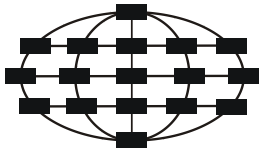
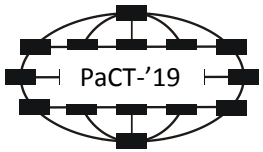


Лаборатория синтеза параллельных программ ИВМиМГ СО РАН



Parallel Computing Technologies – 2019

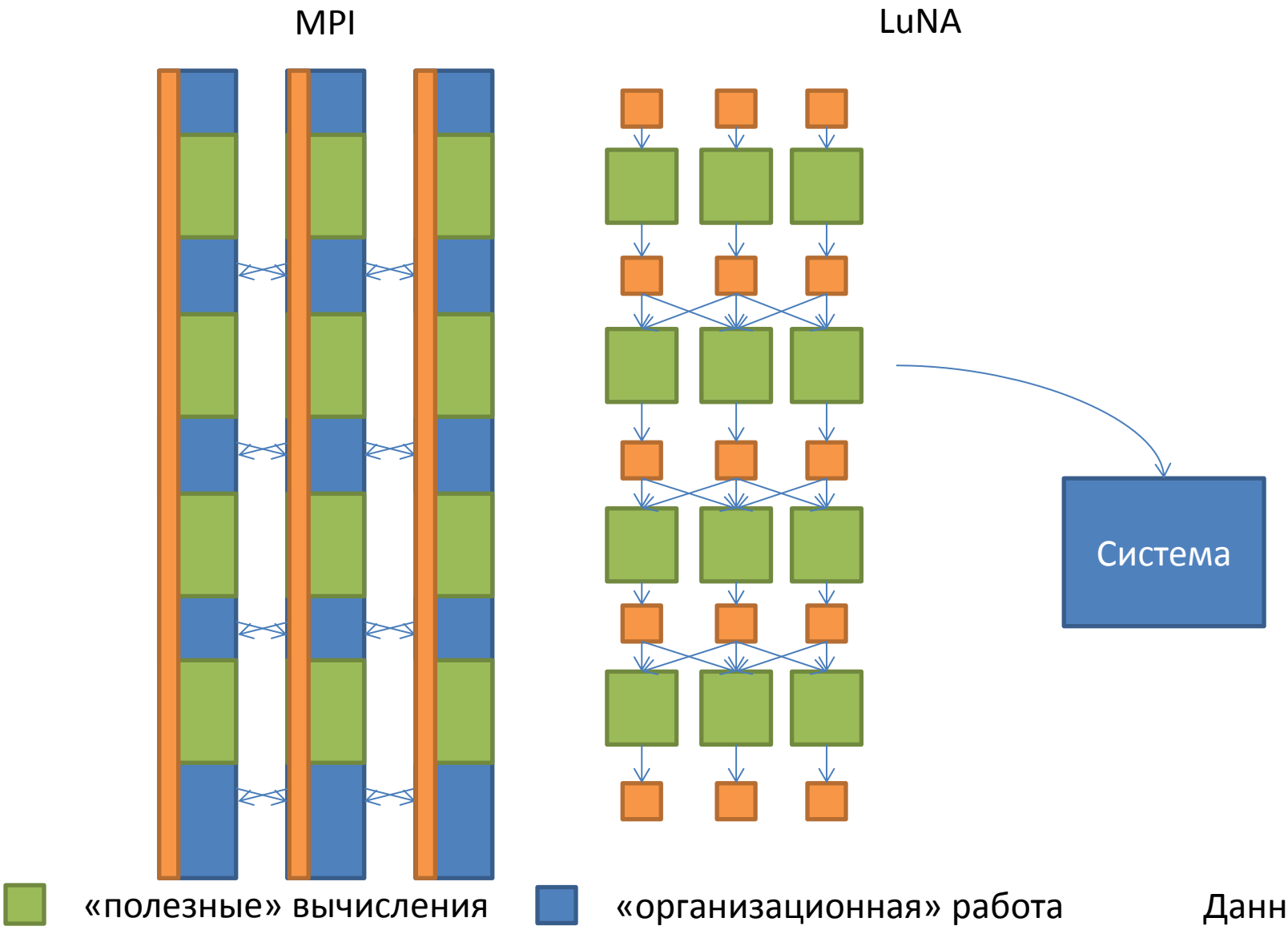
Обеспечение приемлемой эффективности исполнения LuNA-программ

