,ФВ,зависимости)
    - **df** x,y,z;
    - **cf** add: proc_sum(#in x, y, #out z);

Язык LuNA

- Язык LuNA включает:
  - Средства текстовой записи ФА (ФД,ФВ,зависимости)
    - **df** x,y,z;
    - **cf** add: proc_sum(#in x, y, #out z);
  - Средства задания индексированных множеств ФД и ФВ (массивов)
    - **for** i=1..N { **cf** add[i]: proc_sum(#in x[i],y[i], #out z[i]); }
    - **while** (x[i]<eps), i=1..**out** N { **cf** s[i]: step(#in x[i], #out x[i+1]); }

Язык LuNA

- Язык LuNA включает:
  - Средства текстовой записи ФА (ФД,ФВ,зависимости)
    - **df** x,y,z;
    - **cf** add: proc_sum(#in x, y, #out z);
  - Средства задания индексированных множеств ФД и ФВ (массивов)
    - **for** i=1..N { **cf** add[i]: proc_sum(#in x[i],y[i], #out z[i]); }
    - **while** (x[i]<eps), i=1..**out** N { **cf** s[i]: step(#in x[i], #out x[i+1]); }
  - Средства задания условных ФВ
    - **if** (x>10.0) **cf** add: proc_sum(#in x, y, #out z);

Язык LuNA

- Язык LuNA включает:
  - Средства текстовой записи ФА (ФД,ФВ,зависимости)
    - **df** x,y,z;
    - **cf** add: proc_sum(#in x, y, #out z);
  - Средства задания индексированных множеств ФД и ФВ (массивов)
    - **for** i=1..N { **cf** add[i]: proc_sum(#in x[i],y[i], #out z[i]); }
    - **while** (x[i]<eps), i=1..**out** N { **cf** s[i]: step(#in x[i], #out x[i+1]); }
  - Средства задания условных ФВ
    - **if** (x>10.0) **cf** add: proc_sum(#in x, y, #out z);
  - Средства задания структурированных ФВ (подпрограмм)
    - **sub** add_vector(#in **int** n, **name** x, **name** y, #out **name** z) {
    - **for** i=0..n-1 { **cf** add[i]: proc_sum(#in x[i], y[i], #out z[i]); }
    - }

Язык LuNA

- Язык LuNA включает:
  - Средства текстовой записи ФА (ФД,ФВ,зависимости)
    - **df** x,y,z;
    - **cf** add: proc_sum(#in x, y, #out z);
  - Средства задания индексированных множеств ФД и ФВ (массивов)
    - **for** i=1..N { **cf** add[i]: proc_sum(#in x[i],y[i], #out z[i]); }
    - **while** (x[i]<eps), i=1..**out** N { **cf** s[i]: step(#in x[i], #out x[i+1]); }
  - Средства задания условных ФВ
    - **if** (x>10.0) **cf** add: proc_sum(#in x, y, #out z);
  - Средства задания структурированных ФВ (подпрограмм)
    - **sub** add_vector(#in **int** n, **name** x, **name** y, #out **name** z) {
    - **for** i=0..n-1 { **cf** add[i]: proc_sum(#in x[i], y[i], #out z[i]); }
    - }
  - Интерфейс с языком C++
    - **import** c_proc_sum(**int**, **int**, **name**) **as** proc_sum;

Прикладная задача

Моделирование динамики самогравитирующего пылевого облака:
Математическая постановка

Пыль

$$\frac{\partial f}{\partial t} + \mathbf{u}\frac{\partial f}{\partial \mathbf{r}} + \frac{\mathbf{F}_p}{m}\frac{\partial f}{\partial \mathbf{u}} = 0$$

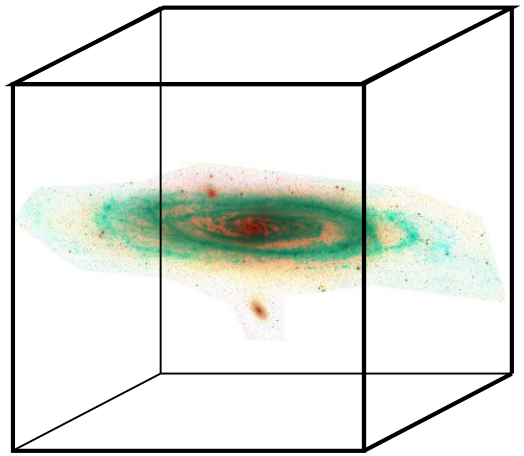
$$\rho(t,\mathbf{r}) = \int f(t,\mathbf{r},\mathbf{u})d\mathbf{u}$$

