ОТЧЕТ О ПРОДЕЛАННОЙ РАБОТЕ С ИСПОЛЬЗОВАНИЕМ ОБОРУДОВАНИЯ ИВЦ НГУ

1. Аннотация

Параллельное программирование задач численного моделирования эффективнее последовательного. Но разработать эффективную параллельную программу непросто, так как это требует определённой квалификации для решения проблем, связанных с параллельным программированием и лежащих за пределами предметной области. Системы автоматического конструирования параллельных программ частично решают эти проблемы.

Но их применение на практике затрудняется тем, что переписывать весь код уже имеющейся программы на другой язык трудоёмко и затратно. Кроме того, в реальных задачах это, как правило, и не требуется: в них есть часть, которая занимает большую долю времени работы и действительно нуждается в эффективной параллельной реализации.

Отсюда и возникает необходимость обеспечить возможность вставлять в программу вызовы подпрограмм, сконструированных такими системами.

Система LuNA как раз является примером такой системы. И связь нужна с MPI как с низкоуровневым высокопроизводительным и наиболее распространённым средством параллельного программирования.

2. Тема работы

Разработка и реализация средства интеграции LuNA-подпрограмм в традиционные параллельные программы.

3. Состав коллектива

- Капралова Регина Евгеньевна, студентка факультета информационных технологий (ФИТ) НГУ, кафедра параллельных вычислений, 4-й курс, гр. 19201. Адрес электронной почты: r.kapralova@g.nsu.ru.
- Научный руководитель: Маркова Валентина Петровна, к.т.н., доцент каф. ПВ ФИТ НГУ.
- Соруководитель: Перепелкин Владислав Александрович, н.с Лаборатории синтеза параллельных программ ИВМиМГ СО РАН, perepelkin@ssd.sscc.ru.

4. Научное содержание работы

4.1. Постановка задачи

Цель работы: разработать и реализовать средство интеграции LuNA-подпрограмм в традиционные параллельные программы.

Задачи:

- Разработать модель взаимодействия MPI- и LuNA- программ.
- Разработать алгоритмы и структуры данных, необходимые для обеспечения взаимодействия программ.
- Реализовать средство интеграции LuNA-подпрограмм в MPI-программы в виде расширения системы LuNA.
- Провести экспериментальные исследования полученного решения.

4.2. Современное состояние проблемы

В мировом обществе уже накоплен опыт решения подобных задач. Существуют разные языки, системы и средства параллельного программирования, в которых также присутствует интероперабельность в разных моделях, а именно, возможность вызова подпрограмм, реализованных с использованием этих средств, из MPI-программ.

Был проведён обзор близких работ, среди которых:

- Charm++.
- Legion.
- Cilk.
- X10.

На основе анализа было заключено, что типового решения данной проблемы нет, и для каждой пары средств необходимо новое частное решение. Чему и посвящена данная работа.

4.4. Подробное описание работы, включая используемые алгоритмы.

4.4.1. Система LuNA

Система LuNA состоит из нескольких экземпляров runtime системы, которые создаются по одному на каждом узле. Они вместе образуют общую распределённую сущность и работают согласованно, представляя собой единую runtime-систему.

На каждом узле находятся объекты двух видов: фрагменты данных и фрагменты вычислений. И те, и другие могут передаваться с узла на узел. Фрагменты вычислений начинают выполняться, как только получены все входные фрагменты данных.

LuNA-программа - это описание такого множества фрагментов. LuNA-программа состоит из подпрограмм.

LuNA-подпрограмма также работает на нескольких узлах. Её входными и выходными параметрами являются фрагменты данных. И выполняется она следующим образом: сначала на вход подпрограммы подаются фрагменты данных, затем подпрограмма отрабатывает с этими данными, после чего выходные фрагменты данных могут быть получены другими подпрограммами.

4.4.2. Модель взаимодействия MPI- и LuNA-программ

Для достижения поставленной цели была разработана модель взаимодействия MPI- и LuNA- программ и проиллюстрирована на рисунке 1.

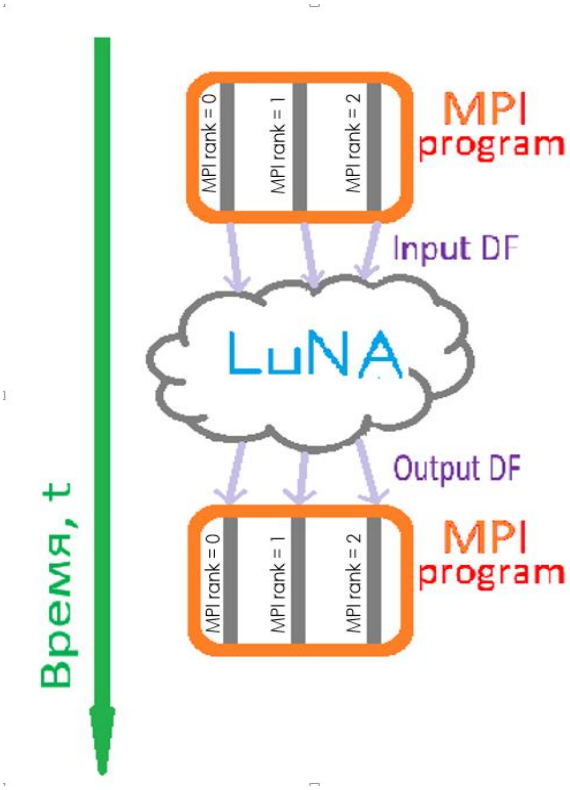

Рисунок 1 – Модель взаимодействия MPI- и LuNA-программ

При вызове LuNA-подпрограммы из MPI-программы экземпляры runtime системы создаются пользователем по одному на каждом MPI-процессе, на которых подпрограмма будет запущена.