В.А. Перепёлкин

Система конструирования параллельных программ численного моделирования **LuNA** *(**L**ang**u**age for **N**umerical **A**lgorithms)*

- Актуальна проблема автоматизации конструирования параллельных программ численного моделирования
- Создаются и развиваются системы параллельного программирования
- Особенности системы LuNA:
  - Задачи научного моделирования, ресурсонезависимое описание алгоритма, метод структурного синтеза

Модель вычислений в системе LuNA



Особенности подхода

- Крупнозернистый параллелизм
- Вычислительная часть явно отделена от организационной
- Системная часть реализуется автоматизированно

Аббревиатуры:
ФА — фрагментированный алгоритм
ФП — фрагментированная программа
ФВ — фрагмент вычислений
ФД — фрагмент данных

Внешний вид LuNA-программы

```
import c_init(int, name) as init;
import c_init_submatrix(int #file, int #row, int #col, int #height,
import c_mult_matrix(value #a, value #b, name #c) as mult_mat;
import c_sum_matrix(value #a, value #b, name #c) as sum_mat;
import c_save_submatrix(value #a, int #row, int #col) as save_mat;
import c_copy_matrix(value #a, name #b) as copy_mat;

##const FG_SIZE 500
##const FG_COUNT 4

sub calc_mat(name A, name B, name C, int i, int j, int N)
{
        df Ctmp, Csum;

        for k=0..N-1
        {
                cf f[i][j][k]: mult_mat(A[i][k], B[k][j], Ctmp[k]);
        }

        if N>1
```

```
extern "C"
void c_init(int val, OutputDF &df)
    df.setValue<int>(val);
    printf("c_init: %d --> %s, size
}

extern "C"
void c_init_submatrix(const char* /
        struct Matrix m;

        create_matr_buf(dest, height, width);
        m.Height = height;
        m.Width = width;
    m.Data = (double*)(dest.getData<size_t>() + 2);

        mf_init_random(&m, 1, 2);
```

Ключевой вопрос: качество реализации фрагментированного алгоритма

- Численным параллельным программам необходима эффективность (по времени выполнения, требуемым ресурсам и пр.)
- Эффективная реализация алгоритма — труднорешаемая задача
- Важно:
  - Представление (прикладного) алгоритма
  - Системные алгоритмы
  - Применяемые эвристики

Поведение

- Поведение — это множество всех возможных реализаций ФА
- Поведение может быть уточнено:
  - Распределение фрагментов по узлам
  - Ограничения на порядок исполнения ФВ
  - …
- Создание практичных «поведенческих операторов» — отдельная проблема
- Эффективная реализация ФА — это вопрос выбора и реализации поведения
- ФП — это ФА с уточнённым поведением

Рекомендации

- Рекомендации — это способ на практике частично преодолеть проблему выбора и реализации поведения
- Виды рекомендаций:
  - Уточнение поведение (выбор подмножества)
  - Описание трудновыявляемых свойств ФА
- Источники рекомендаций:
  - Человек (прикладной и/или системный программист)
  - Статический анализ
  - Профилирование

