###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

Игра "Жизнь"

студента 2 курса, группы 22204

Соломенникова Николая Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент

А.Ю.Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 2](#__RefHeading___1)

[ЗАДАНИЕ 2](#__RefHeading___2)

[ОПИСАНИЕ РАБОТЫ 3](#__RefHeading___3)

[ЗАКЛЮЧЕНИЕ 7](#__RefHeading___4)

[ИСПОЛЬЗУЕМЫЕ КОМАНДЫ 8](#__RefHeading___5)

[ЛИСТИНГИ 8](#__RefHeading___6)

# ЦЕЛЬ

Практическое освоение методов реализации алгоритмов мелкозернистого параллелизма на крупноблочном параллельном вычислительном устройстве на примере реализации клеточного автомата «Игра "Жизнь" Дж. Конвея» с использованием неблокирующих коммуникаций библиотеки MPI.

# ЗАДАНИЕ

1. Написать параллельную программу на языке C/C++ с использованием MPI, реализующую клеточный автомат игры "Жизнь" с завершением программы по повтору состояния клеточного массива в случае одномерной декомпозиции массива по строкам и с циклическими границами массива. Проверить корректность исполнения алгоритма на различном числе процессорных ядер и различных размерах клеточного массива, сравнив с результатами, полученными для исходных данных вручную.
2. Измерить время работы программы при использовании различного числа процессорных ядер: 1, 2, 4, 8, 16, … . Размеры клеточного массива X и Y подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Построить графики зависимости времени работы, ускорения и эффективности распараллеливания от числа используемых ядер.
3. Произвести профилирование программы и выполнить ее оптимизацию. Попытаться достичь 50-процентной эффективности параллельной реализации на 16 ядрах для выбранных X и Y

# ОПИСАНИЕ РАБОТЫ

В ходе профилирования выявилось, что просчёт истории полей (всех ранее сгенерированных полей) занимает очень много времени и сильно влияет на производительность, поэтому было принято решение оптимизировать соответствующую часть программы. Для этого был переделан способ хранения полей в истории. Вместо того, чтобы хранить каждое поле как массив char-ов (1 – клетка жива, 0 – клетка мертва), поля упаковывались в массив char-ов, где каждый char содержит информацию сразу о восьми клетках. В случае параллельной программы эта оптимизация особенно полезна, так как процессам нужно обмениваться в 8 раз меньшим количеством данных.

Приведём пример того, на сколько данная оптимизация улучшает производительность. Измерения проводились при размере поля 50\*51 и 16 процессах (для параллельных программ).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| программа | sequential | parallel | sequential  optimized | parallel optimized |
| время | 41,8 | 12,8 | 7,6 | 2,2 |

Можно заметить, что последовательная программа с использованием оптимизации ускорилась в 5,5 раза, а параллельная примерно в 5,8.

Приведём результаты работы оптимизированной параллельной программы. В качестве тестового поля бралось пустое поле размером N\*(N+1) с один глайдером в углу. Это обеспечивает долгую продолжительность программы, а именно 4\*N\*(N+1) поколений. Сложность алгоритма будет O(N^6), так как размер поля примерно N^2, итераций примерно N^2 и на каждой итерации надо проверять не повторилось ли поле, то есть надо сверять примерно N^2 полей. И того выходит O(N^6).

Результаты измерений при N=80 (поле 80\*81, количество поколений 25920):

Время выполнения последовательной программы: T0 = 100,9 сек.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| количество процессов - p | 2 | 4 | 8 | 16 |
| время - Tp  (в секундах) | 70,5 | 43,5 | 27,3 | 26,9 |

График времени работы параллельной программы в зависимости от количества процессов (время в секундах):

