Отказоустойчивая реализация параллельного решения двумерного уравнения теплопроводности

Марков Владислав Алексеевич
магистрант 1 курса

Научный руководитель –
д.т.н. доцент Курносов М.Г.

Архитектурные свойства высокопроизводительных вычислительных систем

- Современные высокопроизводительные вычислительные системы – являются ***большемасштабными***
- Количество процессорных ядер в таких системах ***порядка миллиона***
- В перспективных вычислительных системах ожидается ***увеличение числа вычислительных узлов на несколько порядков***
- Среднее время наработки на отказ будет составлять ***порядка 30 минут***
- Актуальной является разработка ***программных средств, обеспечивающих отказоустойчивое выполнение*** параллельных программ

№	Местонахождение	Система	Кол-во процессорных ядер
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	***10,649,600***
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	***3,120,000***
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan - Cray XK7**- Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	***560,640***

https://www.top500.org/

Методы восстановления выполнения параллельных программ

Методы прямого восстановления (*forward recovery*) [1]

- *возвращают параллельную программу в безошибочное состояние на **основании текущих данных**, по результатам анализа отказа, без обращения к предыдущим состояниям параллельной программы*

Методы обратного восстановления (*backward recovery*) [2]

- *возвращают параллельную программу в безошибочное состояние из **контрольной точки**, основаны на использовании информации о полном или частичном состоянии параллельной программы*

[1] Elnozahy E.N., Alvisi L., Wang Y.,Johnson D.B. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Computing Surveys. 2002. Vol.34, No. 3 P. 375– 408.

[2] Koren I., Krishna C.M. Fault-Tolerant Systems. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2007. 378 p.

Методы, позволяющие сократить время записи контрольной точки

1. **Методы, основанные на механизмах создания журналов передачи сообщений** (*log-based rollback-recovery*), *на стороне посылающего сохраняется содержание сообщения*
2. **Методы, основанные на сохранении изменений между двумя последовательными контрольными точками**, *инкрементный* (*incremental checkpointing*) *и дифференциальный* (*differential checkpointing) подходы* [3]
3. **Методы, предполагающие сохранение контрольной точки в память вычислительных узлов** [4]

[3] Ferreira K.B., Riesen R., Bridges P.G., Arnold D., Brightwell R. Accelerating incremental checkpointing for extreme-scale computing. Future Generation Computer Systems. 2014. Vol. 30, No 1. P. 66–77

[4] Plank J.S., Li K., Puening M.A. Diskless Checkpointing. IEEE Transanctions on Parallel Distributed Systems. 1998. Vol. 9, No 10. P. 972–986.

Восстановление выполнения параллельной программы на уровне пользователя (user-level checkpointing)

- В работе подробно рассматривается формирование контрольной точки на уровне пользователя
- Использование такого подхода к формированию контрольной точки, позволяет контролировать время и частоту создания контрольной точки во время выполнения параллельной программы

Параллельная программа решения двумерного уравнения теплопроводности

- Решение двумерного уравнения теплопроводности, реализовано методом *последовательных итераций Якоби*
- Метод распараллелен в стандарте MPI путем *двумерной декомпозиции расчетной области*
- Каждому процессу назначена *подматрица и теневые ячейки* по всем четырем направлениям, для расчета значений на границах выделенной области
- Для обеспечения восстановления вычислительного процесса в программу добавлен функционал *формирования контрольных точек*