Внешний вид рекомендаций

```
while max[t]>$EPS, t=0..out iters
{
        poi_part(Ro[0], Fi[t][0], FiU[t][0], FiU[t][1], 0,
                        Fi[t+1][0], FiU[t+1][0], FiD[t+1][0], lmax[t+
                --> (Fi[t][0], FiU[t][0], FiU[t][1]) @ {
                stealable;
                //request Ro[0], Fi[t][0], FiU[t][0], FiU[t][1];
                locator_cyclic:0;
                //req_unlimited Fi[t+1][0], FiU[t+1][0], FiD[t+1][0],
        };
        for i=1..$FG_COUNT-2 {
                poi_part(Ro[i], Fi[t][i], FiD[t][i-1], FiU[t][i+1], i
                                Fi[t+1][i], FiU[t+1][i], FiD[t+1][i],
                        -->(Fi[t][i], FiD[t][i-1], FiU[t][i+1]) @ {
                        stealable;
                        //request Ro[i], Fi[t][i], FiD[t][i-1], FiU[t
                        locator_cyclic:i/11;
                        //req_unlimited Fi[t+1][i], FiU[t+1][i], FiD[
                };
        } @ {
```

Требования к системе по эффективности

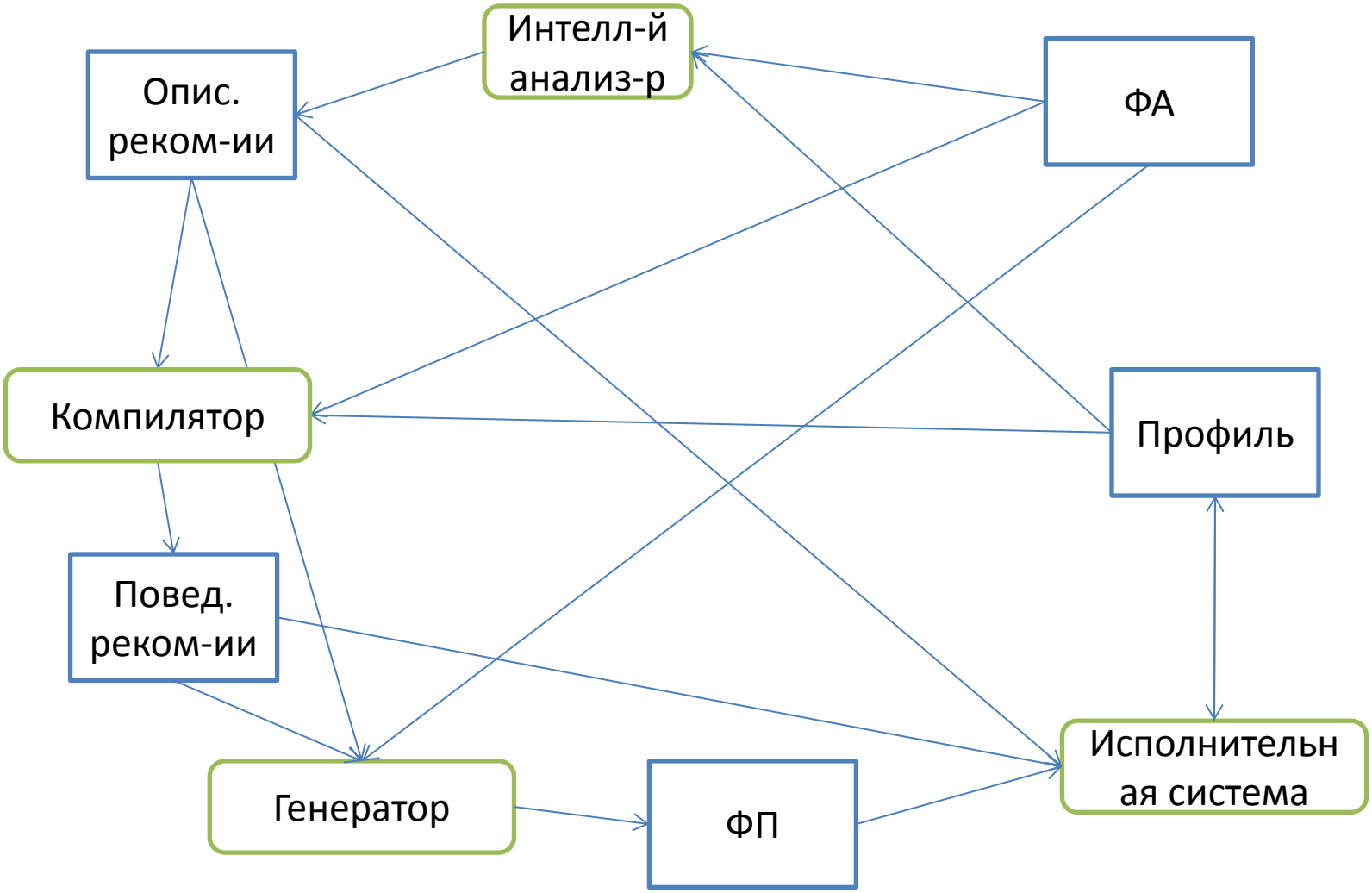
- Описание ФА должно быть ресурсонезависимым
- Отделяемость, расширяемость и повышение уровня рекомендаций
- Возможность независимо управлять поведением
- При условии «хорошо» заданного поведения исполнение должно быть эффективным
- В отсутствии рекомендаций ФА должен быть исполнен (хотя бы неэффективно)