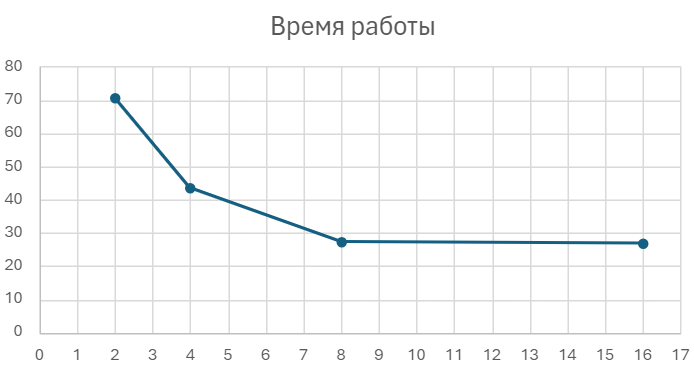
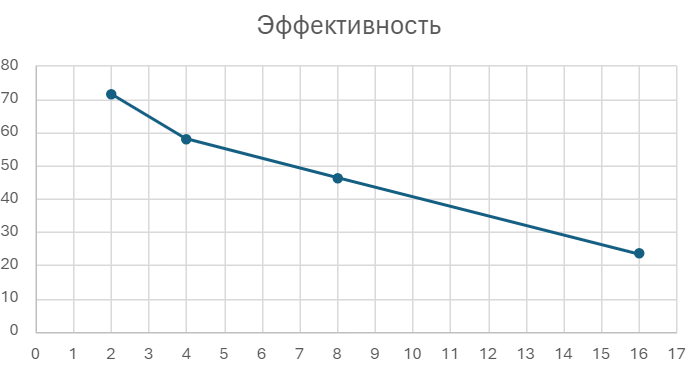


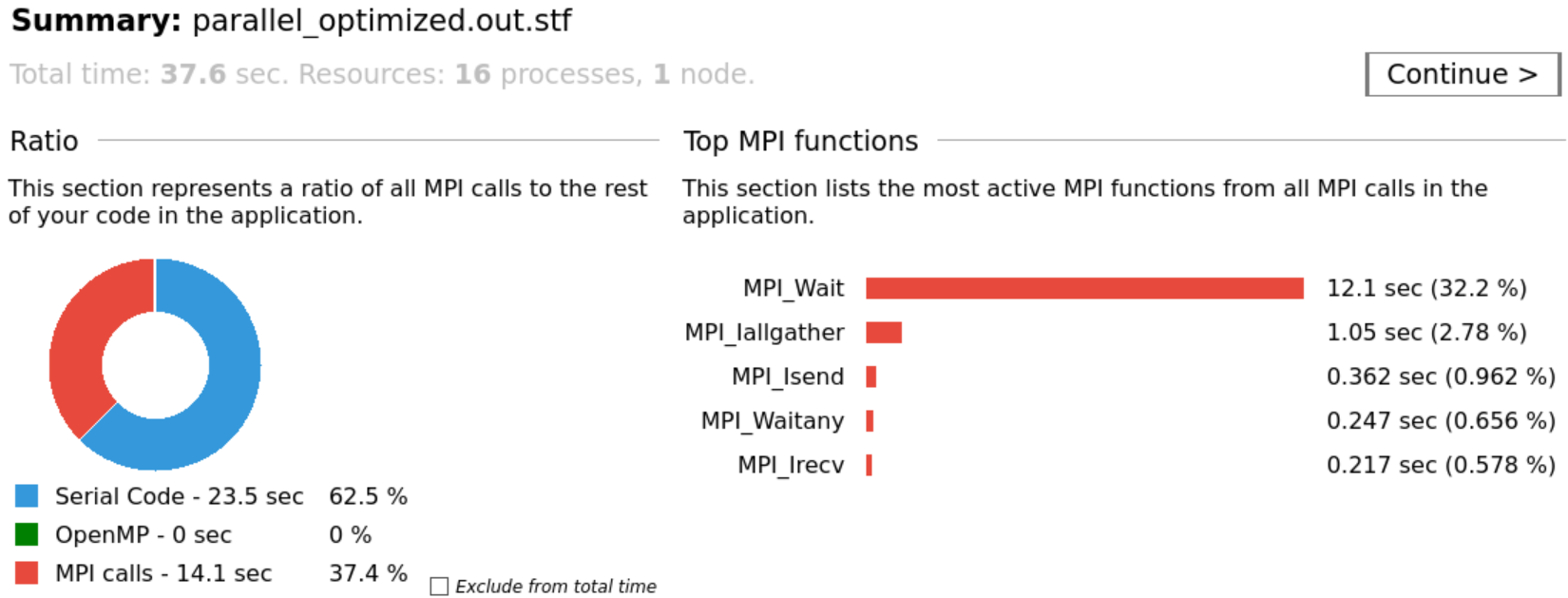
График ускорения параллельной программы относительно последовательной программы:



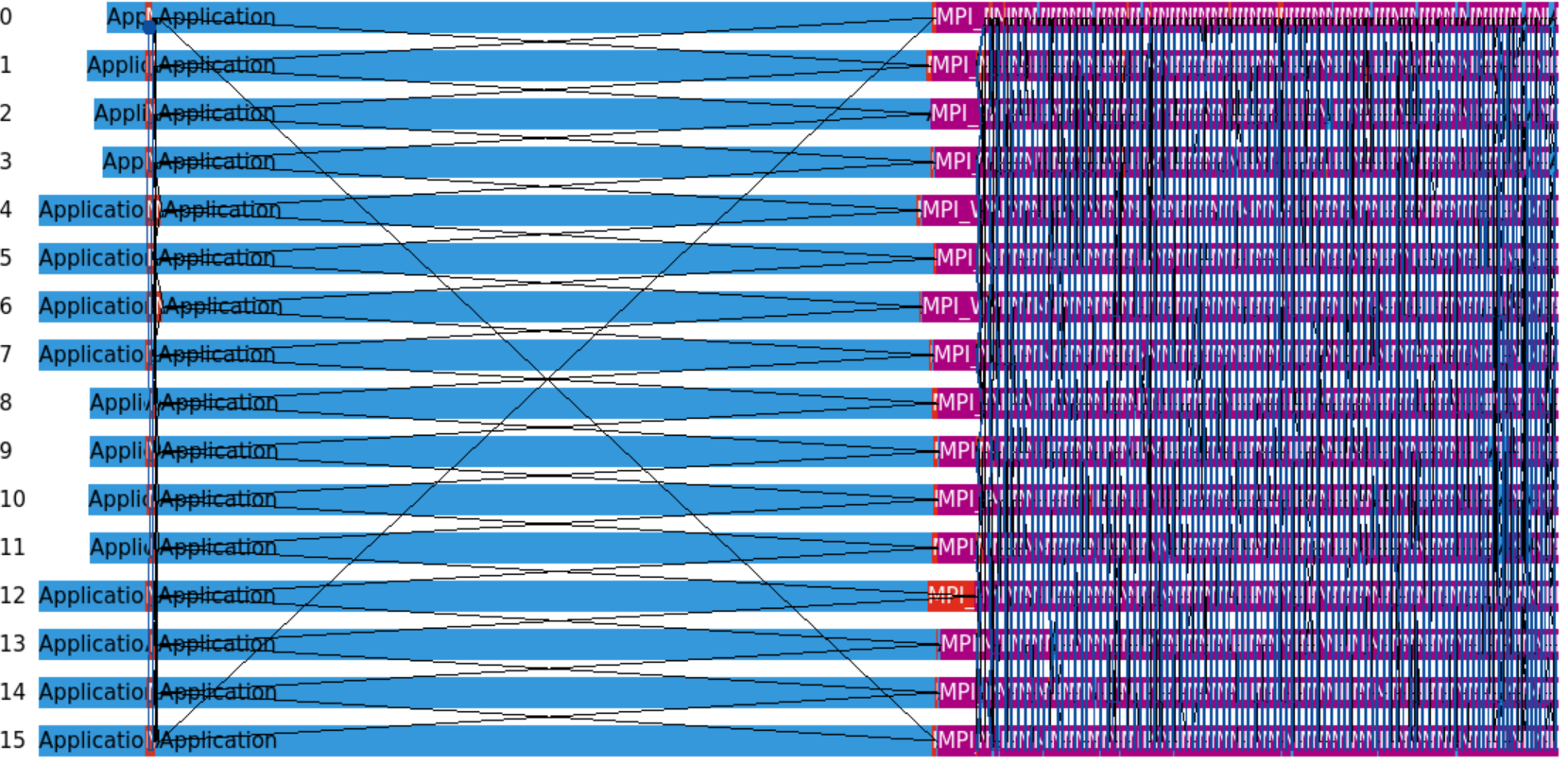
График эффективности параллельной программы (в процентах):