Гравитация

$$\frac{\partial^2 \Phi_p}{\partial x^2} + \frac{\partial^2 \Phi_p}{\partial y^2} + \frac{\partial^2 \Phi_p}{\partial z^2} = 4\pi\rho$$

$$\Phi = \Phi_p + \Phi_s$$

$$\mathbf{F} = -\rho\, grad\, \Phi$$



Область моделирования

Прикладная задача

Моделирование динамики самогравитирующего пылевого облака:
Методы решения

- Решение уравнения Больцмана (пыль) – PIC-метод
  - Данные
    - Пыль – свободно движущиеся модельные частицы одинаковой массы
    - Поля – 3D сетка
  - Алгоритм PIC-метода
    - Вычисление распределения плотности частиц (частицы → сетка)
    - Вычисление распределения гр. потенциала (сетка → сетка)
    - Вычисление гр. сил, действующих на частицы (сетка → сетка)
    - Сдвиг частиц под действием гр. сил (сетка → частицы)
- Решение уравнения Пуассона (гравитация) – метод Якоби
  - Явный итерационный метод
  - Вход – распределение плотности на 3D сетке
  - Выход – распределение гравитационного потенциала на 3D сетке

Прикладная задача

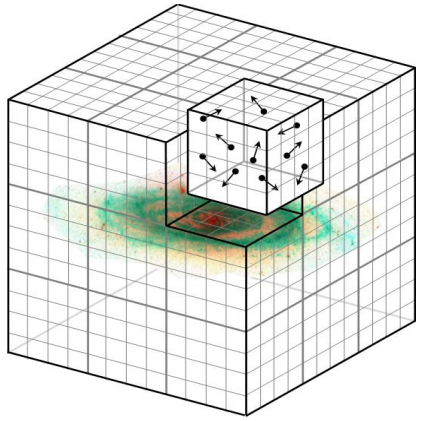
Моделирование динамики самогравитирующего пылевого облака:
Параметры задачи

- Размер сетки: NX × NY × NZ
- Количество частиц: NP
- Количество шагов по времени: NT
- …

Прикладная задача

Моделирование динамики самогравитирующего пылевого облака: **Фрагментация**

- Декомпозиция пространства моделирования на подобласти одинакового размера (domain decomposition)



- Параметры фрагментации
  - Количество подобластей: FX × FY × FZ
- Каждая подобласть содержит:
  - Фрагменты 3D сеток с перекрытием границ
  - Подмножество частиц, находящихся в данной подобласти
- В алгоритм добавляются этапы обмена:
  - Обмен граничными значениями фрагментов 3D сеток
  - Передача перелетевших частиц

Реализация задачи в языке LuNA

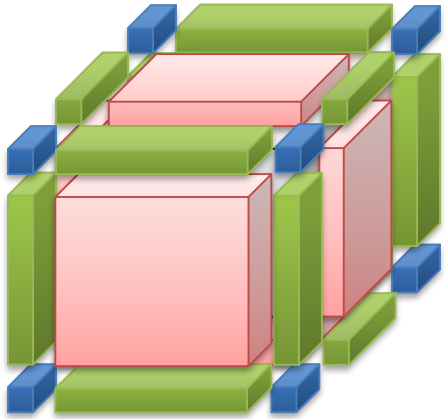
- Основная LuNA-программа повторяет структуру алгоритма PIC-метода
- Вспомогательные модули:
  - Модуль **lib_basic**: простые базовые операции
  - Модуль **lib_mesh3d**: операции с фрагментированным 3D массивом
    - Различные редукции
    - Обмены граничными значениями с различным соседством
    - Вывод в файл
  - Модуль **lib_poisson3d**: решатель уравнения Пуассона на фрагментированной 3D сетке методом Якоби
  - Модуль **lib_mesh3d_exchange_particles**: обмены частицами между фрагментами фрагментированного множества частиц