Существенные статьи расходов

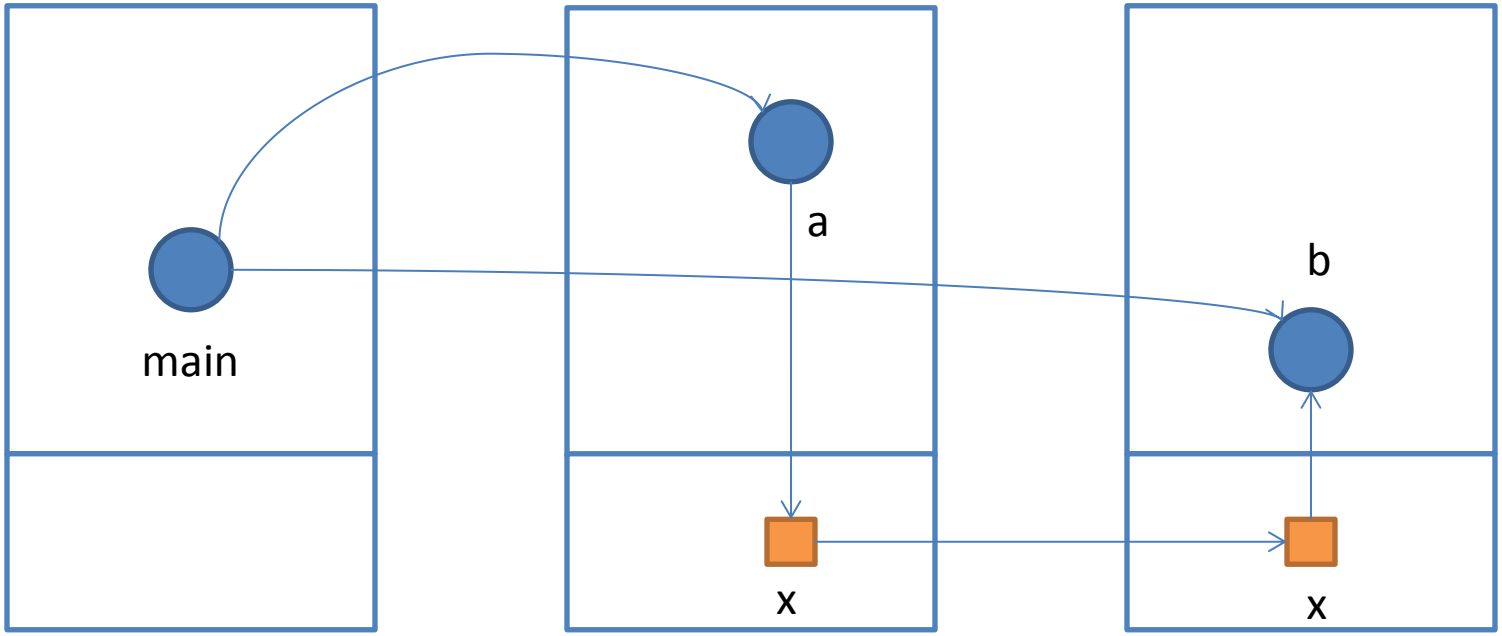
- Фрагменты данных (копии, хранение, удаление)
- Фрагменты вычислений
- Сетевые сообщения