Поскольку MPI- и LuNA-программы имеют структуры данных собственного вида. Данные, имеющиеся в MPI-программе, необходимо представить в форме, пригодной для работы с ними в LuNA-подпрограмме и передать системе. Система отработает с этими данными и по запросу вернёт результат.

Согласно разработанной модели взаимодействия, выполнение LuNA-подпрограммы может осуществляться синхронно или асинхронно.

При запуске в синхронном режиме выполнение MPI-программы приостанавливается до тех пор, пока LuNA-подпрограмма не завершится и не вернет управление в MPI.

При запуске в асинхронном режиме создается отдельный поток, в котором будет происходить выполнение подпрограммы. После создания потока, управление возвращается в MPI, и обе программы продолжают выполняться.

4.4.3 Интерфейс

Был разработан и реализован интерфейсный класс SP, который обеспечивает изложенное выше взаимодействие MPI- и LuNA-программ.

У класса SP есть все необходимые методы для передачи аргументов, обмена фрагментами данных и запуска LuNA-подпрограммы из MPI-программы. В конструкторе класса определяется, какая подпрограмма и на каком коммуникаторе будет запущена. Объекты класса создаются на каждом узле, при этом создадутся и экземпляры runtime системы.

Интерфейс класса SP приведен в листинге 1.

Листинг 1 – Файл sp.h

```
1  class SP {
2    public:
3      SP(const std::string &fp_path, const std::string &subprogram_name,
4          MPI_Comm comm = MPI_COMM_WORLD);
5      ~SP();
6      void push_arg(const DF &val);
7      Id push_id();
8      void push_df(Id id, const DF &val, const Locator &loc = CyclicLocator(0));
9      void request_df(Id id, DF &result, const Locator &loc = CyclicLocator(0));
10     void push_df_from(SP* source_sp, Id source_id,
11             const Locator &source_loc = CyclicLocator(0),
12             DF &result, Id target_id,
13             const Locator &target_loc = CyclicLocator(0));
14     int run();
15     void run_acync();
16     void join();
17     void get_thread_object(std::thread* thread_);
18   };
```

Интерфейс предоставляет пользователю следующие возможности:

- Вызывать выбранный фрагмент кода LuNA-программы – аргумент subprogram_name конструктора класса SP.
- Вызывать LuNA-подпрограмму из MPI-программы на заданном коммуникаторе – аргумент comm конструктора класса SP.
- Подавать в подпрограмму аргументы-константы и нефункциональные аргументы – метод push_arg().
- Подавать в подпрограмму аргументы-имена – метод push_id().
- Подавать входные фрагменты данных – метод push_df().
- Получать значения выходных фрагментов данных, которые были инициализированы в ходе выполнения LuNA-подпрограммы – метод request_df().
- Асинхронно передавать фрагменты данных из одной подпрограммы в другую по мере их вычисления в первой подпрограмме – метод push_df_from().
- Запускать LuNA-подпрограмму синхронно – метод run().
- Запускать LuNA-подпрограмму асинхронно – метод run_acync().
- При асинхронном запуске блокировать вызывающий поток до завершения потока выполнения подпрограммы– метод join().
- При асинхронном запуске получать объект std::thread, с которым связан поток выполнения LuNA-подпрограммы – метод get_thread_object().

4.4.4 Тестирование

Для проведения экспериментального исследования были выбраны уже имеющиеся LuNA-программы, из них удалён код инициализации начальных данных и расширена их сигнатура для передачи им уже инициализированных данных.

Первая LuNA-программа вычисляет произведение матриц. Вторая решает уравнение Пуассона методом Якоби.

Затем были реализованы MPI-программы, которые вызывают LuNA-программы, передают им входные данные и получают результаты их работы.

Обе программы выполнились без ошибок и вывели ожидаемый результат. Кроме того, как показано на графиках, приведённых на рисунках 2 и 3, время выполнения MPI-программы, вызывающей LuNA-подпрограмму, оказалось сопоставимым с временем выполнения соответствующей LuNA-программы в обоих случаях.

