Документация, содержащая описание функциональных характеристик заявляемого ПО

Функциональные характеристики программного пакета EWT‑128‑ЦАГИ:

* Численное решение системы уравнений Навье-Стокса, замкнутой моделями турбулентности SST и Спаларта – Альмараса;
* Численный метод реализован в рамках конечно-объемного подхода, и обеспечивает получение решения со вторым порядком точности по пространственным переменным и первым — по времени;
* Работа со структурированными расчетными сетками в формате CGNS;
* Расчет напряженно-деформированного состояния модели летательного аппарата на основе распределений давления, полученных при численном решении уравнений Навье-Стокса;
* Автоматическое перестроение математической модели и расчетной сетки летательного аппарата на основе данных о деформации его элементов;
* Совместимость формата выходных данных программного комплекса с информационной системой получения и обработки экспериментальных данных;

Программный комплекс EWT-128-ЦАГИ состоит из компонентов, написанных на языках C++, Python, Matlab, VBA.

Программный комплекс EWT-128-ЦАГИ может быть установлен на операционные системы семейств Windows и GNU Linux.

Программный комплекс EWT-128-ЦАГИ в процессе работы использует следующие дополнительные компоненты: библиотека cgns, библиотека openmpi.

Программный комплекс EWT-128-ЦАГИ состоит из следующих расчётных модулей, разделённых по основному функциональному назначению (в скобках указаны альтернативные имена программ, MPI – обозначает возможность распараллеливания на кластере):

Подготовка расчета

* + z\_import\_cgns — используется для конвертации расчётной сетки из формата CGNS, содержащей координаты узлов, признаки граничных условий и названия семейств граней сетки, в файлы \*.zmesh внутреннего формата и создания соответствующего файла \*.zboco с описанием этой расчётной сетки;
  + z\_extrude\_bl — используется в случае, если построенная сетка не имеет сгущения к поверхностям с прилипанием потока. При запуске программы производится «выдавливание» дополнительных блоков из поверхности со сгущением к ней сетки для правильного описания пограничного слоя (кроме угловых блоков);
  + z\_convert — данная программа конвертирует всю задачу (все файлы задачи), которая ранее считалась по пакету EWT на базе решателей V3Solver и COMGLEI. Этот пакет использует более ранние форматы некоторых основных файлов;
  + z\_make\_connection — программа производит стыковку блоков, и записывает информацию об адресах стыкующихся ячеек. Эта информация необходима для передачи данных через границы блоков во время обмена данными после цикла прогона расчётного алгоритма по каждому блоку;
  + z\_dwall (MPI) — производится расчет расстояния до твёрдых стенок. Информация используется в численных схемах и методах солвера;
  + z\_init\_field (MPI) — программа используется для задания начального поля.
  + z\_cntl\_points — программа заполняет файл \*.zpoints дополнительной информацией, которая привязывает контрольные точки к сетке. Первоначально файл содержит номера точек и их координаты. В процессе счёта солвером для каждой точки создаётся отдельный файл \*.zstat, в котором записывается статистика по итерациям параметров газа для данной точки.

Расчет

* + z\_run (MPI) — основной расчетный модуль

Обработка

* + z\_export — производит вывод графической информации в форматах cgns, dat, fld. Формат fld внутренний. Данные могут быть импортированы в другие программы визуализации, такие как Tecplot, ParaView.
  + z\_post\_proc (MPI)  — производит поэлементную обработку расчетной модели с вычислением аэродинамических характеристик и коэффициентов.

В данном разделе приведена документация к основным модулям, а также пример использования программного комплекса для расчета поля течения тестовой задачи.

# Модуль стыковки блоков z\_make\_connection

## Описание

Модуль производит стыковку блоков структурированной расчётной сетки. Поддерживаются два варианта стыковки: строгий (стыковка ячейка в ячейку, граничное условие joint) и нестрогий (стыковка с помощью интерполяции, граничное условие connect). Информация об адресах состыкованных ячеек записывается в файл \*.zjoint в случае строгой стыковки и в файл \*.zconnect — в случае нестрогой. Оба варианта стыковки могут использоваться как по отдельности, так и совместно в рамках одной расчётной сетки.

## Команда вызова, входные данные

Команда вызова модуля z\_make\_connection выглядит следующим образом (необязательные элементы указаны в квадратных скобках):

z\_make\_connection zboco [parameters]

здесь

zboco — путь к файлу топологии;

parameters — параметры вызова программы, список параметров и их значение приводится в таблице 1.

Таблица 1 — Список и значение параметров вызова модуля z\_make\_connection

|  |  |  |
| --- | --- | --- |
| Короткое написание | Полное написание | Действие |
| -h | --help | вывести справочную информацию |
| -j | --joint | выполнить только строгую стыковку (граничное условие *joint*) |
| -c | --connect | выполнить только нестрогую стыковку (граничное условие *connect*) |
| -m | --modify | изменять заплатки |
| -v | --verbose | режим вывода дополнительной информации |

**Пример**. Команда

z\_make\_connection z.zboco

в соответствии с файлом топологии z.zboco выполнит стыковку блоков (как по граничному условию joint, так и connect) и запишет результаты в файлы z.zjoint и z.zconnect.

# Модуль расчёта расстояния до стенки z\_dwall

## Описание

Модуль производит расчёт расстояния от каждой ячейки расчётной сетки до ближайшей твёрдой стенки. Вычисленные расстояния сохраняются в файлы <*имя\_блока*>.zdwall и используются в процессе расчёта.

## Команда вызова, входные данные

Команда вызова модуля z\_dwall выглядит следующим образом (необязательные элементы указаны в квадратных скобках):

z\_dwall zboco [parameters]

здесь

zboco — путь к файлу топологии;

parameters — параметры вызова программы, список параметров и их значение приводится в таблице 2.