Условие: «достаточно крупные» фрагменты

Архитектура системы



Распределённый алгоритм исполнения ФА




```
sub main {
        df x;
        cf a: f -> x;
        cf b: x -> g;
}
```

Концепция локаторов

- Локатор — это способ найти объект
  - Примеры: locator_cyclic <выражение>, rope1d <выражение>
- Локатор отвечает на вопрос: «какой узел следующий в маршруте до искомого места?»
- Локатор «означивается» динамически и (обычно) локально

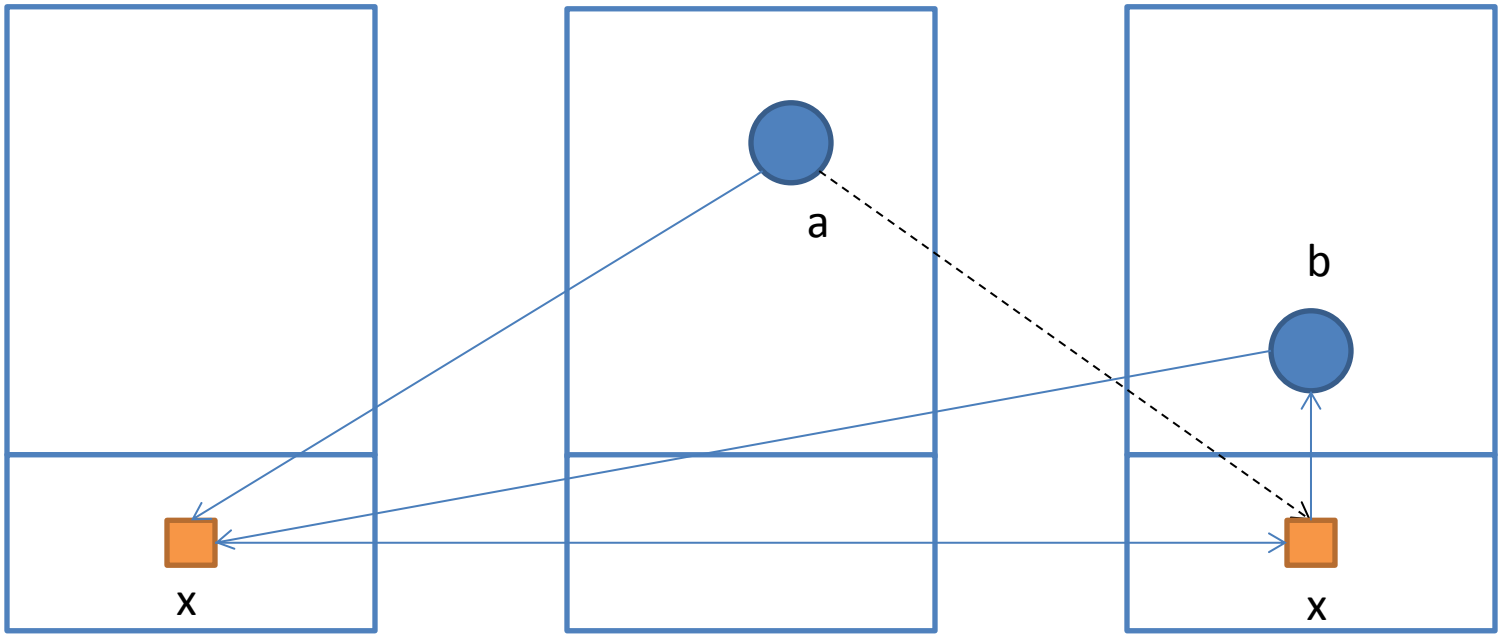
Стратегии раскрутки циклов

- Проблема: перечисляющий алгоритм (LuNA-программа) может перечислять фрагменты не в том порядке, в котором они должны исполняться
- Циклы (for, while) и рекурсия могут порождать огромные множества фрагментов раньше времени
- Решение: рекомендациями определять стратегии раскрутки (перечисления)
  - unroll_at_once
  - unroll(5)