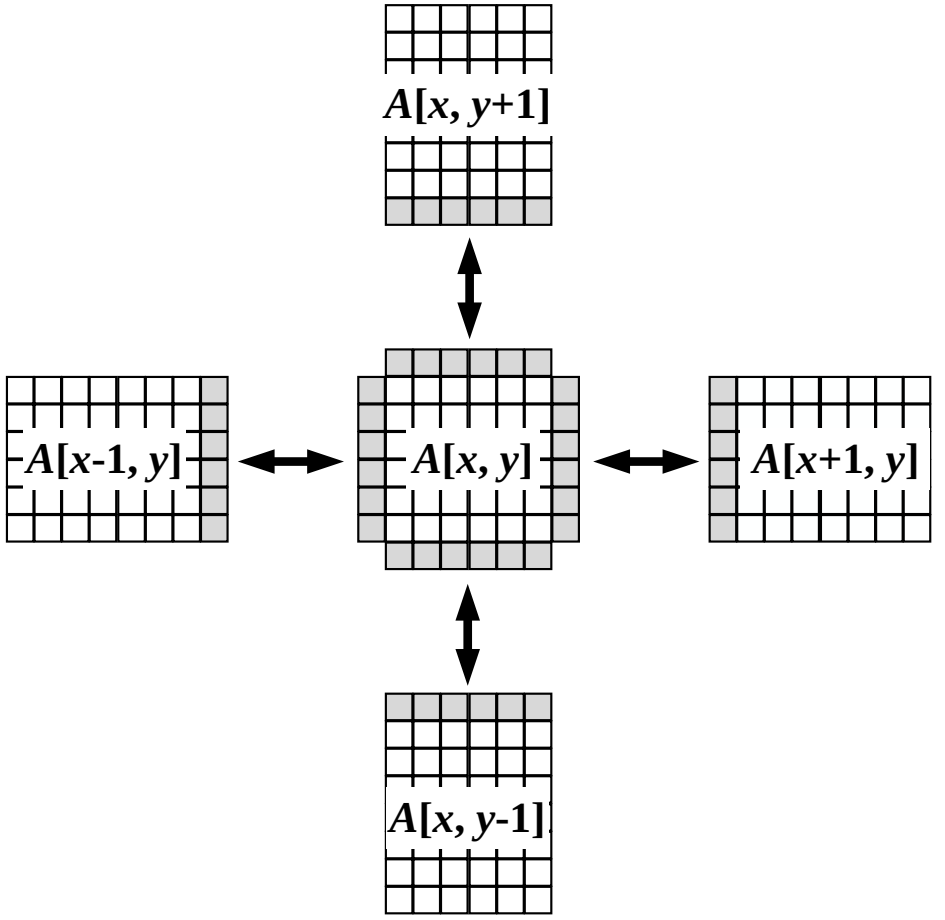


Схема обмена теневыми ячейками подматрицы $A[x, y]$

Параллельная программа решения двумерного уравнения теплопроводности

```
while true do
    for 1 to ny
        for 1 to nx
            // Расчет значений внутренних точек
        end for
    end for

    for 1 to ny
        for 1 to nx
            maxdiff = fmax(maxdiff,...)
        end for
    end for

    if maxdiff < EPS then
        // Проверка условия на выход
        break
    end if

    // Обмен теневыми ячейками с соседями
    // Создание контрольной точки
end while
```

Главный цикл вычисления значений в подматрице $A[x, y]$ и обмена теневыми ячейками

