Разработка прототипа библиотеки на C++ для поддержки вычислений на распределенных массивах

Выполнил:
студент гр. 18201 ФИТ НГУ Баранов Илья Николаевич
Руководитель:
Киреев Сергей Евгеньевич, н.с. ИВМиМГ СО РАН

Цели проекта

- Повысить уровень параллельного программирования – спрятать несущественные детали реализации
  - Для класса задач - задачи на регулярных сетках
- Обеспечить настраиваемость реализации
  - Фрагментация (декомпозиция)
  - Распределение фрагментов по процессам
  - …
- Сохранить (по возможности) производительность, сравнимую с оптимизированной вручную реализацией
  - Проанализировать причины снижения производительности
  - Постараться их преодолеть

Специфика нашего проекта

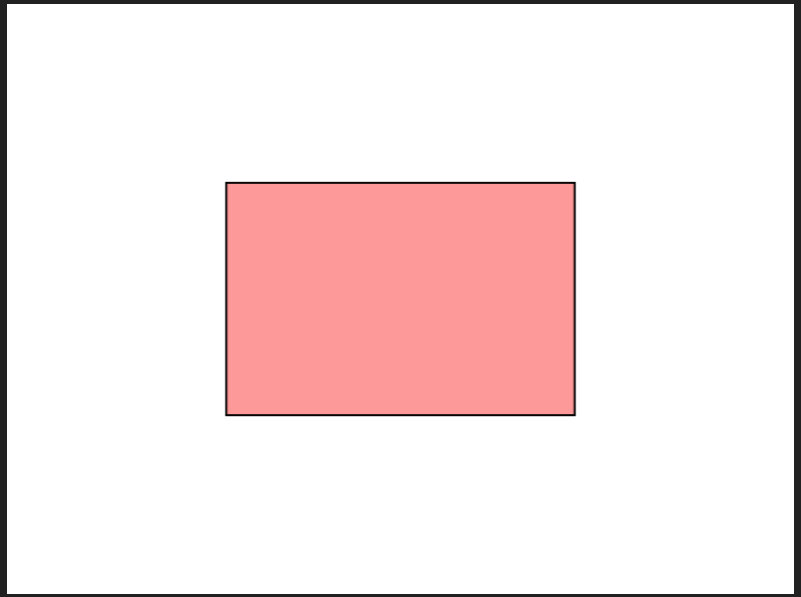
- Использование MPI
- Использование C++ в качестве средства повышения уровня

Разработан прототип библиотеки

- Grid - распределенный массив
  - Декомпозиция средствами MPI (Decomposition_type)
  - Асинхронный обмен границами (Waiter)
  - Синхронный обмен границами
- Range - “прямоугольное” подмножество индексов
  - for_each - применение лямбда-функции к элементам подмножества
- Reduce - редукция по подмножеству индексов
  - Применение лямбда-функции и MPI_Allreduce для сбора результата

Grid

В терминах библиотеки Grid<int dimention, typename T> - это массив, определенным образом распределенный между процессами.



```
DecompositionType dec_type(MPI_COMM_WORLD, { true,false }, { 0,0 });
Grid<2, double> v1({ N,M }, dec_type);
```

При создании массива пользователь сам выбирает каким образом массив будет разделен между процессами. На картинке показано что будет в случае с одномерной декомпозицией по оси X.

Обмен границами, Waiter

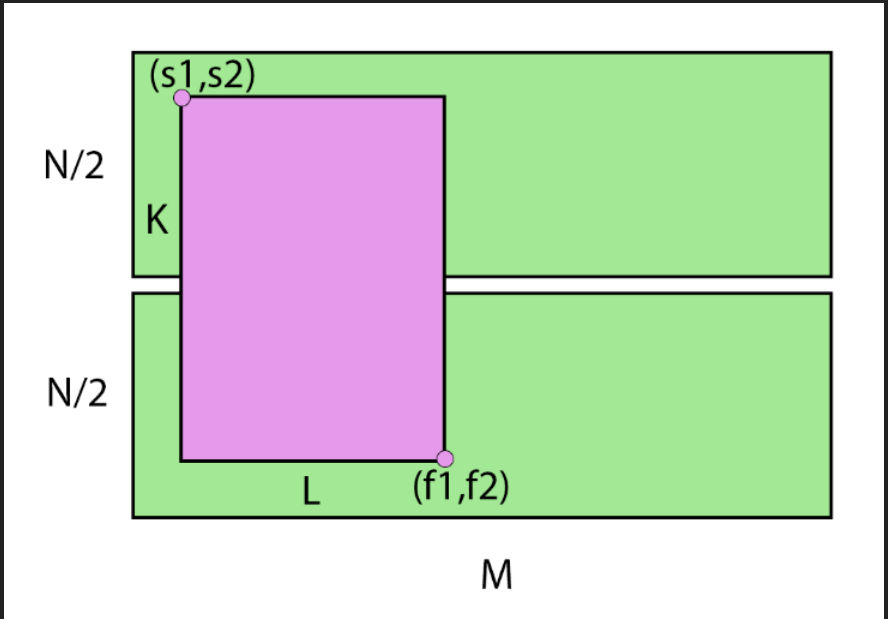
У объекта класса Grid есть методы обмена границами. При синхронном обмене используется MPI_Send и MPI_Recv и такому обмену соответствует метод void send_grid_bound(). При асинхронном обмене используются MPI_Isend и MPI_Irecv и такому обмену соответвует метод Waiter send_grid_bound_async().

```
do {
u1.send_grid_bound();

//some work

} while (...)
```

```
do {
Waiter w = u1.send_grid_bound_async();
// some work in centre
w.wait();
//some work in bounds
} while (...)
```