Сборка мусора

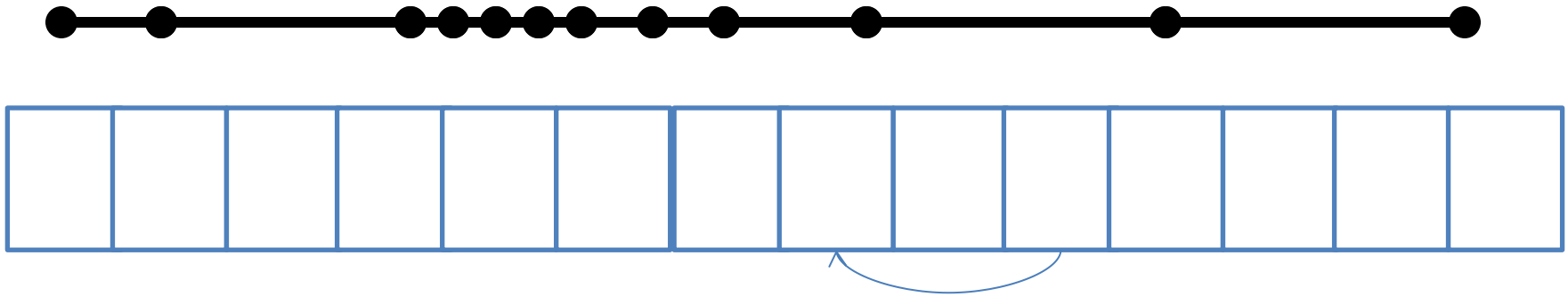
- Удаление ФД по срабатыванию ФВ
- Удаление ФД по счётчику потреблений
- Упреждающая посылка с последующим удалением
- По умолчанию убран алгоритм по завершению области видимости