Реализация задачи в языке LuNA: трудности

Обмен «границами» 3D массива фрагментов

- Варианты:
  - Различная структура массива и положение фрагмента в структуре

  - Различные типы соседства фрагментов
    - 6 соседей, 19 соседей, 27 соседей
  - Различные операции
    - Копирование граничных значений
    - Суммирование граничных значений
    - Передача частиц
  - Различная ширина границ



Реализация задачи в языке LuNA: трудности

Обмен «границами» 3D массива фрагментов

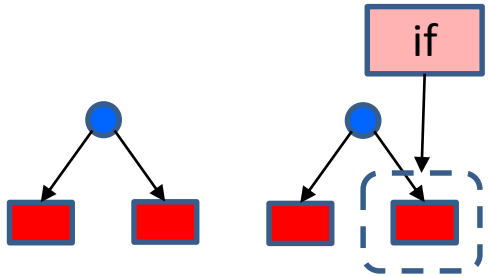
- Реализация:
  - Параметризованный генератор LuNA-программы на Python-е
  - Генерирует LuNA-программу всевозможных обменов (произвольная структура 3D массива и положение фрагмента в массиве) для заданной операции обменов и заданного типа соседства
  - Размер получаемых файлов LuNA-программы:
    - ~2060 строк – fa-программа
    - ~2000-9000 строк – cpp-программа

Реализация задачи в языке LuNA: трудности

Удаление фрагментов данных (сборка мусора)

- Реализация сборки мусора в системе LuNA
  - По завершению области видимости (не подходит для массивов)
  - По явному указанию: «удалить эти ФД после завершения этого ФВ»

- Проблема:
  - как обнаружить, что ФД больше не нужен?



- Вынужденное решение:
  - Логические переменные (флаги), отмечающие факт использования ФД в данном ФВ,
  - Удаление ФД (и флагов) по готовности всех флагов

- Следствия:
  - разрастание кода
  - Потенциальные ошибки

Реализация задачи в языке LuNA: трудности

Сериализация ввода-вывода

- Проблема: неупорядоченный ввод/вывод
  - Пример:
    - **for** i = 1..N print(#in x[i]);

- Вынужденное решение – фиктивные зависимости по данным
  - Пример 1:
    - **for** i = 1..N print(#in x[i], seq[i], #out seq[i+1]);
  - Пример 2:
    - **sub** print_vector(#in **int** n, **name** x, **int** seq_in, #out **name** seq_out) {
    - **df** seq;
    - copy(#in seq_in, #out seq[0]);
    - **for** i = 1..n print(#in x[i], seq[i], #out seq[i+1]);
    - copy(#in seq[n], #out seq_out);
    - }
- Следствие:
  - разрастание кода

Реализация задачи в языке LuNA: трудности

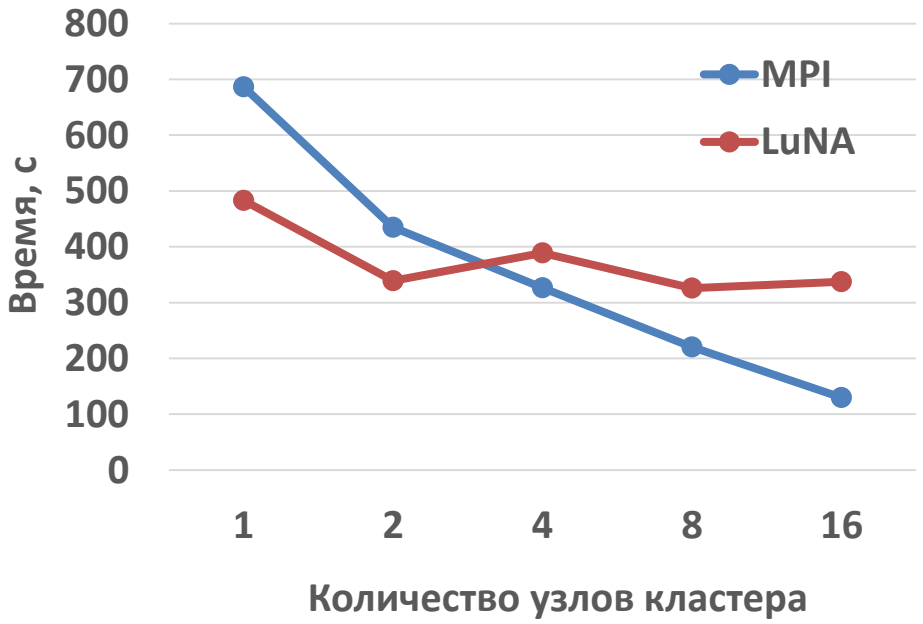
Неконтролируемый фронт исполнения ФВ

- Проблема: массовое порождение ФД
  - Пример:
    - **for** i = 1..N {
    - count(#in x, #out y[i]);
    - print(#in y[i]);
    - }