Таблица 2 — Список и значение параметров вызова модуля z\_dwall

|  |  |  |
| --- | --- | --- |
| Короткое написание | Полное написание | Действие |
| -h | --help | вывести справочную информацию |
| -v | --verbose | режим вывода дополнительной информации |

**Пример**. Команда

z\_dwall z.zboco

в соответствии с файлом топологии z.zboco вычислит расстояние от каждой ячейки сетки до ближайшей твёрдой стенки и запишет результаты в файлы <*имя\_блока*>.zdwall.

# Модуль генерации начального поля z\_init\_field

## Описание

Модуль генерирует начальное поле в каждом блоке расчётной сетки в соответствии с названием семейства блока. Параметры начального поля для каждого семейства задаются в проектном файле. На поверхности твёрдой стенки с прилипанием потока «намазывается» равномерный пограничный слой. Это необходимо для плавного и надёжного старта расчёта с начального поля.

## Команда вызова, входные данные

Команда вызова модуля z\_init\_field выглядит следующим образом (необязательные элементы указаны в квадратных скобках):

z\_init\_field zproj [zboco] [parameters]

здесь

zproj — путь к файлу проекта;

zboco — путь к файлу топологии (указывается относительно директории, в которой находится файл проекта);

путь к файлу топологии (zboco) может быть указан в файле проекта и/или в команде вызова (в случае, если задействованы оба варианта, путь берётся из команды вызова);

parameters — параметры вызова программы, список параметров и их значение приводится в таблице 3.

Таблица 3 — Список и значение параметров вызова модуля z\_init\_field

|  |  |  |
| --- | --- | --- |
| Короткое написание | Полное написание | Действие |
| -h | --help | вывести справочную информацию |
| -d | --bl\_delta | задать толщину пограничного слоя (по умолчанию 0.01 м) |
| -v | --verbose | режим вывода дополнительной информации |

**Пример**. Команда

z\_init\_field z.zproj ../z.zboco

сгенерирует начальное поля в соответствии с файлами z.zproj и ../z.zboco.

# Расчётный модуль z\_run

## Описание

Производит итерационный расчёт полей переменных газа в расчётном пространстве, определяемом сеткой. Переменные определяются дискретно в центрах ячеек.

## Команда вызова, входные данные

Команда вызова модуля z\_run выглядит следующим образом (необязательные элементы указаны в квадратных скобках):

z\_run zproj [zboco] [zengine] [parameters]

здесь

zproj — путь к файлу проекта;

zboco — путь к файлу топологии (указывается относительно директории, в которой находится файл проекта);

zengine — путь к файлу двигателя (указывается относительно директории, в которой находится файл проекта);

пути к файлу топологии (zboco) и файлу двигателя (zengine) могут быть указаны в файле проекта и/или в команде вызова (в случае, если задействованы оба варианта, пути берутся из команды вызова);

parameters — параметры вызова программы, список параметров и их значение приводится в таблице 4.

Таблица 4 — Список и значение параметров вызова модуля z\_run

|  |  |  |
| --- | --- | --- |
| Короткое написание | Полное написание | Действие |
| -h | --help | вывести справочную информацию |
| -m | --part\_by\_metis | декомпозиция с помощью METIS |
| -p | --post\_proc | при каждом сохранении полей делать обработку (по правилам z\_post\_proc) |
| -v | --verbose | режим вывода дополнительной информации |
| -w | --warning | выводить коррекции: 0 — только критичные, 1  — важные (по умолчанию), 2 — все |

**Пример**. Команда

z\_run z.zproj ../z.zboco

запускает расчёт в соответствии с файлом проекта z.zproj на сетке, описанной в файле топологии ../z.zboco.

## Файл проекта zproj

Проектный файл служит для организации расчёта, в нём указываются все необходимые настройки расчёта: система уравнений, метод организации шага по времени, порядок расчётной схемы, параметры газа для всех граничных условий и т.д. Файл состоит из набора вложенных разделов, каждый из которых начинается с названия и открывающейся фигурной скобки, а заканчивается закрывающейся фигурной скобкой. Все разделы, кроме раздела set\_flow, присутствуют в единственном числе. Перечисление наиболее важных разделов и полей файла zproj вместе с их кратким описанием находится в таблице 5.