Упреждающая посылка ФД

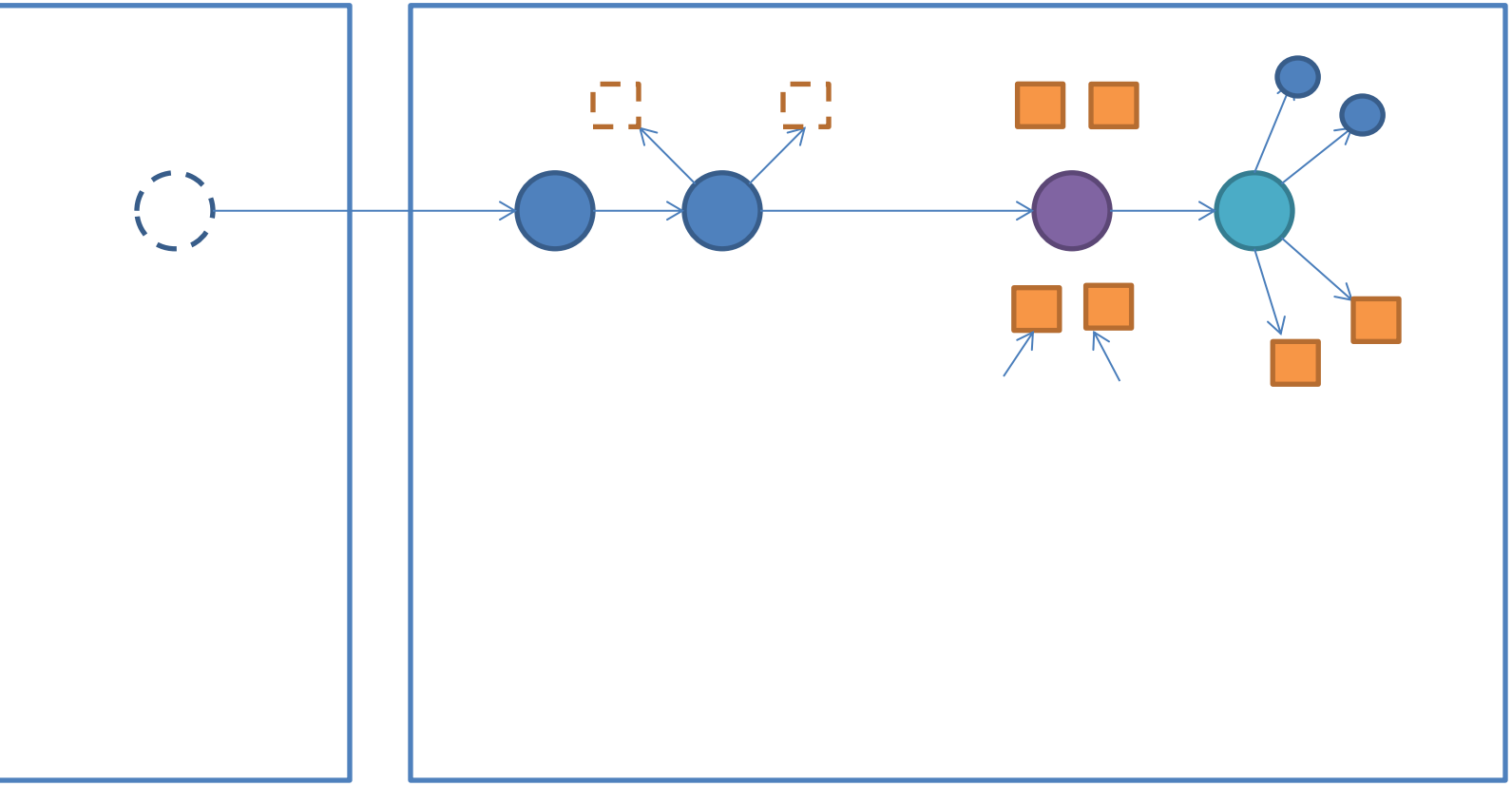


- Производство-потребление ФД через хранилище
- Прямая отправка ФД потребителю
- Возможно: дублирование по всем узлам

Динамическая балансировка нагрузки



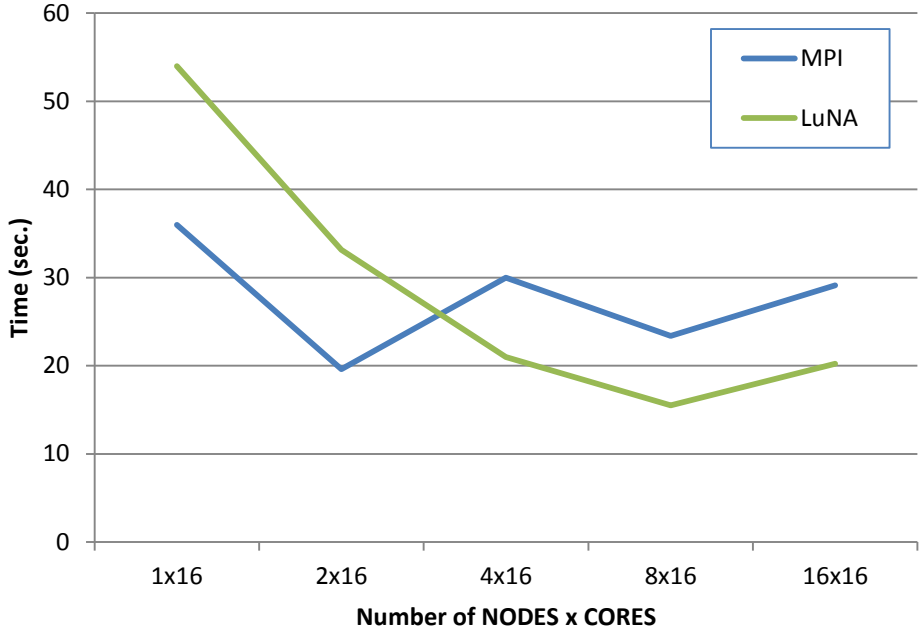
Переход от декларативного к императивному представлению



