Алгоритм генерации прямого управления в системе LuNA на базе событийно-ориентированной модели вычислений

Докладчик: Ткачёва А.А., аспирантка ИВМиМГ СО РАН

Летняя международная XXIX молодежная Школа-конференция по параллельному программированию
3-14 июля 2017

Введение

Система фрагментированного программирования LuNA, предназначена для автоматизации процесса реализации больших задач численного моделирования на суперкомпьютере.

Фрагментированная программа

Фрагментированная программа (ФП) представляет собой двудольный ориентированный граф, вершинами которого являются множество фрагментов вычислений (ФВ) и фрагментов данных (ФД), а дуги – информационные зависимости между фрагментами.

ФВ реализуется вызовом функции без побочных эффектов для некоторого набора входных ФД.

ФП исполнена, когда все ФВ исполнены. Порядок исполнения ФВ основан только на информационных зависимостях.

В системе LuNA реализуется в динамике потоковое управление (dataflow).

Проблема (1)

В системе LuNA прикладной алгоритм представляется в явной параллельной форме. Она не содержит ограничений по исполнению и распределению ресурсов, которые появляются, если мы оптимизируем исполнение под конкретный вычислитель и учитываем специфику входных данных.

Проблема (2)

В представлении ФП определено множество вариантов исполнения, автоматический выбор лучшего (в смысле времени исполнения, потребления памяти) варианта исполнения труднорешаемая задача.

Под **поведением** будем понимать множество всех вариантов исполнения программы.

Глобальная цель работы

Разработать средства задания поведения ФП для оптимизации их исполнения в распределенной памяти и алгоритмы их автоматизированного применения в системе LuNA.

Runtime-система LuNA

Runtime-система LuNA является полу-интерпретатором ФП. В частности при исполнении ФП она осуществляет следующие функции:

- выделение ресурсов вычислителя для ФВ и ФД;
- выбор из множества всех ФВ множество ФВ готовых к исполнению;
- планирование вычислений (какой ФВ из множества готовых к исполнению ФВ исполнить: один, все или не одного и т.д.);
- обеспечение динамических свойств ФП (например, динамическая балансировка нагрузки).

Виды накладных расходов

1. Накладные расходы на организацию вычислений внутри узла
2. Накладные расходы на организацию вычислений между узлами
3. Накладные расходы на коммуникации

Такие накладные расходы характерны для параллельных программ, написанных для задач численного моделирования.

Фреймворк LuNAFW

Для оптимизации исполнения ФП в распределенной памяти был разработан фреймворк LuNAFW, в котором управление вычислениями задается с помощью управляющих программ.

Управляющая программа пишется для конкретной ФП, и в ней заложены частичные решения о распределении ресурсов и порядке исполнения операций.

Управляющая программа

Управляющая программа, которая представляет собой C++ класс для исполнения в распределенной и общей памяти.

Следующие функции в управляющей программе должны быть определены:

- **onInit ()** – функция начального значения для начальной инициализации.
- **onComputed (df_id)** – функция, вызываемая после того как каждый ФД с идентификатором df_id вычислен.
- **onReceived (df_id)** – функция, вызываемая после того как каждый ФД с идентификатором df_id получен от другого узла.
- **onCfFinished (cf_id)** – функция, вызываемая после того как каждый ФВ с идентификатором cf_id завершил исполнения.

Действия, поддерживаемые LuNAFW

Внутри выше описанных функций могут быть вызваны следующие функции (действия), поддерживаемые фреймворком LuNAFW:

- **startCF** (CF description) – запустить исполнение ФВ.
- **checkCF** (CF description) – если все ФД доступны, то вызвать функцию **startCF** для CF.
- **destroyDF**(df_id) – удалить ФД с идентификатором df_id.
- **sendDF** (df_id, rank) - послать ФД с идентификатором df_id на процесс rank.
- **exit()** – завершить работу фреймворка.
- **getRank**(identifier) - функция, выдающая номер процесса, на который ФВ распределен.

Фреймворк LuNAFW

Преимущества по сравнению с LuNA:

- Часть решений по распределению ресурсов и управлению принимаются на стадии компиляции, следовательно меньше накладных расходов по сравнению с базовым алгоритмом исполнения системы LuNA.