Таблица 5 — Описание разделов и полей файла zproj

| **Раздел или поле** | **Возможные значения** | **Значение по умолчанию** | **Описание** |
| --- | --- | --- | --- |
| **z\_project {** |  |  |  |
| version | 3.00 | 3.00 | Версия библиотеки ZEUS |
| Солвер | | | |
| **solver {** |  |  |  |
| solver | eu ns nut qom sst ltt |  | Уравнения Эйлера  Уравнения Навье-Стокса (NS)  RANS, модель Спаларта-Алмарaса  RANS, модель 𝑞-𝜔  RANS, модель SST  RANS, модель 𝛾SST (учёт ЛТП) |
| tau\_type | local local\_implicit global global\_implicit fractional dual dual\_implicit |  | Локальный явный шаг  Локальный неявный шаг  Глобальный явный шаг  Глобальный неявный шаг  Дробный шаг  Дуальный шаг по явной схеме  Дуальный шаг по неявной схеме |
| decay | godunov roe | godunov | Распад Годунова  Распад Роу |
| reconstr | godunov central kolgan van\_leer roe weno5 weno9 | van\_leer | Схема первого порядка  Центральная разность  Ограничитель Колгана  Ограничитель Ван Лира  Ограничитель Роу  WENO 5-точечный  WENO 9-точечный |
| reconstr\_fun | vector |  | Векторный ограничитель |
| component | component | Компонентный ограничитель в примитивных переменных |
| minmod\_q | 1.0–2.0 | 1.25 | Коэффициент для ограничителей Ван Лира и Роу |
| use\_wl | yes |  | Закон стенки |
| no | no |
| **}** |  |  |  |
| Параметры газа | | | |
| **solver {** |  |  |  |
| kappa | вещ. число | 1.4 | Показатель адиабаты |
| Rgas | вещ. число | 287.053 | Универсальная газовая постоянная |
| Pr\_t\_lam | вещ. число | 0.72 | Ламинарное число Прандтля для темпе-  ратуры |
| Pr\_t\_turb | вещ. число | 0.9 | Турбулентное число Прандтля для температуры |
| mu\_const | вещ. число | -1.0 | Молекулярная вязкость (константа) |
| **}** |  |  |  |
| Параметры, отвечающие за сходимость решения | | | |
| **diminish\_tau {** |  |  |  |
| stab\_coeff | вещ. число |  | Коэффициент стабилизации |
| start\_stab\_coeff | вещ. число |  | Гладкий старт: начальное значение stab\_coeff |
| prelim\_diminish\_tau\_steps | нат. число | 0 | Гладкий старт: количество итераций для линейного выхода на stab\_coeff |
| **}** |  |  |  |
| Параметры, отвечающие за ход решения | | | |
| **steps {** |  |  |  |
| end\_iter | нат. число | 1e9 | Номер итерации завершения расчёта |
| end\_time | вещ. число | 1.e32 | Время завершения расчёта (сек.) |
| end\_max\_misclosure | вещ. число | -60.0 | Значение максимальной невязки завершения расчёта |
| end\_ave\_misclosure | вещ. число | -60.0 | Значение средней невязки завершения расчёта |
| write\_field\_iter\_period | нат. число | 100 000 | Период записи полей (итерации) |
| write\_field\_time\_period | вещ. число | 1000.0 | Период записи полей (сек.) |
| save\_interim\_field | yes |  | Сохранять промежуточные поля |
| no | no |
| first\_averaging\_iter | нат. число | 1e9 | Номер итерации начала осреднения |
| first\_averaging\_time | вещ. число | 1.e32 | Время начала осреднения (сек.) |
| unsteady\_step\_time | вещ. число | 0.0 | Расчёт с постоянным шагом по физическому времени (сек.). Используется совестно с global, fractional, dual, dual\_implicit и global\_implicit. |
| **}** |  |  |  |
| Параметры газа для граничных условий.  Допускается несколько разделов set\_flow, идентификация происходит по названию семейства.  Возможны 3 набора параметров газа  1. Примитивные переменные | | | |
| **set\_flow {** |  |  |  |
| family | строка | outer\_flow | Название семейства, допускается несколько строк family |
| T | вещ. число | 0.0 | Температура |
| u | вещ. число | 0.0 | Компонента вектора скорости |
| v | вещ. число | 0.0 | Компонента вектора скорости |
| w | вещ. число | 0.0 | Компонента вектора скорости |
| p | вещ. число | 0.0 | Давление |
| nut | вещ. число | 0.0 | Коэфф. турбулентной вязкости (солвер nut) |
| q | вещ. число | 0.0 | Величина турбулентных пульсаций скорости (солвер qom) |
| k | вещ. число | 0.0 | Кин. энергия турбулентности (солвер sst) |
| om | вещ. число | 0.0 | Характерная частота турбулентных пульсаций (солверы qom и sst) |
| gamma | вещ. число | 0.0 | Перемежаемость (солвер ltt) |
| **}** |  |  |  |
| 2. Mach, alpha, p0, T0 (АДТ) | | | |
| **set\_flow {** |  |  |  |
| family | строка | outer\_flow | Название семейства, допускается несколько строк family |
| Mach | вещ. число | 0.0 | Число Маха |
| alpha | вещ. число | 0.0 | Угол атаки |
| beta | вещ. число | 0.0 | Угол скольжения |
| p\_total | вещ. число | 0.0 | Полное давление |
| T\_total | вещ. число | 0.0 | Полная температура |
| nut | вещ. число | 0.0 | Коэфф. турбулентной вязкости (солвер nut) |
| q | вещ. число | 0.0 | Величина турбулентных пульсаций скорости (солвер qom) |
| k | вещ. число | 0.0 | Кин. энергия турбулентности (солвер sst) |
| om | вещ. число | 0.0 | Характерная частота турбулентных пульсаций (солверы qom и sst) |
| gamma | вещ. число | 0.0 | Перемежаемость (солвер ltt) |
| **}** |  |  |  |
| 3. Mach, alpha, p, T (полёт) | | | |
| **set\_flow {** |  |  |  |
| family | строка | outer\_flow | Название семейства, допускается несколько строк family |
| Mach | вещ. число | 0.0 | Число Маха |
| alpha | вещ. число | 0.0 | Угол атаки |
| beta | вещ. число | 0.0 | Угол скольжения |
| p | вещ. число | 0.0 | Давление |
| T | вещ. число | 0.0 | Температура |
| nut | вещ. число | 0.0 | Коэфф. турбулентной вязкости (солвер nut) |
| q | вещ. число | 0.0 | Величина турбулентных пульсаций скорости (солвер qom) |
| k | вещ. число | 0.0 | Кин. энергия турбулентности (солвер sst) |
| om | вещ. число | 0.0 | Характерная частота турбулентных пульсаций (солверы qom и sst) |
| gamma | вещ. число | 0.0 | Перемежаемость (солвер ltt) |
| **}** |  |  |  |
| Характерные геометрические размеры тела для расчета аэродинамических коэффициентов | | | |
| **aircraft\_features {** |  |  |  |
| specific\_square |  |  | Характерная площадь |
| specific\_length |  |  | Характерная длина |
| average\_chord |  |  | Средняя аэродинамическая хорда |
| Focus |  |  | Аэродинамический фокус |
| Long\_axis |  |  | Продольная ось самолета |
| Norm\_axis |  |  | Вертикальная ось самолета |
| **}** |  |  |  |
| Дополнительные файлы, необходимые для подготовки данных, расчета и обработки | | | |
| **files {** |  |  |  |
| boco |  |  | Файл топологии |
| ranks |  |  | Файл декомпозиции |
| points |  |  | Файл координат контрольных точек |
| engine |  |  | Файл двигателя |
| **}** |  |  |  |
| Специфические параметры солвера ns | | | |
| **ns {** |  |  |  |
| version | ns | ns | Уравнения Навье-Стокса |
| les |  | LES, модель Смагоринского |
| c\_s | вещ. число | 0.15 | Константа Смагоринского |
| **}** |  |  |  |
| Специфические параметры солвера qom | | | |
| **sst {** |  |  |  |
| version | sst-zeus |  | SST, версия ZEUS |
| sst-2003 | sst-2003 | SST, оригинальная версия 2003 |
| sst-ddes |  | DDES, 2012 |
| sst-iddes |  | IDDES, 2012 |
| kato\_launder | yes |  | Источниковый член Като-Лаундера |
| no | no |
| **}** |  |  |  |
| Специфические параметры солвера nut | | | |
| **nut {** |  |  |  |
| version | sa-zeus | sa-zeus | SA, версия ZEUS |
| sa-neg sa- |  | SA, оригинальная версия |
| sa-ddes |  | DDES |
| sa-iddes |  | IDDES |
| **}** |  |  |  |
| Специфические параметры солвера ltt | | | |
| **ltt {** |  |  |  |
| c\_tu1 | вещ. число | 100.0 | Мин. значение 𝑅𝑒𝑐𝑟 |
| disable\_intermittency | yes |  | Отключение уравнения для перемежаемости |
| no | no |
| lam\_family | строка |  | Включение уравнения для перемежаемости в блоках, содержащих указанное семейство. Применяется в паре с disable\_intermittency, допускается несколько строк. |
| **}** |  |  |  |
| **}** |  |  |  |