«Системные вызовы» процесса

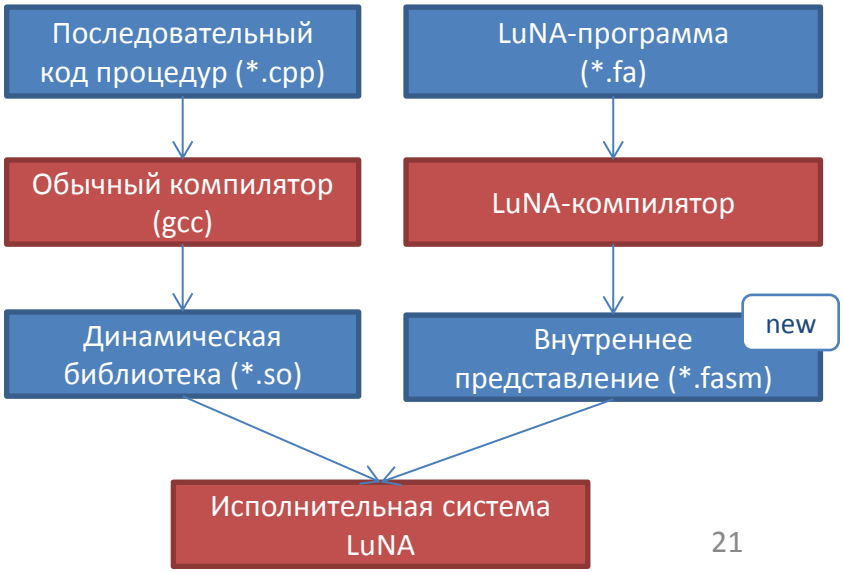
```
53          DF wait(const Id &id);
54
55          Id create_id();
56
57          CF *fork(int block, const std::string &caption="");
58
59          void post(const Id &id, const DF &val, const Locator &,
60                  int req_count);
61
62          void request(const Id &id, const Locator &);
63
64          bool migrate(const Locator &loc);
65
66          void destroy(const Id &id, const Locator &);
67
68          void store(const Id &id, const DF &val);
69
70          void push(const Id &dfid, const DF &val, const Id &cfid,
71                  const Locator &);
72
73          void expect_pushes(const Id &cfid);
74
```

Тестирование производительности. Предварительное тестирование показало высокую эффективность выполнения LuNA-программы, сравнимую с ручной реализацией прикладного алгоритма средствами MPI (ранее LuNA давала отставание по времени на 2-3 порядка).



Модельная задача: решение уравнения Пуассона итерационным методом.
Размер сетки: 1600x100x1600
Количество итераций: 68
Декомпозиция: 1D (на слои по числу ядер)
MPI: обычная реализация
(1 MPI-процесс на каждое ядро)
LuNA: 1 процесс на узел, 1 поток на каждое ядро
Вычислитель: кластер МСЦ MVS-10p

Выполненные оптимизации:
—На этап компиляции перенесено планирование исполнения фрагментов вычислений
— В систему LuNA добавлены новые возможности по контролю исполнения фрагментированной программы (упреждающая посылка фрагментов данных, сборка мусора по счётчику потреблений, рекомендации по распределению ресурсов)
— Модернизирован ряд системных модулей (программные оптимизации без изменения системных алгоритмов)



Заключение

- Рассмотрена проблема обеспечения удовлетворительной эффективности исполнения LuNA-программ
- Разработаны требования к системе с т.з. эффективности
- Предложены архитектура и алгоритмы системы
- Проведено предварительное тестирование эффективности на модельной задачи

Дальнейшие планы:

- Исследование эффективности на других классах задач
- Совершенствование алгоритмов анализа и исполнения LuNA-программ
- Совершенствование языка описания рекомендаций