Распределенные системы. Отчёт по практическому заданию 2

Андрей Лебедев, группа 424

1 Постановка задачи

1.1 Описание задачи релаксации

В задаче рассматривается двумерная матрица $A \in \mathbb{R}^{N \times N}$, где значения на границах фиксированы (или равны нулю), а внутренние узлы подвергаются итерационной релаксации по формуле:

$$A_{i,j} \leftarrow \frac{A_{i-1,j} + A_{i+1,j} + A_{i,j-1} + A_{i,j+1}}{4}.$$

Сходящийся процесс останавливается при достижении заданного порога точности eps $<$ maxeps либо при превышении числа итераций it_{max}.

1.2 Требования к параллельной версии с контрольными точками

- Параллельная реализация на основе MPI.
- Периодическое сохранение состояния в файл. Сохраняются:
  - номера текущей итерации;
  - локальные фрагменты массива A на каждом процессе.
- В случае сбоя возможно продолжение вычислений.
- В завершение требуется оценить временные затраты на запись и чтение чекпоинтов, а также общее время решения.

2 Описание алгоритма и реализации

2.1 Основные этапы

Алгоритм состоит из следующих главных шагов:

1. Инициализация
   - Если существует файл контрольной точки, загружаем:
     - глобальный индекс итерации;
     - соответствующий локальный фрагмент матрицы A;

- Если файл не существует, инициализируем матрицу вручную.

2. Основной цикл итераций (relax)
   - Вычисление новых значений $A_{i,j}$ с помощью усреднения соседей.
   - Подсчёт локального изменения (ошибки) и дальнейший `MPI_Allreduce` для вычисления глобальной `eps`.
   - Обмен граничными строками/столбцами между соседними процессами.
   - Если `eps` < `maxeps`, завершить.
3. Периодическое сохранение состояния
   - Каждые k итераций, все процессы коллективно открывают файл (MPI-IO) и записывают:
     - номер итерации (процесс 0);
     - локальные фрагменты матрицы, все процессы записывают *в свою* позицию в файле.
4. Имитация сбоя
   - На заранее заданной итерации один из процессов вызывает `raise(SIGKILL)` или `exit(1)`.

2.2 Детали реализации

Ключевые моменты реализации, которые важны для корректной работы:

- Блочное разбиение по строкам: процесс с рангом p получает строки $[p \cdot (N/\text{size}), (p+1) \cdot (N/\text{size}))$.
- MPI-IO
  - Использование функций `MPI_File_open`, `MPI_File_write_at`, `MPI_File_read_at`.
  - Вычисление смещения для каждого процесса, чтобы записывать/считывать соответствующий фрагмент матрицы в одном общем файле.
- Обработчик ошибок ULFM
  - `MPI_Comm_create_errhandler` и `MPI_Comm_set_errhandler`.
  - При сбое вызывается `MPIX_Comm_revoke` и `MPIX_Comm_shrink`, чтобы создать новый коммуникатор без «упавшего» процесса.
  - Перечитывается чекпоинт, продолжается счёт (итерация откатывается на несколько шагов).

3 Временные оценки работы алгоритма

3.1 Время записи чекпоинта

Пусть размер матрицы $N \times N$. Тогда объём данных:

$$\text{Volume} = N \times N \times \text{sizeof(double)}.$$

При записи в файл с помощью MPI-IO в худшем случае мы имеем примерно $O(N^2)$ байт.

3.2 Время вычислений без сбоя

Обозначим t_{iter} время одной итерации без учёта IO. Тогда общее время при it_{max} итерациях можно приблизительно оценить как

$$T_{\text{compute}} = it_{\max} \times t_{\text{iter}}.$$

При параллельном запуске на p процессах:

$$t_{\text{iter}} \approx \frac{N^2}{p \cdot R_{\text{CPU}}} \quad (\text{где } R_{\text{CPU}} — \text{ скорость выполнения }).$$

3.3 Время восстановления после сбоя

- `MPIX_Comm_shrink` (создание нового коммуникатора без упавшего процесса), время зависит от числа оставшихся процессов.
- Повторная инициализация массивов из чекпоинта на всех процессах, примерно $O(N^2/p)$ для чтения.
- Продолжение итераций.

4 Заключение

В ходе работы выполнено:

- Распараллеливание задачи релаксации на $N \times N$ сетке средствами MPI.
- Добавлен механизм чекпоинтов с использованием параллельного ввода-вывода.
- Реализована логика восстановления из сохранённого файла.