# Модуль экспорта данных z\_export

## Описание

Модуль производит вывод графической информации (аэродинамических полей) в форматах fld и cgns.

Формат fld — собственный текстовый формат пакета прикладных программ EWT–ЦАГИ.

Формат cgns (CFD General Notation System) — бинарный формат, де-факто являющийся стандартом для сохранения и передачи результатов аэродинамических расчётов. Данные в формате cgns могут быть импортированы в наиболее распространённые программы визуализации и обработки данных, такие как ParaView или Tecplot.

## Команда вызова, входные данные

Команда вызова модуля z\_export выглядит следующим образом (необязательные элементы указаны в квадратных скобках):

z\_export zproj [zboco] [parameters]

здесь

zproj — путь к файлу проекта;

zboco — путь к файлу топологии (указывается относительно директории, в которой находится файл проекта);

путь к файлу топологии (zboco) может быть указан в файле проекта и/или в команде вызова (в случае, если задействованы оба варианта, путь берётся из команды вызова);

parameters — параметры вызова программы, список параметров и их значение приводится в таблице 6.

Таблица 6 — Список и значение параметров вызова модуля z\_export

|  |  |  |
| --- | --- | --- |
| Короткое написание | Полное написание | Действие |
| -h | --help | вывести справочную информацию |
| -p | --patches | вывести двумерные поля с граней блоков |
| -n | --nodes | вывести значения в узлах сетки (иначе данные выводятся из центров ячеек сетки) |
| -b *name* | --boco *name* | использовать грани с граничным условием *name* |
| -f *name* | --family *name* | использовать блоки или грани, относящиеся к семейству *name* |
| -s *name* | --set *name* | использовать один из наборов выводимых функций (по умолчанию активен набор *base*):  *mesh* — геометрические координаты  *primitive* — *mesh* + примитивные переменные (T, VelocityX, VelocityY, VelocityZ, p, [параметры модели турбулентности])  *base* — *primitive* + rho, p\_total, T\_total, Mach, ∼d\_rho, ∼courant  *solid* — Cp, Cf, d\_wall, delta, y\_plus, Tw, tauw, Qw  *turb —* entropy, VorticityX, VorticityY, VorticityZ, helicity, q\_criterion, ∼mut  *turb-corr* — корреляции турбулентных пульсаций (u′u′, v′v′ и т. д.)  *extra* — M\_is, nu |
| -j *par* | --jump *par* | Если включен набор solid и слово solid есть в названии граничного условия, то все примитивные переменные, а также rho, p\_total, t\_total и Mach «выпрыгивают» c поверхности. Возможны следующие варианты:  *cell* — выпрыгнуть в центр первой ячейки  *bl* — выпрыгнуть из пограничного слоя  *<value>* — выпрыгнуть на <*value*> метров  *mixed* — в центр первой ячейки выпрыгивают только примитивные переменные кроме T и p (сделано для рисования линий тока и ЛТП) |
| -w *par* | --writer *par* | формат записи вывода: cgns (по умолчанию), dat, fld |
| -d | --double | вывод данных с двойной точностью (для dat и fld) |
| -o *name* | --output *name* | имя выходного файла |
| -v | --verbose | режим вывода дополнительной информации |

**Пример**. Команда

z\_export z.zproj ../z.zboco –n –p –b solid\_insulated –s solid –o solid

вызовет модуль z\_export, который в соответствии с файлами z.zproj и ../z.zboco выведет в формате cgns двумерные поля, относящиеся к граничному условию solid\_insulated. Данные, включающие геометрические координаты, примитивные переменные, функции rho, p\_total, T\_total, Mach, ∼d\_rho, ∼courant, а также функции Cp, Cf, d\_wall, delta, y\_plus, Tw, tauw, Qw, будут интерполированы в узлы расчётной сетки и сохранены в файл solid.cgns.