Преимущества по сравнению с MPI:

- Программирование в терминах фрагментов.

Применение фреймворка для практических задач

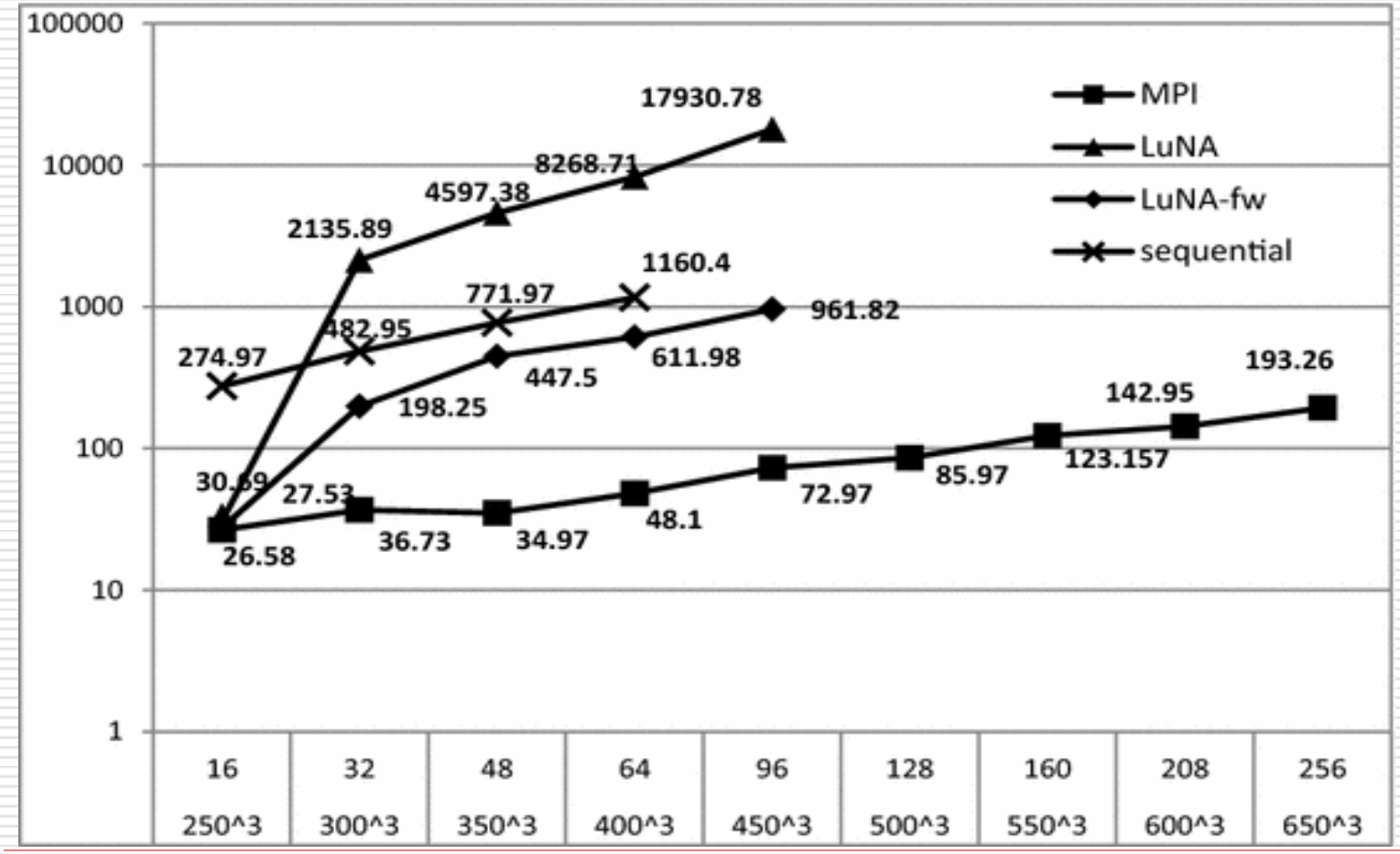
На LuNAFW были реализованы следующие задачи:

- Задача решение краевой задачи фильтрации трехфазной жидкости (нефть-вода-газ)[1]
- Метод IADE-RB-CG[2]

[1]Akhmed-Zaki D.Z., Lebedev D.V., Perepelkin V.A. (2017) Implementation of a three dimensional three-phase fluid flow (“oil–water–gas”) numerical model in LuNA fragmented programming system. Journal of Supercomputing, Volume 73, Issue 2, pp 624–630.