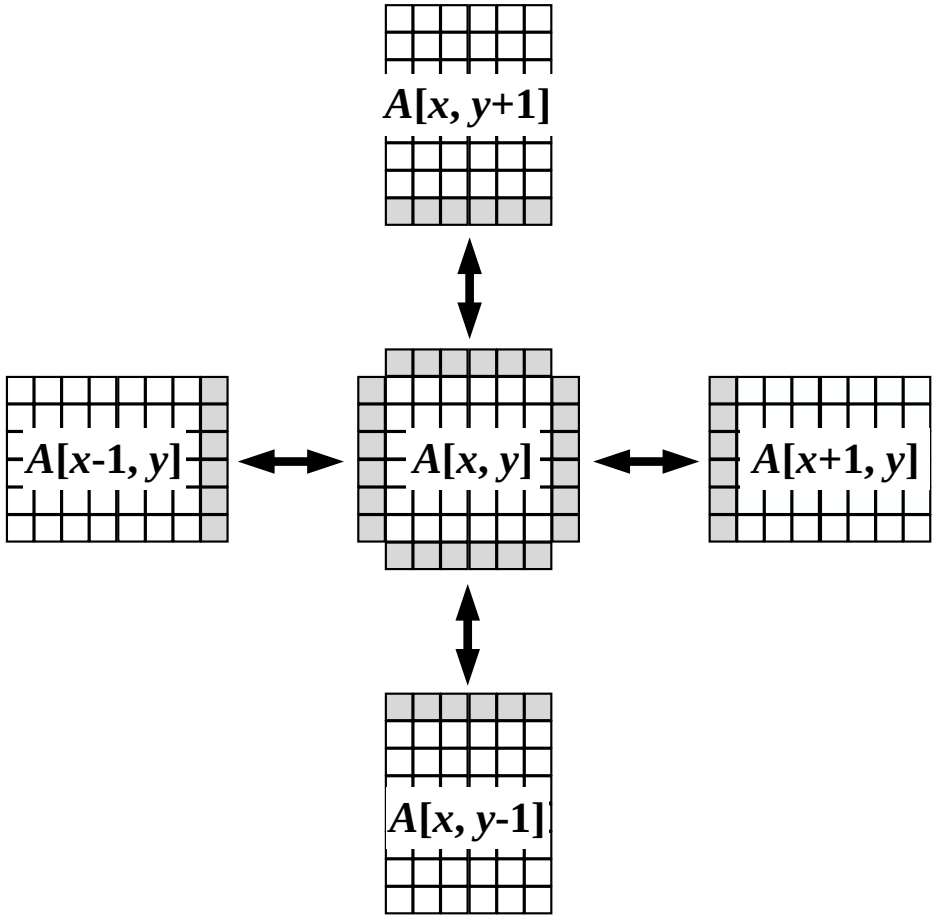


Схема обмена теневыми ячейками подматрицы $A[x, y]$

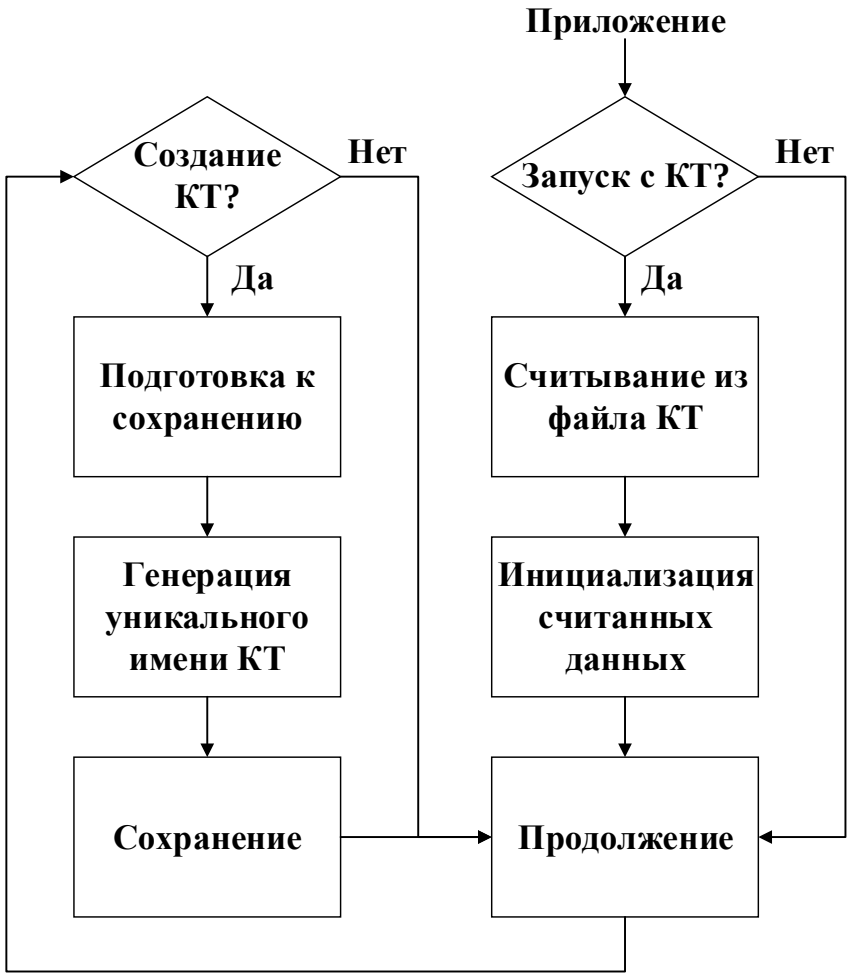
Формирование контрольных точек

Формирование и создание контрольных точек (КТ) реализовано во внешней библиотеке.