Range

Объекты класса Range<int dimention> служат для работы с распределенными массивами Grid

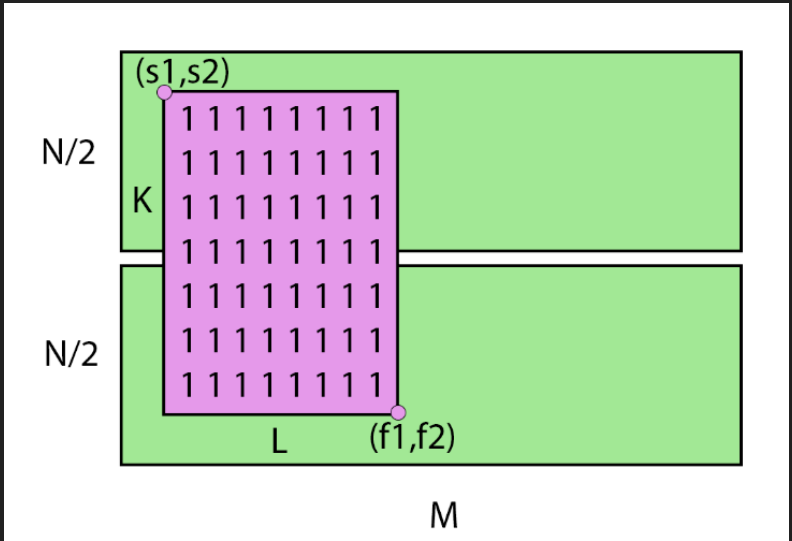



```
Range<2> range({ s1,s2 }, { f1, f2 });
```

Создавая объект класса Range необходимо указать координаты начальной и конечной точек. На картинке K = f1 - s1, L = f2 - s2. Range может быть произвольных размеров и не зависит от массивов, которые нужно будет обходить

Range.for_each(...)

У Range есть метод for_each, который служит для обхода распределенных массивов. Он принимает в качестве аргументов “опорный” распределенный массив и лямбда-функцию.




```
Range<2> range({ s1,s2 }, { f1, f2 });
Grid<2,double> &v1 = u1;
range.for_each(&u1, [&u1](const int indexes[2]) {
        u1(indexes) = 1;
    });
```

Reduce

Класс Reduce отвечает за реализацию операций редукции. Метод reduce данного класса принимает в качестве аргументов Range, “опорный” массив и лямбда-функцию, а также MPI коммуникатор.

```
std::function<double(const int[2])> f = [&u1](const int indexes[2]) -> double {
        return u1(indexes);
    };

double sum = Reduce::reduce<2, double, Operation::SUM>(range, &u1, f, MPI_COMM_WORLD);
// sum = L * K;
```

Тестирование

С помощью прототипа библиотеки была реализован метод Якоби на 2D сетке с 1D декомпозицией и синхронным обменом границами. Результат производительности был сравнен с реализацией этой задачи с использованием “чистого” MPI с асинхронным обменом границами.

Число процессов	Чистый MPI, с	Библиотека, с
1	5.875067	19.717063
2	2.932830	10.390092
4	1.823691	5.773280
8	1.482392	5.058812

Процессор: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (6 ядер, 12 потоков)

Результаты

- Разработан прототип библиотеки
- Реализован метод Якоби на 2D сетке с 1D декомпозицией с помощью библиотеки и “чистого” MPI.
- Выполнено сравнение производительности двух реализаций задачи

Дальнейшие планы

- Требуется выявить причины более низкой производительности, в сравнении с чистой MPI программой, и попытаться их устранить.
- Необходимо довести интерфейс библиотеки до более удобного в использовании и понятного в понимании.
- Реализация запланированной функциональности библиотеки (различное число измерений массива, различные способы декомпозиции и распределения ресурсов и т.д.)
- Расширение библиотеки поддержкой других параметров реализации (ширина теневых граней, способ обмена границами и т.д.)
- Реализации 3D задачи и сравнение производительности с чистой MPI реализацией