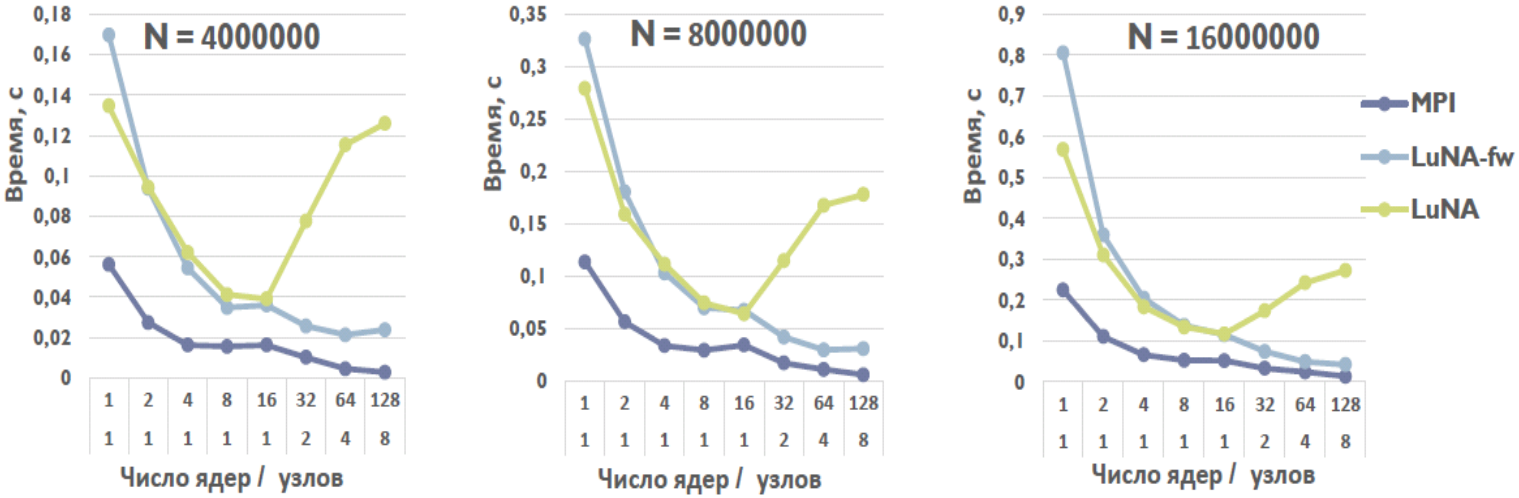
[2]Киреев С.Е., Перепелкин В.А., Ткачёва А.А. Реализация метода IADE_RB_CG в системе фрагментированного программирования LuNA // Вос. Сиб. конф. по параллельным и высокопроизводительным вычислениям: труды конф., Томск: изд-во Том.Ун-та, 2015, с. 66-72.

Задача решение краевой задачи фильтрации трехфазной жидкости (нефть-вода-газ), mvs10p



Тестирование:
Сравнение различных реализаций

IADE метод



- Выводы
  - В пределах узла варианты LuNA и LuNA-fw имеют сравнимую производительность
  - При использовании более одного узла производительность варианта LuNA падает
  - Вариант MPI обгоняет LuNA-fw примерно в 2 раза

Генерация управляющих программ для ФП частного вида

Написанные вручную управляющие программы для представленных выше задач, показали, что с использованием фреймворка LuNAFW можно получить приемлемую производительность исполнения ФП.

Следующий шаг – это их генерация для ФП частного вида.

Цель работы

Разработать алгоритм генерации управляющих программ для оптимизации исполнения ФП в распределенной памяти.

Для этого для подпрограмм из ФП частного вида генерировать управляющую программу, в которой заложены частные решения о распределении ресурсов и порядке исполнения операций.