КТ формируется каждым MPI-процессом и содержит:

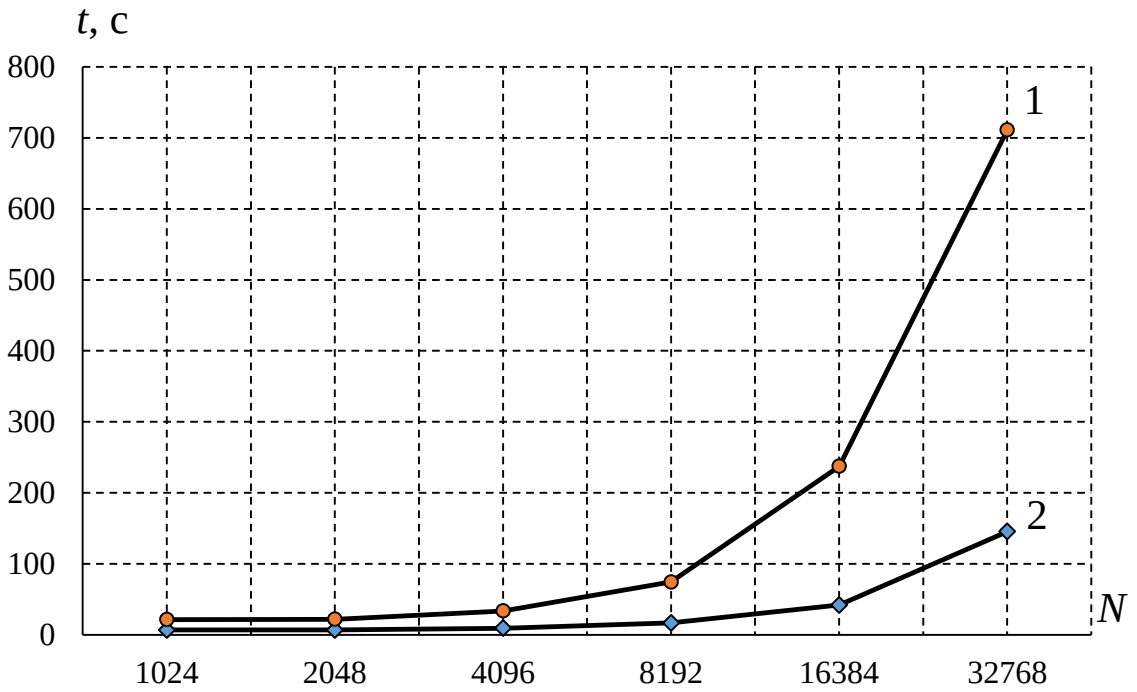
- ❑ Подматрицу MPI-процесса
- ❑ Метаданные КТ*
  - *время выполнения*
  - *количество созданных КТ,*
  - *время затраченное на создание КТ,*
  - *признак целостности КТ*

**формируется внешней библиотекой автоматически*



Алгоритм создания КТ и возобновления вычислительного процесса

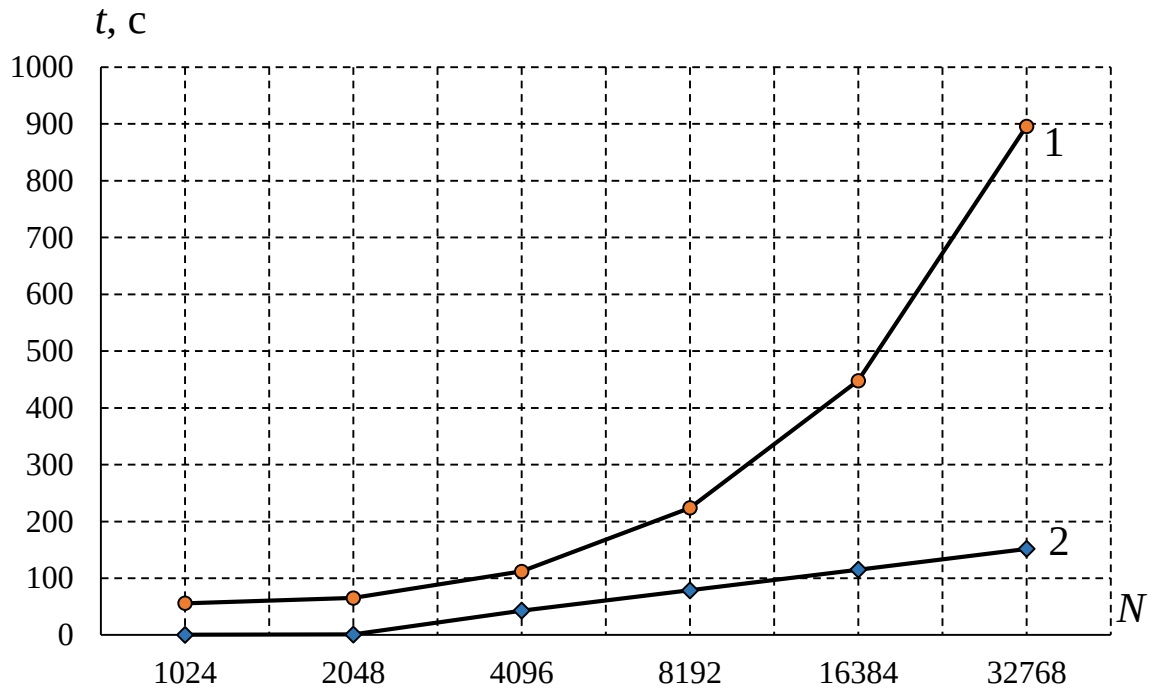
Эксперименты (запись контрольной точки по протоколу NFS)