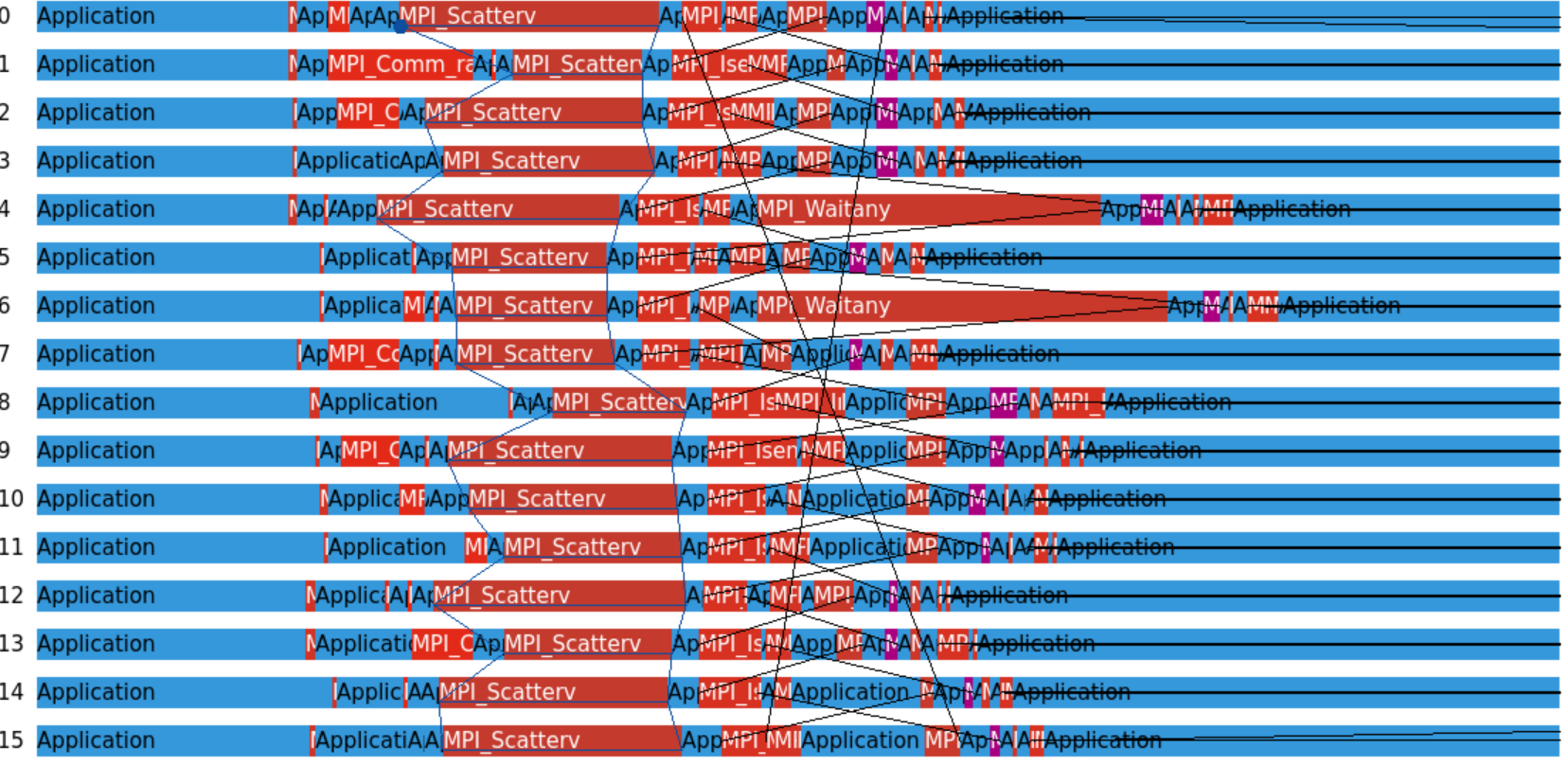
Профилирование:



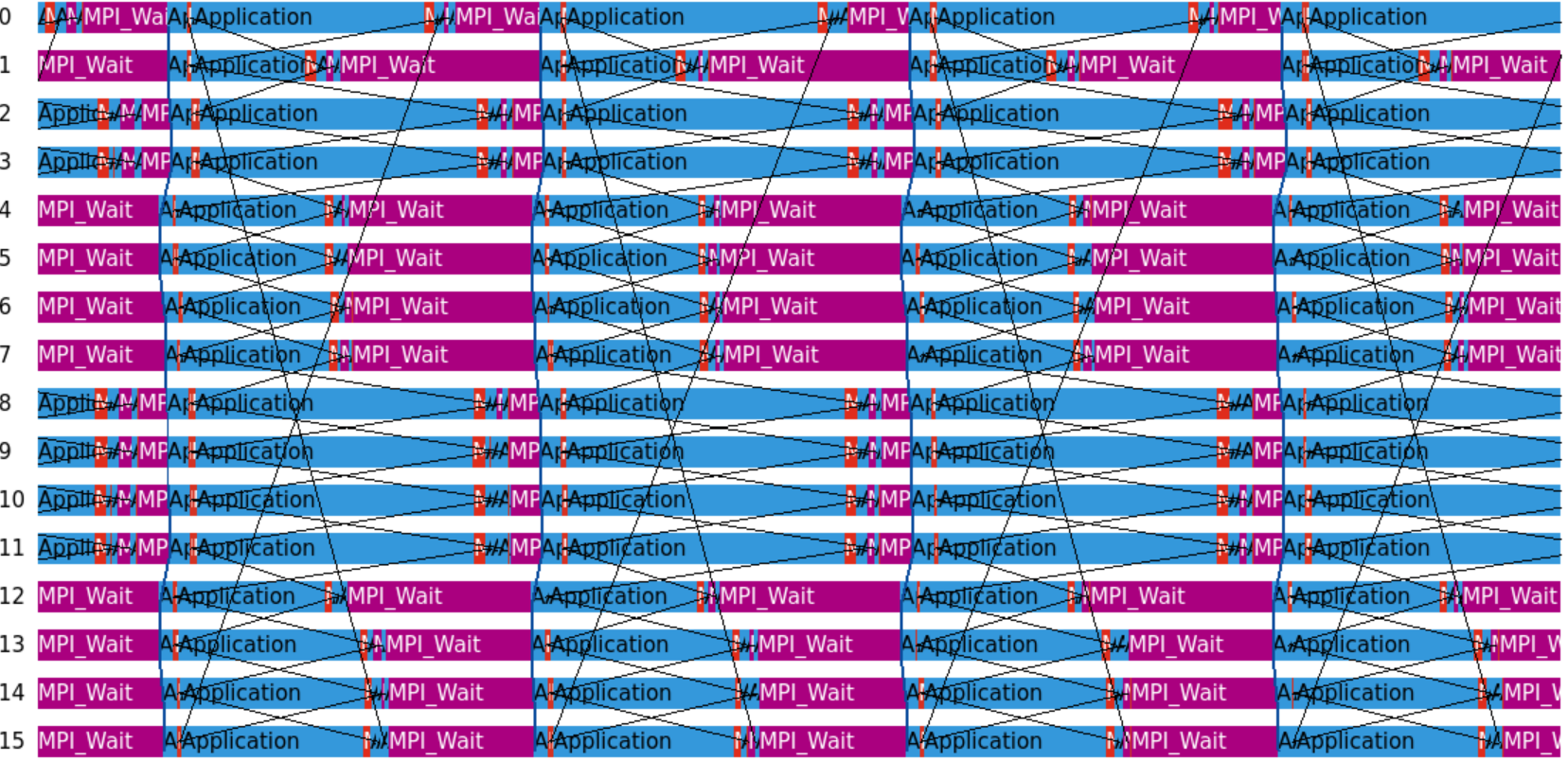
Начало программы:



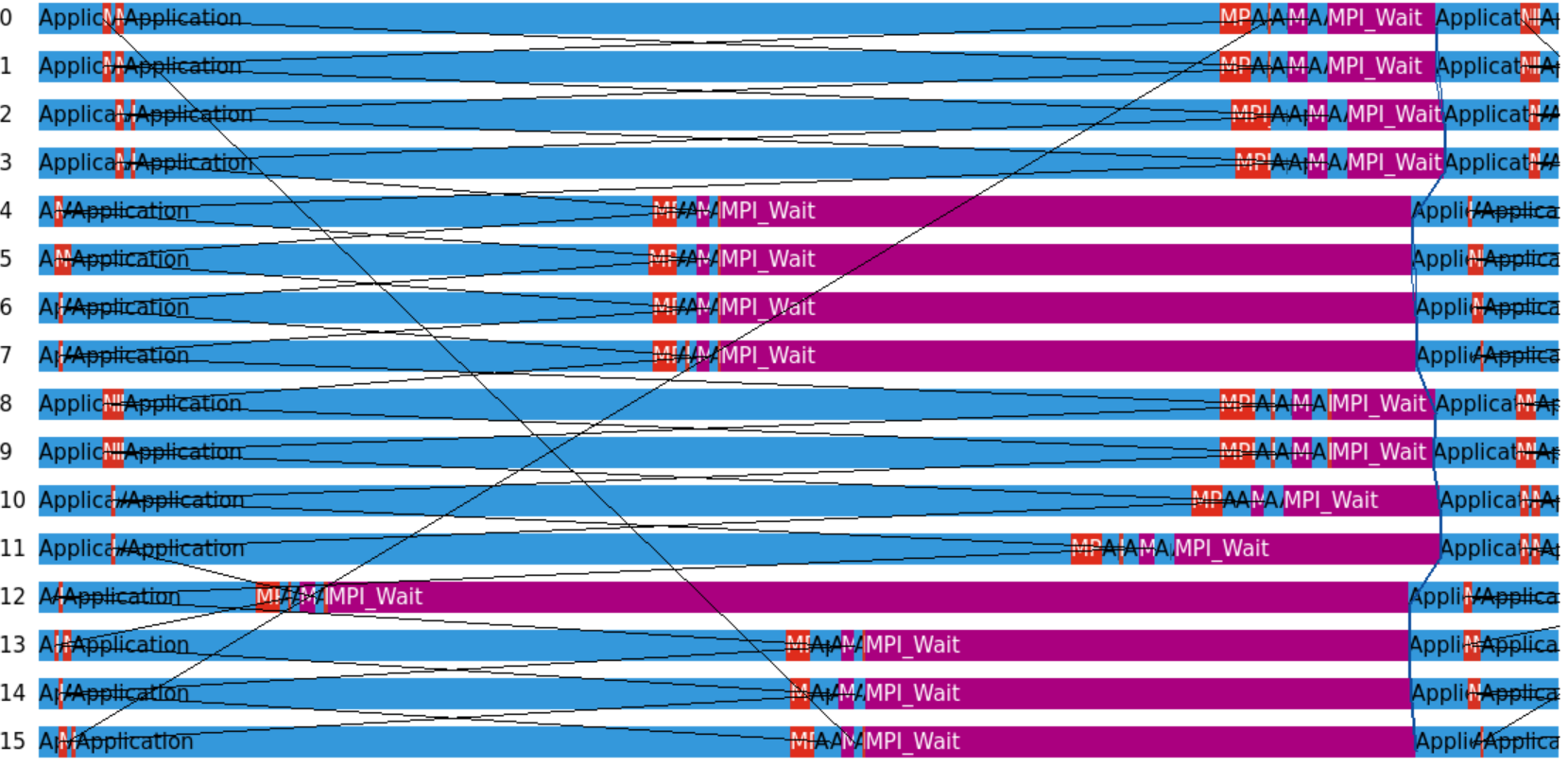
Приближенное начало программы:



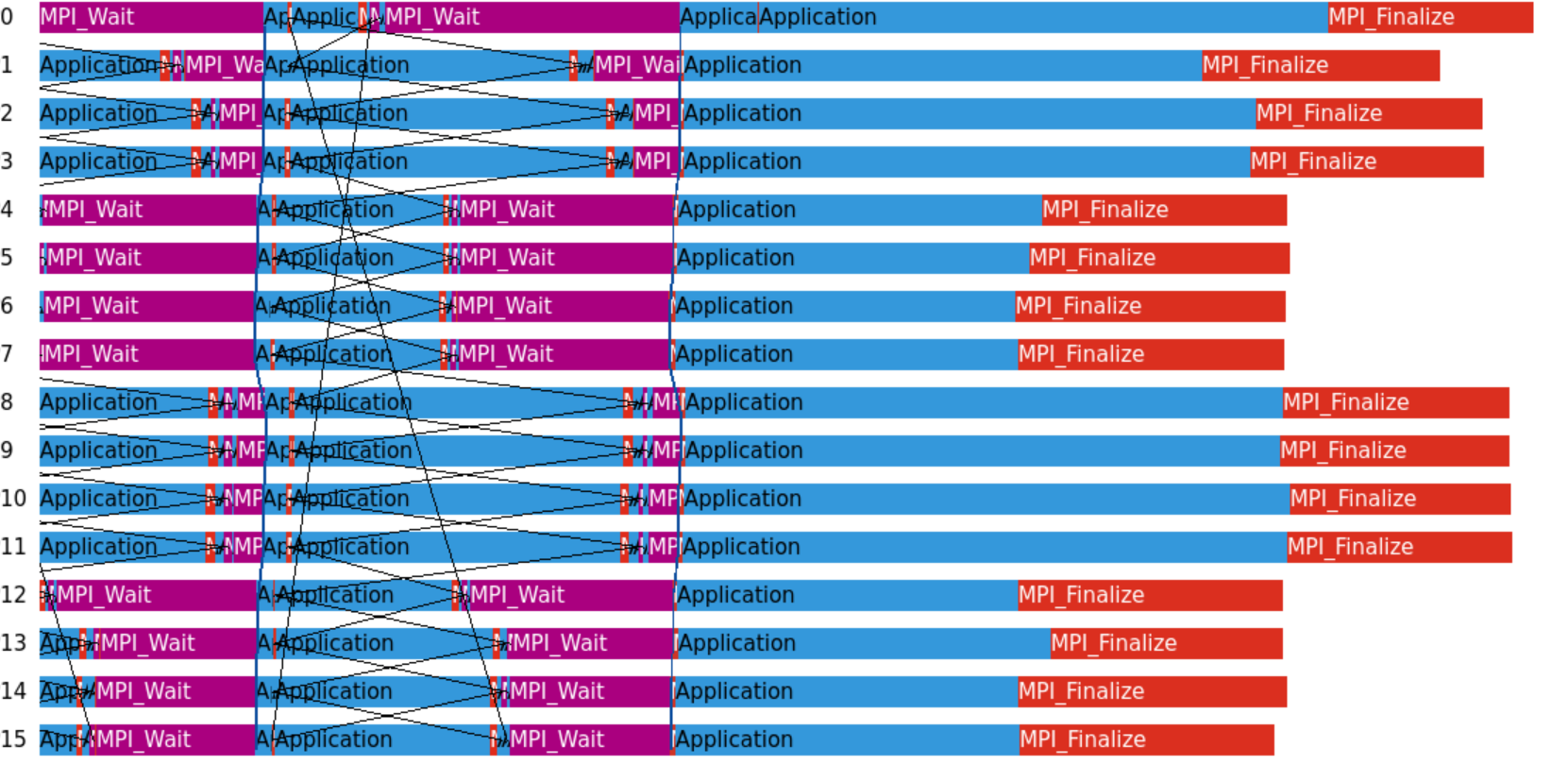
Середина программы:



Приближенная середина программы:



Конец программы:



# ЗАКЛЮЧЕНИЕ

В начале программы почему-то очень долго выделяется память под буферы для флагов (такого быть не должно). В середине программы бОльшую часть времени занимает нахождение совпадений полей в истории. Причём, так как разные процессы считают разные части поля, где-то совпадение может найтись раньше, а где-то позже (это видно на графиках). Отсюда следует, что MPI\_Wait (фиолетовые участки на графиках) занимает много времени не потому, что он долго работает сам по себе (но и это тоже), а потому, что он ждёт массивы флагов со ВСЕХ процессов (это нормально, так и должно быть).

# ИСПОЛЬЗУЕМЫЕ КОМАНДЫ

$mpicxx -O2 prog.cpp -o prog.out – компиляция

$mpicxx -Wall -g prog.cpp -o prog.out – копиляция для отладки

$valgrind --leak-check=yes --show-leak-kinds=all ./prog.out – поиск утечек памяти

$module load mpi/intelmpi – загрузка mpi модуля

$module load itac – загрузка модуля для профилирования

$sbatch ./launch.sbatch – поставить задачу в очередь

$squeue - посмтреть очередь

$scancel <идентификатор\_задачи> – снять задачу с выполнения

$traceanalyzer prog.out.stf – запустить профилирование

# ЛИСТИНГИ

Ссылка на код:

https://github.com/SolomennikovKolya/NSU\_OPP/tree/ab411bad3c82485c9ae1066496f0986711f6e4b8/task5/src

launch.sbatch:  
#!/bin/bash

#SBATCH -J solomennikov

#SBATCH -p compclass

#SBATCH -o reports/job%j.out

#SBATCH -e reports/job%j.err

#SBATCH -N 4 # Total number of nodes requested

#SBATCH -n 16 # Total number of mpi tasks requested

#SBATCH -t 00:01:00

#Run executable files and commands

module load mpi/intelmpi

module load itac

#TCP-ports, opened for MPI-communications

export I\_MPI\_PORT\_RANGE=23000:23100

which mpiexec

mpiexec -trace -ppn 4 -n 16 ./src/prog.out

# -n – количество mpi процессов

# -ppn – количество процессов на узел

# -trace – для трассировки