- Вынужденное решение – фиктивные зависимости по данным
  - Пример:
    - copy(#in 1, #out ready[0]);
    - **for** i = 1..N **if** (ready[i]) {
    - count(#in x, #out y[i]);
    - print(#in y[i]);
    - set_ready(#in y[i], #out ready[i+1]);
    - }

Оценка производительности: MPI vs LuNA

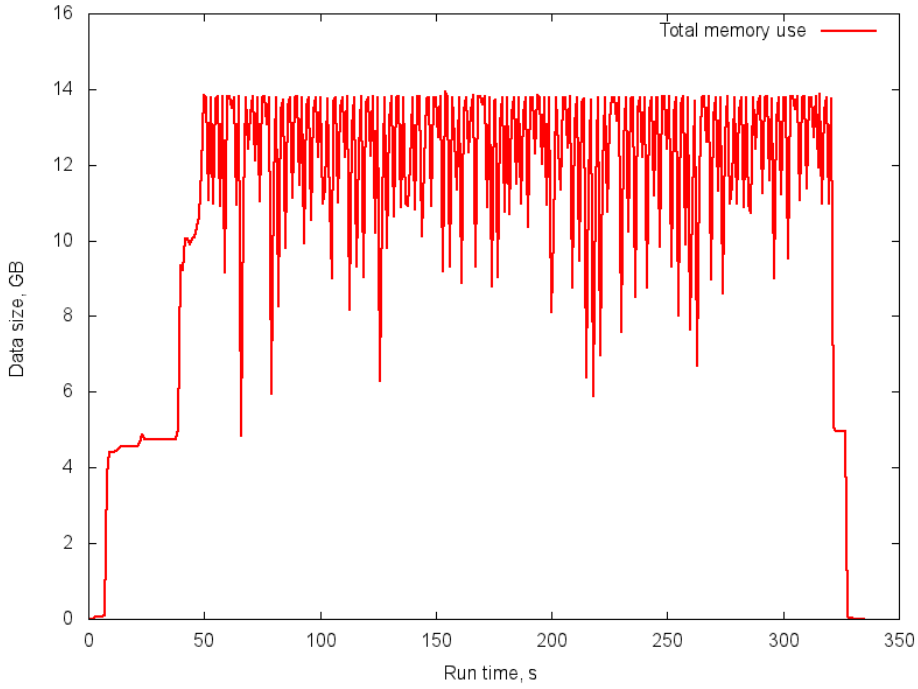
- Параметры задачи:
  - Сетка: 200 × 200 × 200
  - Количество частиц: 100 000 000
    - Начальное распределение – вращающийся плоский диск
  - Число шагов по времени: 100
- Декомпозиция пространства:
  - MPI-программа: X × Y × 1
  - LuNA-программа: 16 × 16 × 1
- Вычислительные ресурсы:
  - Кластер МВС-10П
    - 16 ядер на узел



Вывод: на малом числе узлов производительность сравнимая

Оценка потребления памяти

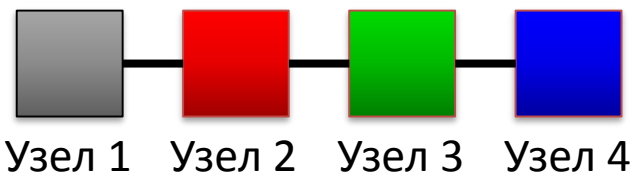
- Параметры задачи:
  - Сетка: 200 × 200 × 200
  - Количество частиц: 100 000 000
    - Начальное распределение – вращающийся плоский диск
  - Число шагов по времени: 100
- Декомпозиция пространства:
  - LuNA-программа: 16 × 16 × 1
- Вычислительные ресурсы:
  - Кластер МВС-10П
    - 8 узлов × 16 ядер
- Объём данных: ~5 GB