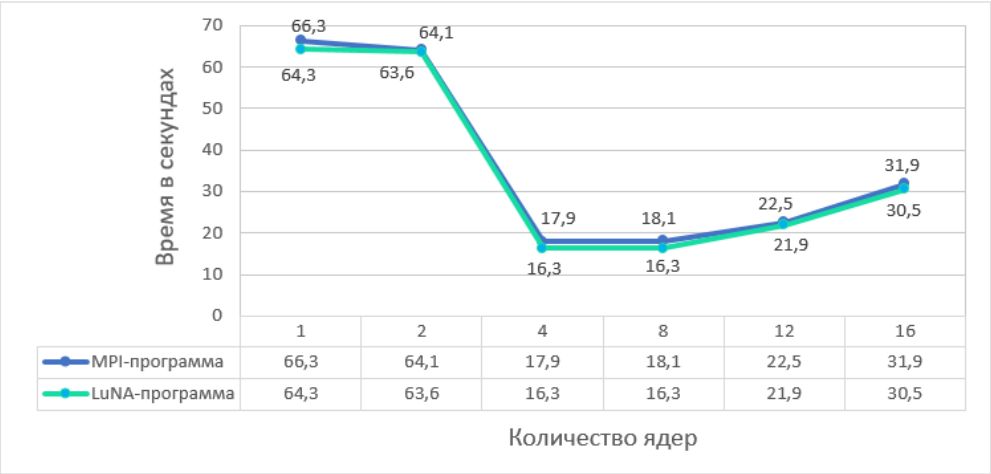

Рисунок 2 – График зависимости времени выполнения LuNA-программы, выполняющей умножение матриц, и MPI-программы, вызывающей данную LuNA-подпрограмму, от количества ядер

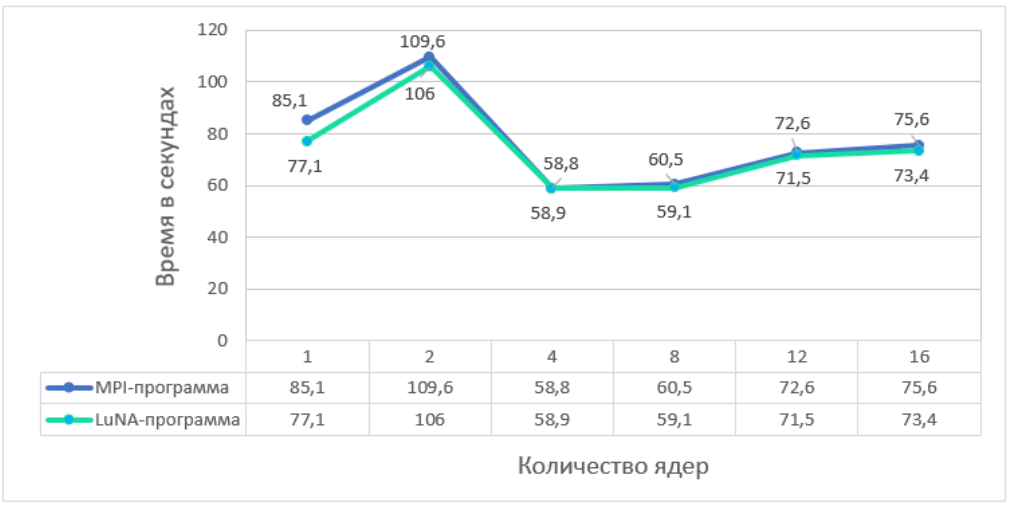

Рисунок 3 – График зависимости времени выполнения LuNA-программы, решающей уравнение Пуассона методом Якоби, и MPI-программы, вызывающей данную LuNA-подпрограмму, от количества ядер

Таким образом, в ходе экспериментального исследования подсистема запуска LuNA-подпрограмм из MPI-программ показала свою работоспособность при разных параметрах и на разных конфигурациях. Кроме того, по итогам исследования было заключено, что применение разработанного программного средства не вносит существенных накладных расходов.

4.5. Полученные результаты

Была обеспечена возможность интеграции LuNA-подпрограмм в традиционные параллельные программы.

В частности, была разработана модель взаимодействия MPI- и LuNA-программ. А также интерфейс, который это взаимодействие обеспечивает. И было реализовано программное средство, которое позволяет вызывать LuNA-подпрограмму из MPI-программы, передавать аргументы, осуществлять обмен данными между программами и выполнять LuNA-подпрограмму на заданном наборе узлов синхронно и асинхронно.

Работоспособность программного средства показана с помощью тестовых MPI-программ, осуществляющих вызов LuNA-подпрограмм: вычисляющей произведение матриц и решающей уравнение Пуассона методом Якоби.

5. Эффект от использования кластера в достижении целей работы.

Использование кластера позволило проверить работоспособность разработанного программного средства и установить, что применение последнего не вносит существенных накладных расходов.

6. Перечень публикаций содержащие результаты работы

Капралова Р. Е. Разработка и реализация средства интеграции LuNA-подпрограмм в традиционные параллельные программы / Капралова Р. Е. //Информационные технологии : Материалы 61-й Междунар. науч. студ. конф. 17–26 апреля 2023 г. / Новосиб. гос. ун-т. — Новосибирск : ИПЦ НГУ, 2023 (Принято в печать).