Вход алгоритма генерации (1)

Фрагментированная подпрограмма:

- Описание ФВ
- Множество ФВ, описанных через языковую конструкцию for

Вход алгоритма генерации (2)

Фрагментированная подпрограмма:

- Описание ФВ
  - Идентификатор ФВ
  - Множество входных ФД
  - Множество выходных ФД
  - Имя функции без побочных эффектов
  - Множество ФД, которые следует уничтожить после завершения исполнения ФВ (*опциальный параметр)*
- Множество ФВ, описанных через языковую конструкцию for

Вход алгоритма генерации (3)

Фрагментированная подпрограмма:

- Описание ФВ
  - …
- Множество ФВ, описанных через языковую конструкцию for
  - Переменная счетчика цикла
  - Нижнее граничное значение (должно быть целым числом)
  - Верхнее граничное значение (должно быть целым числом)
  - Множество ФВ или множества ФВ описанных через for

Требования к входным данным

- Информационные зависимости между ФВ можно проанализировать на стадии компиляции.
- Индексная форма идентификатора может состоять из:
  - Целочисленных констант (a[0])
  - Переменных счетчика цикла (a[i])
  - Выражений вида +/- константа (a[i+1])

Выход алгоритма генерации

Управляющая программа, которая представляет собой C++ класс для исполнения в распределенной и общей памяти.

Следующие функции в управляющей программе должны быть определены:

- **onInit ()** – функция начального значения для начальной инициализации.
- **onComputed (df_id)** – функция, вызываемая после того как каждый ФД с идентификатором df_id вычислен.
- **onReceived (df_id)** – функция, вызываемая после того как каждый ФД с идентификатором df_id получен от другого узла.
- **onCfFinished (cf_id)** – функция, вызываемая после того как каждый ФВ с идентификатором cf_id завершил исполнения.

Действия, поддерживаемые LuNAFW

Внутри выше описанных функций могут быть вызваны следующие функции (действия), поддерживаемые фреймворком LuNAFW:

- **startCF** (CF description) – запустить исполнение ФВ.
- **checkCF** (CF description) – если все ФД доступны, то вызвать функцию **startCF** для CF.
- **destroyDF**(df_id) – удалить ФД с идентификатором df_id.
- **sendDF** (df_id, rank) - послать ФД с идентификатором df_id на процесс rank.
- **exit()** – завершить работу фреймворка.
- **getRank**(identifier) - функция, выдающая номер процесса, на который ФВ распределен.

Этапы работы алгоритма генерации:

1. Конвертер – конвертация ФП из потоковой модели вычислений в модель типа event-driven
2. Генератор – генерация управляющий программы из выходных данных Конвертера, учитывая распределенное исполнение ФП.

Конвертер

Входные данные для конвектора такие же, как и для алгоритма генерации. Выходные данные Конвертера состоят из:

- *Init* - список ФВ идентификаторов, у которых нет входных ФД.
- *B* - список ФВ идентификаторов, у которых нет выходных ФД.
- *Out* - список выходных ФД ФП.
- Словарь *GarCol*:
  - *Ключ* - ФВ идентификатор.
  - Значение - список идентификаторов ФД, которые следует уничтожить после завершения исполнения ФВ с идентификатором *ключ*. Если идентификатор ФВ или ФД имеет индексную форму, то также записываются границы использования индексов.
- Словарь *DAG*:
  - *Ключ* - ФД идентификатор.
  - Значение - список идентификаторов ФВ, для которых *ключ* – входной ФД. Если идентификатор ФВ или ФД имеет индексную форму, то также записываются границы использования индексов.

Порядок исполнения ФВ с помощью сгенерированной управляющей программы не противоречит информационным зависимостям исходной ФП.

Предполагается, что представленный алгоритм можно будет использовать для итерационных задач численного моделирования, редукции, умножении матрицы на вектор и т.д.