Зависимость времени *t* выполнения параллельной программы от размера *N* расчетной области на кластере Oak (32 процесса – 4 вычислительных узла)

1 – время выполнения параллельной программы с формированием контрольной точки

2 – время выполнения исходной параллельной программы

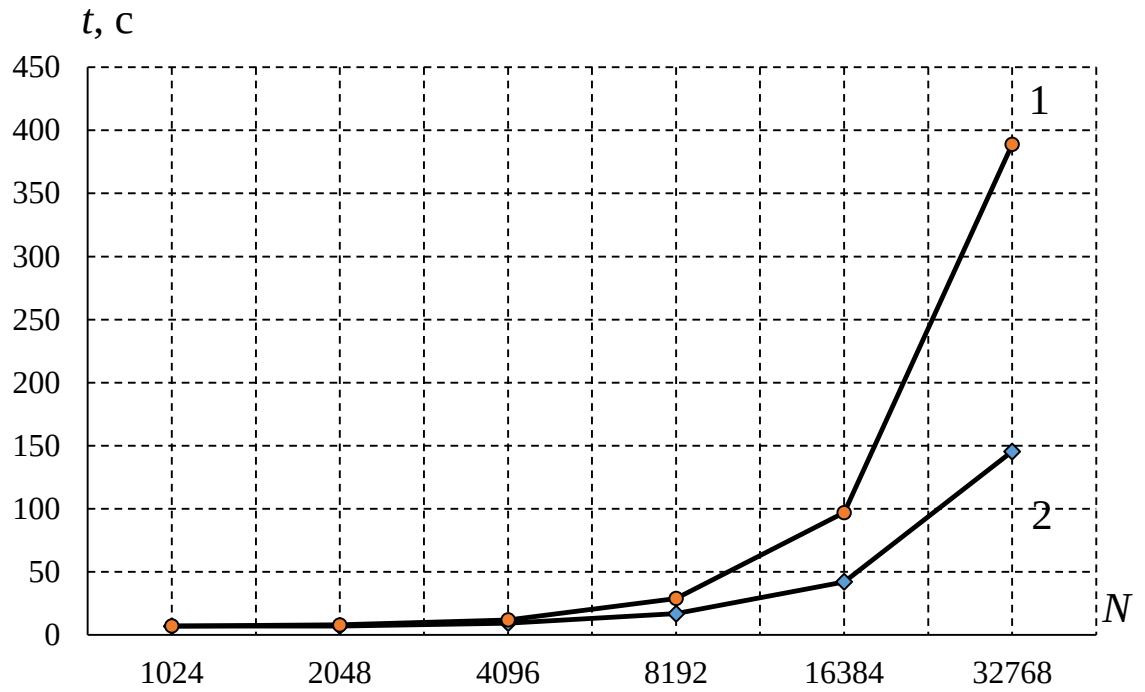


Зависимость времени *t* выполнения параллельной программы от размера *N* расчетной области на кластере Jet (128 процессов – 16 вычислительных узлов)