# Модуль обработки расчёта z\_post\_proc

## Описание

Модуль производит обработку расчёта: вычисляет силы и моменты, действующие на твёрдые поверхности со стороны газа. Для воздухозаборника (ВЗ) и сопла, помимо сил, вычисляются также специфические характеристики: потоки импульса через сечения входа в двигатель и выхода из него, коэффициенты массового расхода, тяги, эффективной тяги и т.д. Обработка воздухозаборников и сопел производится по общепринятым методикам.

## Команда вызова, входные данные

Команда вызова модуля z\_post\_proc выглядит следующим образом (необязательные элементы указаны в квадратных скобках):

z\_post\_proc zproj [zboco] [zengine] [parameters]

здесь

zproj — путь к файлу проекта;

zboco — путь к файлу топологии (указывается относительно директории, в которой находится файл проекта);

zengine — путь к файлу двигателя (указывается относительно директории, в которой находится файл проекта);

пути к файлу топологии (zboco) и файлу двигателя (zengine) могут быть указаны в файле проекта и/или в команде вызова (в случае, если задействованы оба варианта, пути берутся из команды вызова);

parameters — параметры вызова программы, список параметров и их значение приводится в таблице 7.

Таблица 7 — Список и значение параметров вызова модуля z\_post\_proc

|  |  |  |
| --- | --- | --- |
| Короткое написание | Полное написание | Действие |
| -h | --help | вывести справочную информацию |
| -d | --dir | название папки с результатами |
| -v | --verbose | режим вывода дополнительной информации |
| -w | --warning | выводить коррекции: 0 — только критичные, 1  — важные (по умолчанию), 2 — все |

**Пример**. Команда

z\_post\_proc z.zproj ../z.zboco z.zengine –d results

вызовет модуль z\_post\_proc, который сделает обработку расчёта в соответствии с файлами z.zproj, ../z.zboco, z.zengine и сохранит результаты в папку results.

## Файл двигателя zengine

Файл двигателя zengine служит для организации обработки расчёта: в нём указывается названия семейств поверхностей, параметры режима работы двигателя и другая необходимая для обработки расчёта информация. Структура файла zengine приводится на рисунке 1. Файл состоит из набора вложенных разделов, каждый из которых начинается с названия и открывающейся фигурной скобки, а заканчивается закрывающейся фигурной скобкой. На первом уровне находится раздел z\_engine (см. рисунок 1), в него вложены разделы inlet, nozzle и airframe. Раздел inlet посвящен воздухозаборнику, в него входят разделы третьего уровня engine\_intake, описывающий сечение входа в двигатель, и throat, описывающий горловое сечение. Раздел nozzle предназначен для писания сопла, в него входят разделы core и fan, содержащие параметры внутреннего и наружного контуров, а также раздел outer\_flow, предназначенный для описания внешних поверхностей сопла. Раздел второго уровня airframe содержит список семейств, относящихся к планеру летательного аппарата. Перечисление всех разделов и полей файла zengine вместе с их кратким описанием находится в таблице 8.

Рисунок 1 — Структура файла zengine

Таблица 8 — Описание разделов и полей файла zengine

| **Раздел или поле** | **Возможные значения** | **Значение по умолчанию** | **Описание** |
| --- | --- | --- | --- |
| **z\_engine {** |  |  |  |
| version | 3.00 | 3.00 | Версия библиотеки ZEUS |
| scale | вещ. число | 1.0 | Мультипликатор, используемый при обработке данных расчёта (например, для полумодели scale = 2) |
| s\_middle | вещ. число | specific\_square \*scale | Площадь миделя мотогондолы (по умолчанию площадь зачитывается в zproj и умножается на scale) |
| Axis | вектор |  | Направление оси OX для обработки (по умолчанию используется направление вектора скорости набегающего потока) |
| Center | вектор | "0 0 0" | Центр входа в двигатель (используется для вычисления неравномерности и моментов) |
| composition | engine | engine | Двигатель (присутствуют и вход, и выход) |
| inlet |  | Воздухозаборник |
| nozzle |  | Сопло |
| airframe |  | Планер |
| use\_boco | yes | yes | Обработка по большим величинам (потоки на гранях ячеек после распада разрыва) |
| no |  | Интерполяция из центров ячеек |
| **Обработка воздухозаборника (допускается 1 раздел)** | | | |
| **inlet {** |  |  |  |
| name | строка | — | Имя |
| s\_0 | вещ. число | 1.0 | Площадь ВЗ по передней кромке |
| **Вход в двигатель (допускается 1 раздел)** | | | |
| **engine\_intake {** |  |  |  |
| N\_r\_sensors | целое число | 0 | Число датчиков (точек для обработки полей в сечении двигателя) по радиальному направлению. Если N\_r\_sensors=0, неравномерность не вычисляется |
| N\_phi\_sensors | целое число | 0 | Число датчиков (точек для обработки полей в сечении двигателя) по окружному направлению. Если N\_phi\_sensors=0, неравномерность не вычисляется |
| intake | Семейство |  | Допускается несколько строк intake |
| solid | Семейство |  | Допускается несколько строк solid |
| **}** |  |  |  |
| **Горло воздухозаборника** | | | |
| **throat {** |  |  |  |
| throat | Семейство |  | Допускается несколько строк throat |
| **}** |  |  |  |
| **}** |  |  |  |
| **Обработка сопла (допускается 1 раздел)** | | | |
| **nozzle {** |  |  |  |
| name | строка | — | Имя |
| type | 1 |  | одноконтурное сопло |
| 2 | двухконтурное сопло с разд. выходом |
| 3 | двухконтурное сопло с внутр. смешением |
| **Первый (внутренний) контур** | | | |
| **core {** |  |  |  |
| s\_sonic | вещ. число | 1.0 | Площадь звукового сечения |
| G\_factor | вещ. число | 1.0 | Множитель, на который умножаются расход и тяга при обработке |
| intake | Семейство |  | Допускается несколько строк intake |
| solid | Семейство |  | Допускается несколько строк solid |
| **}** |  |  |  |
| **Второй (наружный) контур** | | | |
| **fan {** |  |  |  |
| s\_sonic | вещ. число | 1.0 | Площадь звукового сечения |
| G\_factor | вещ. число | 1.0 | Множитель, на который умножаются расход и тяга при обработке |
| intake | Семейство |  | Допускается несколько строк intake |
| solid | Семейство |  | Допускается несколько строк solid |
| **}** |  |  |  |
| **Внешние вход и поверхности** | | | |
| **outer\_flow {** |  |  |  |
| intake | Семейство |  | Допускается несколько строк intake сопла |
| solid | Семейство |  | Допускается несколько строк solid сопла |
| **}** |  |  |  |
| **}** |  |  |  |
| **Обработка планера** | | | |
| **airframe {** |  |  |  |
| name | строка | — | Имя |
| solid | Семейство |  | Допускается несколько строк solid |
| **}** |  |  |  |
| **}** |  |  |  |