Преимущества

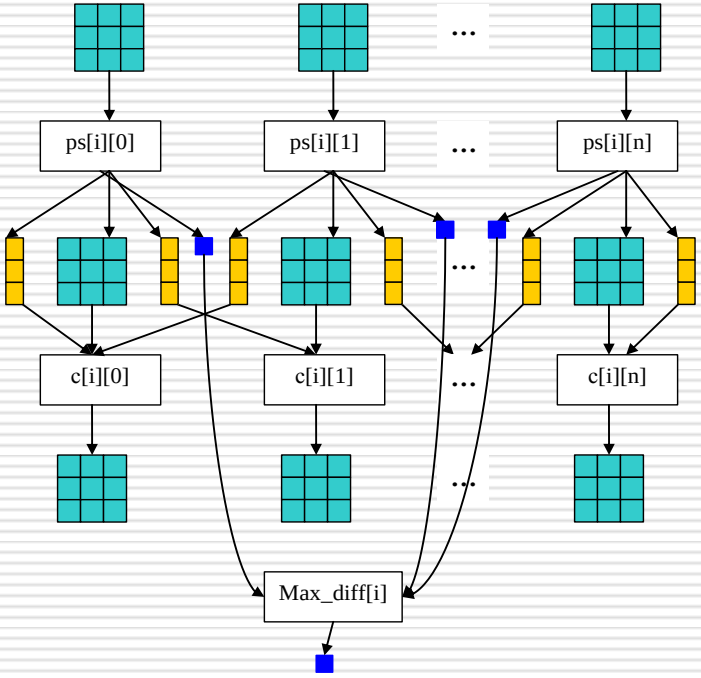
1. В LuNAFW нет менеджера ФД (ФД посылается на нужный узел без необходимости дополнительных запросов)
2. Т.к. применяем для случая, когда динамика не нужна, то не необходимости поддерживать локальный алгоритмы распределения ресурсов, а можно посылать сразу на нужный узел.

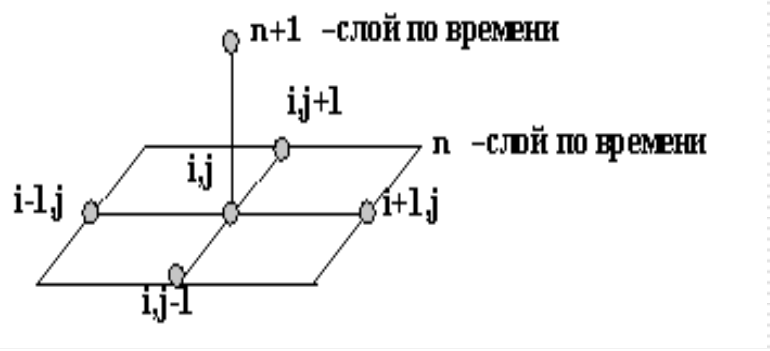
Тестирование на прикладных задачах

Решение уравнения Пуассона явным методом:

- Одномерная декомпозиция данных
- Трехмерная декомпозиция данных

Решение уравнения Пуассона явным методом (i-итерация 1D декомпозиция)