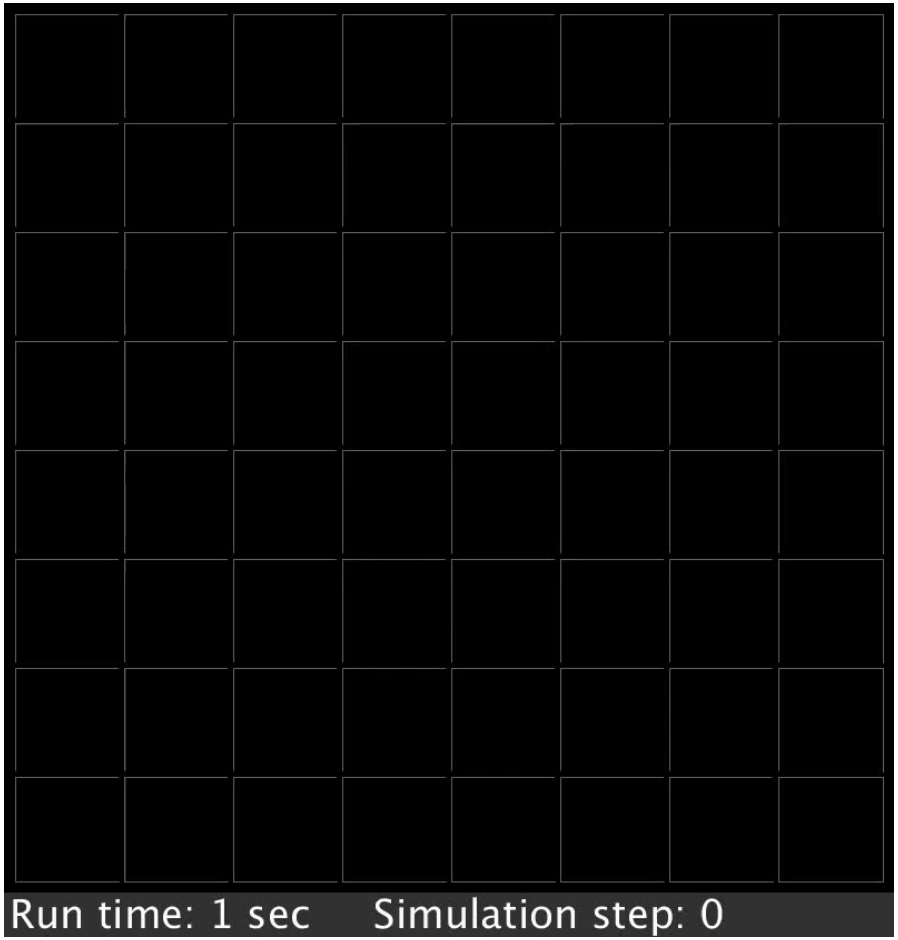


Вывод: расход памяти разумный, утечек памяти нет

Тестирование динамической балансировки нагрузки

- Параметры задачи:
  - Сетка: 200 × 200 × 200
  - Количество частиц: 10 000 000
    - Начальное распределение – вращающийся плоский диск
  - Число шагов по времени: 501
- Декомпозиция пространства:
  - LuNA-программа: 8 × 8 × 1
- Вычислительные ресурсы:
  - Кластер МВС-10П
    - 4 узла × 16 ядер





(цвет подобласти обозначает узел, на котором она обрабатывается)

Выводы

- Язык LuNA позволяет описывать сложные численные модели
- Система LuNA позволяет производить расчеты на кластере, причём на малом числе узлов показывает производительность близкую к MPI
- «Недостатки» языка и системы LuNA:
  - Вследствие статичной структуры LuNA-программы описание всевозможных реализаций трудоёмко и приводит к большому размеру программы
  - Отсутствие приемлемой автоматической сборки мусора усложняет программирование
  - Отсутствие поддержки сериализации ввода-вывода также усложняет программирование
  - Неконтролируемый фронт исполнения усложняет программирование и уменьшает возможный параллелизм
  - Плохая масштабируемость
  - Динамическая балансировка работает пока плохо

Спасибо за внимание