## Выходные данные

Все выходные данные модуль z\_post\_proc записывает в файл zresult. Структура выходного файла zresult почти полностью повторяет структуру входного файла zengine с единственным отличием: к разделам inlet, nozzle и airframe добавлен раздел total, содержащий суммарные характеристики всей компоновки (см. рисунок 2). Перечисление всех разделов и полей файла zresult вместе с их кратким описанием находится в таблице 9.

Рисунок 2 — Структура файла zresult

Таблица 9 — Описание файла zresult

| **Поле** | **Единица измерения** | **Описание** |
| --- | --- | --- |
| **Характеристики воздухозаборника (ВЗ)** | | |
| **Inlet features** |  |  |
| p\_total\_min | Па | Минимум полного давления |
| p\_total\_max | Па | Максимум полного давления |
| p\_total | Па | Среднее полное давления |
| p\_pitot | Па | Среднее полное давление Пито |
| T\_total\_min | К | Минимум температуры торможения |
| T\_total\_max | К | Максимум температуры торможения |
| T\_total | К | Средняя температура торможения |
| alpha\_min | ° | Скосы потока на входе в двигатель |
| alpha\_max | ° |
| beta\_min | ° |
| beta\_max | ° |
| lambda | — | Приведённая скорость в сечении входа в двигатель |
| q\_lambda | — | Значение функции в сечении входа в двигатель |
| nu | — | Коэффициент восстановления полного давления в сечении входа в двигатель |
| S | м2 | Площадь сечения входа в двигатель |
| G | кг/с | Расход воздуха ВЗ |
| J | Н | Поток импульса через сечение входа в двигатель |
| Se | м2 | Площадь засасываемой ВЗ струйки тока |
| f | — | Коэффициент расхода воздуха ВЗ |
| Je | Н | Поток импульса засасываемой струйки тока на бесконечности |
| F internal | Н | «Внутреннее сопротивление ВЗ», |
| F (*family-1*) | Н | Сила, действующая на поверхности семейства «*family-1*» |
| F (*family-2*) | Н | Сила, действующая на поверхности семейства «*family-2*» |
| … |  |  |
| F fric (family-1) | Н | Сила трения, действующая на поверхности семейства «*family-1*» (входит в F (*family-1*)) |
| F fric (family-1) | Н | Сила трения, действующая на поверхности семейства «*family-2*» (входит в F (*family-2*)) |
| … |  |  |
| F sum | Н | Суммарная сила, действующая на все поверхности ВЗ |
| F fric sum | Н | Суммарная сила трения, действующая на все поверхности ВЗ (входит в F sum) |
| F external | Н | «Внешнее сопротивление ВЗ», |
| Cx Cy Cz | — | Аэродинамические коэффициенты внешней силы |
| **Характеристики сопла** | | |
| **Nozzle features** |  |  |
| S sonic | м2 | Площадь критического сечения |
| mu | — | Коэффициент расхода |
| p | — | Коэффициент тяги |
| p eff | — | Коэффициент эффективной тяги |
| F external | Н | Суммарная сила, действующая на внешнюю поверхность сопла |
| F external fric | Н | Сила трения, действующая на внешнюю поверхность сопла (входит в F sum) |
| Cx Cy Cz | — | Аэродинамические коэффициенты внешней силы |
| NPR (core) | — | Степень понижения давления, |
| S sonic (core) | м2 | Площадь критического сечения сопла внутреннего контура |
| mu (core) | — | Коэффициент расхода сопла внутреннего контура |
| p (core) | — | Коэффициент тяги сопла внутреннего контура |
| NPR (fan) | — | Степень понижения давления, |
| S sonic (fan) | м2 | Площадь критического сечения сопла наружного контура |
| mu (fan) | — | Коэффициент расхода сопла наружного контура |
| p (fan) | — | Коэффициент тяги сопла наружного контура |
| G | кг/с | Суммарный расход сопла |
| G ideal | кг/с | Идеальный расход сопла |
| G (core) | кг/с | Расход сопла внутреннего контура |
| G (fan) | кг/с | Расход сопла наружного контура |
| BPR | — | Степень двухконтурности сопла |
| P | Н | Тяга сопла |
| P fric | Н | Сила трения, действующая на внутренние поверхности сопла (входит в P) |
| P eff | Н | Эффективная тяга сопла |
| P ideal | Н | Идеальная тяга сопла |
| P (core) | Н | Тяга сопла внутреннего контура |
| P fric (core) | Н | Сила трения, действующая на внутренние поверхности сопла внутреннего контура (входит в P (core)) |
| P eff (core) | Н | Эффективная тяга сопла внутреннего контура |
| P ideal (core) | Н | Идеальная тяга сопла внутреннего контура |
| P (fan) | Н | Тяга сопла наружного контура |
| P fric (fan) | Н | Сила трения, действующая на внутренние поверхности сопла наружного контура (входит в P (fan)) |
| P eff (fan) | Н | Эффективная тяга сопла наружного контура |
| P ideal (fan) | Н | Идеальная тяга сопла наружного контура |
| J (*family*) | Н | Поток импульса через поверхности семейства «*family*» |
| F (*family*) | Н | Сила, действующая на поверхности семейства «*family*» |
| … |  |  |
| F fric (*family*) | Н | Сила трения, действующая на поверхности семейства «*family*» (входит в F (*family*)) |
| … |  |  |
| J (*family*) / P ideal | — | Поток импульса через поверхности семейства «*family*», отнесённый к идеальной тяге сопла |
| F (*family*) / P ideal | — | Сила, действующая на поверхности семейства «*family*», отнесённая к идеальной тяге сопла |
| … |  |  |
| F fric (*family*)/ P ideal | — | Сила трения, действующая на поверхности семейства «*family*», отнесённая к идеальной тяге сопла |
| … |  |  |
| **Характеристики планера** | | |
| **Airframe features** |  |  |
| F (*family*) | Н | Сила, действующая на поверхности семейства «*family*» |
| … |  |  |
| F fric (*family*) | Н | Сила трения, действующая на поверхности семейства «*family*» (входит в F (*family*)) |
| … |  |  |
| F sum | Н | Суммарная сила, действующая на все поверхности планера |
| F fric sum | Н | Суммарная сила трения, действующая на все поверхности планера (входит в F sum) |
| Cx Cy Cz |  | Аэродинамические коэффициенты планера |
| **Суммарные характеристики** | | |
| **Total** |  |  |
| Angles | ° | Направление оси OX системы координат, в которой производилась обработка |
| Focus | м | Точка, относительно которой посчитаны моменты сил |
| Engine Center | м | Центр входа в двигатель, использовавшийся для вычисления неравномерностей на входе в двигатель |
| F external | Н | Суммарная внешняя сила, действующая на компоновку |
| Cx Cy Cz | — | Аэродинамические коэффициенты суммарной внешней силы |
| P ideal | Н | Идеальная тяга двигателя |
| P | Н | Тяга двигателя |
| P eff | Н | Эффективная тяга двигателя, |
| dP eff | — | Потери эффективной тяги двигателя, |