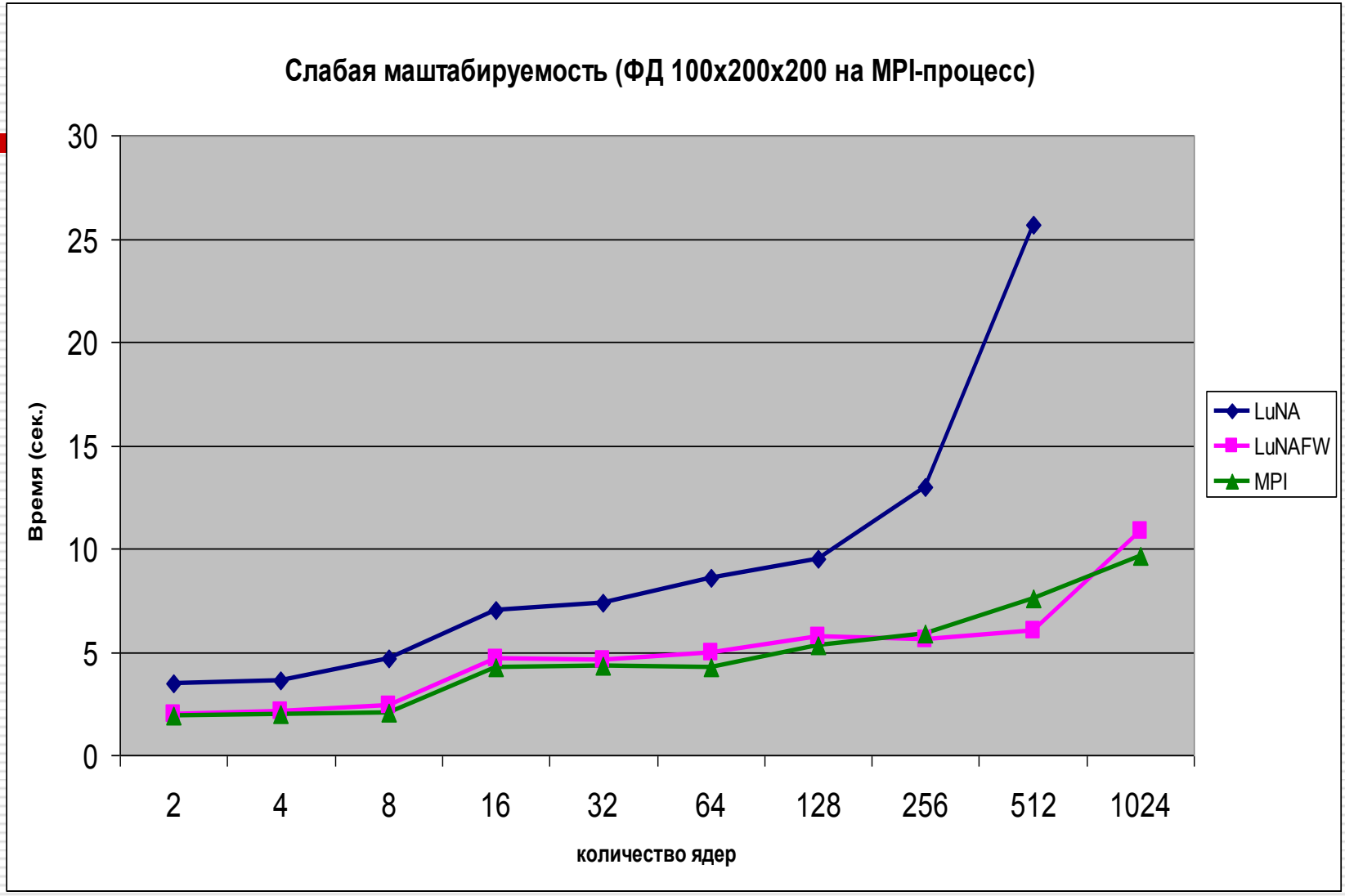
Исследование производительности

Вычислитель: mvs10p (16 ядер на узел)

- 1 поток в системе LuNA.
- 1 MPI процесс на ядро.
- 1 ФВ на 1 MPI процесс.

Параметры задачи:

- Количество итераций 20.
- Размер ФД 100x200x200 на 1 MPI процесс


Слабая маштабируемость (ФД 100x200x200 на MPI-процесс)
Время (сек.)
0
5
10
15
20
25
30
2
4
8
16
32
64
128
256
512
1024
количество ядер
LuNA
LuNAFW
MPI

Результаты

Автоматически сгенерированная управляющая программа позволила улучшить производительность исполнения ФП на 40% по сравнению с базовым алгоритмом исполнения ФП run-time системы LuNA.