1 – время выполнения параллельной программы с формированием контрольной точки

2 – время выполнения исходной параллельной программы

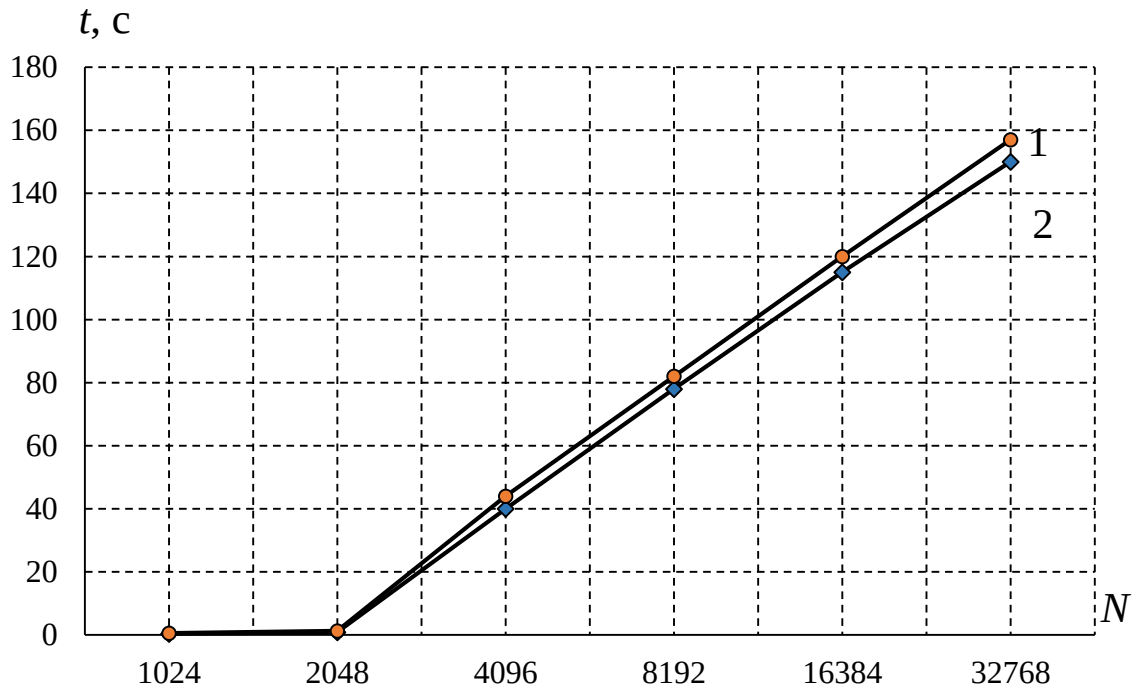
Эксперименты (запись контрольной точки в каталог /tmp/)



Зависимость времени *t* выполнения параллельной программы от размера *N* расчетной области на кластере Oak (32 процесса – 4 вычислительных узла)

1 – время выполнения параллельной программы с формированием контрольной точки

2 – время выполнения исходной параллельной программы



Зависимость времени *t* выполнения параллельной программы от размера *N* расчетной области на кластере Jet (128 процессов – 16 вычислительных узлов)

1 – время выполнения параллельной программы с формированием контрольной точки

2 – время выполнения исходной параллельной программы

Заключение

- Накладные расходы на формирование контрольных точек зависят от размера входных данных
- Направление дальнейшей работы – разработка алгоритма, оптимизирующего накладные расходы на формирование и сохранение контрольных точек

Спасибо за внимание!

Исходный код тестовой программы

```
int main(int argc, char *argv[])
{
        /* Block Initialize */
        /* Save checkpoint function */
        for (;;) {
                /* Update interior points */
                for (int i = 1; i <= ny; i++) {
                        for (int j = 1; j <= nx; j++) {
                                local_newgrid[IND(i, j)] =
                                        (local_grid[IND(i - 1, j)] +
                                        local_grid[IND(i + 1, j)] +
                                        local_grid[IND(i, j - 1)] +
                                        local_grid[IND(i, j + 1)]) * 0.25;
                        }
                }
                /* Check termination condition */
                for (int i = 1; i <= ny; i++) {
                        for (int j = 1; j <= nx; j++) {
                                maxdiff = fmax(args);
                        }
                }
                /* Exchange shadow points with neughbors */
                MPI_Irecv(args);
                MPI_Isend(arg);
                /* Save checkpoint function */
                /* In simulation case:
                 * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
                 */
        }
        /* Save checkpoint function */
        /* Block show result */
}
```