# Декомпозиция расчетной сетки и поля

Для повышения эффективности расчёта на одном или нескольких многоядерных узлах необходимо учитывать сбалансированность загрузки вычислительной системы. Утилита decomp.py позволяет исследовать балансировку и, в случае необходимости, разбить блоки исходной структурированной сетки на более мелкие и тем самым уменьшить исходную несбалансированность.

## Скрипт decomp.py. Подкоманды и флаги

Общий формат вызова:

decomp.py [-h] [-c N] -r N [-s {auto,equal,slice}] [-v] [-d] {boco,field,graph,mesh,split,view} ...

Напомним, что знак $ не является частью команды. Это приглашение командной среды. В квадрантных скобках указываются необязательные флаги или аргументы, а в фигурных скобках через запятую перечисляются варианты выбора. Троеточие в конце означает, что у подкоманд (boco, field и др.) имеются свои собственные флаги (нижнего уровня).

Каждый успешно завершившийся запуск фиксируется в файле \*\_decomp.log (где \* означает имя проекта) в виде строки с запущенной командой и полным списком её аргументов, включая выбранные умолчания. Эта информация может пригодиться, например, при [отдельной декомпозиции полей](http://192.168.103.124/decomp-guide/#Usage-field-decomposition) или [передекомпозиции](http://192.168.103.124/decomp-guide/#Usage-decomposition-repeat).

**Флаги**верхнего**уровня**

Эти флаги, перечисленные в таблицах 10 и 11, указываются сразу за командой вызова decomp.py и обеспечивают управление процессом разбиения блоков.

Таблица 10 — Основные флаги

|  |  |
| --- | --- |
| -h, --help | Вывести на экран краткую справку об использовании. |
| -r, --n-ranks N | Количество желаемых MPI процессов (рэнков), на которых предполагается проводить расчёт. Обязательный флаг. |
| -c, --max-cells N | Mаксимальное желаемое количество ячеек в блоке. По умолчанию значение равно среднему количеству ячеек в расчёте на один MPI рэнк. Этот флаг используется в дополнение к предыдущему для более [тонкой](http://192.168.103.124/decomp-guide/#Usage-tuning) настройки разбиения. |
| -v, --verbose | Вывести на экран дополнительную информацию, например, сообщения о чтении, записи и удалении файлов. |

Таблица 11 — Экспертные флаги

|  |  |
| --- | --- |
| -s,  --splits {auto,equal,slice} | Способ разбиения блоков. По умолчанию auto. |
| -d, --debug-output | Как -v плюс ещё больше деталей в целях отладки. |

**Подкоманды и флаги нижнего уровня**

Подкоманды определяют выбор действия, которое следует произвести. Каждая из таких подкоманд имеет свои обязательные аргументы и флаги — флаги нижнего уровня, которые указываются после имени подкоманды.

Пути всех файлов указываются либо полностью, либо относительно текущего рабочего каталога, где запущен скрипт.

**Анализ**

* + view — выводит на экран статистику декомпозиции: потенциальное ускорение (без учёта затрат на обмены), дисбаланс загрузки (со знаком “+” идёт перегрузка сверх среднего уровня, а со знаком “-” — недогрузка) и прочее. Флаги подкоманды перечислены в таблице 12.