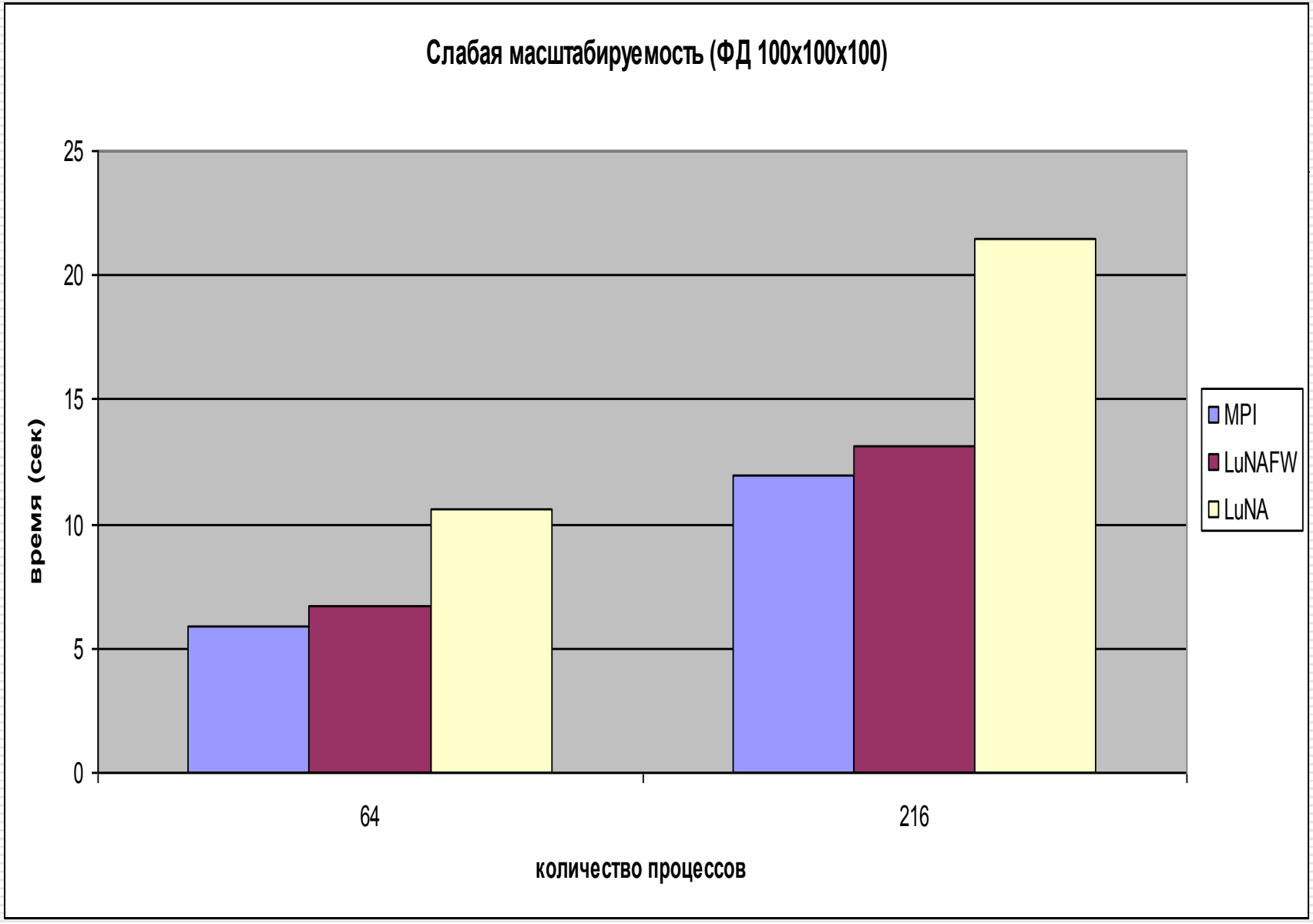
Решение уравнения Пуассона явным методом 3D декомпозиция

Вычислитель: mvs10p (16 ядер на узел)

- 1 поток в системе LuNA.
- 1 MPI процесс на ядро.
- 1 ФВ на 1 MPI процесс.

Параметры задачи:

- Размер ФД 100x100x100.
- Количество итераций 10.


Слабая масштабируемость (ФД 100x100x100)
25
20
15
10
5
0
время (сек)
64
216
количество процессов
MPI
LuNAFW
LuNA

Выводы

- Был разработан алгоритм генерации управляющих программ для оптимизации исполнения ФП частного вида.
- Проведено сравнительное тестирование производительность исполнения ФП предлагаемым подходом по сравнению с базовым алгоритмом исполнения runtime системы LuNA и аналогичной реализации на MPI на задаче решения уравнения Пуассона явным методом одномерная декомпозиция и трехмерной декомпозиции данных.
- Сгенерированная управляющая программа позволяет получить сравнимую с MPI производительность и улучшить время исполнения ФП на 40% по сравнению с базовым алгоритмом исполнения в системе LuNA.

Дальнейшие планы

- Расширить применимость алгоритма генерации управляющих программ
- Провести исследование производительности для автоматически сгенерированных управляющих программах на других задачах.

Спасибо
за внимание!