$ decomp.py ... view [-h] [-i] boco\_file

Таблица 12 — Флаги подкоманды view

|  |  |
| --- | --- |
| boco\_file | Путь к файлу с описанием топологии блоков и граничных условий (\*.zboco). |
| -h, --help | Вывести на экран краткую справку об использовании. |
| -i, --origin | Вычислить статистику для исходной сетки, без декомпозиции. |

* + graph — в дополнение к view рисует в файл статистику полученного разбиения: распределение размеров блоков расчётной сетки, распределение загрузки по рэнкам и потенциальную масштабируемость (без учёта затрат на обмены). Флаги подкоманды указаны в таблице 13.

$ decomp.py ... graph [-h] [-b N] [-f {pdf,png,svg}] [-i] boco\_file

Таблица 13 — Флаги подкоманды graph

|  |  |
| --- | --- |
| boco\_file | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -h, --help | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -b, --bins N | Количество столбцов в гистограмме распределения размеров блоков. По умолчанию 50. |
| -f, --format {pdf,png,svg} | Формат выходного изображения. По умолчанию png. |
| -i, --origin | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |

**Декомпозиция**

* + mesh — выполняет декомпозицию топологии и блоков расчётной сетки. Записывает новый \*.zboco файл и сохраняет старый под именем \*\_orig.zboco. Флаги подкоманды указаны в таблице 14.

$ decomp.py ... mesh [-h] [-n N] boco\_file

Таблица 14 — Флаги подкоманды mesh

|  |  |
| --- | --- |
| boco\_file | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -h, --help | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -n, --n-threads N | Количество нитей для параллельной декомпозиции. По умолчанию 1. |

* + field — выполняет декомпозицию блоков расчётного поля, включая сохранённые патчи. Необходимо указать исходный BoCo-файл (часто это \*\_orig.zboco, см. mesh). Флаги подкоманды указаны в таблице 15.

$ decomp.py ... field [-h] [-n N] proj\_file boco\_file

Таблица 15 — Флаги подкоманды field

|  |  |
| --- | --- |
| proj\_file | Путь к файлу с описанием проекта (\*.zproj). |
| boco\_file | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -h, --help | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -n, --n-threads N | См. [mesh](http://192.168.103.124/decomp-guide/#Decomp-subcommands-decomposition). |

* + split — выполняет декомпозицию сразу целиком: топология, сетка и поле. Равносильна выполнению команд mesh и field подряд.

## Скрипт recomp.py

Если была выполнена декомпозиция, то последующий перенос поля на другую сетку или переразбиение с другими настройками требует восстановления полей в исходные (до разбиения) блоки. Эта операция может быть выполнена при помощи скрипта recomp.py. Флаги вызова этого скрипта указаны в таблице 16.

$ recomp.py [-h] [-n N] [-v] [-d] proj\_file boco\_file

Таблица 16 — Флаги cкрипта  recomp.py

|  |  |
| --- | --- |
| proj\_file | См. [field](http://192.168.103.124/decomp-guide/#Decomp-subcommands-decomposition). |
| boco\_file | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -h, --help | См. [view](http://192.168.103.124/decomp-guide/#Decomp-subcommands-analysis). |
| -n, --n-threads N | См. [mesh](http://192.168.103.124/decomp-guide/#Decomp-subcommands-decomposition). |
| -v, --verbose | См. основные флаги. |
| -d, --debug-output | См. [экспертные флаги](http://192.168.103.124/decomp-guide/#Decomp-flags-expert). |

## Типичное использование

До начала декомпозиции необходимо понять исходное состояние сетки, её пригодность к расчёту на предполагаемых ресурсах — количество MPI процессов, или рэнков, равное обычно количеству доступных ядер CPU.

**Предварительный анализ задачи**

Рассмотрим пример. Пусть имеется 60 расчётных узлов по 12 ядер CPU каждый, то есть всего 720 ядер CPU. Запустим из директории, где хранится расчётная сетка, следующую команду:

$ decomp.py -r 720 graph -i z.zboco

В результате этого в текущей директории появится файл z\_decomp.png. Этот рисунок состоит из трёх графиков (рисунки 3 – 5).

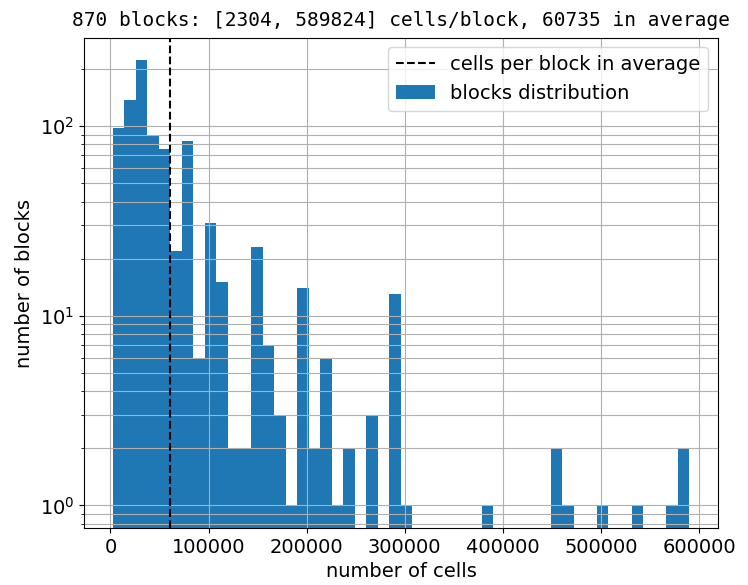


Рисунок 3 — Распределение размеров блоков сетки

На рисунке 3 отображается распределение размеров блоков сетки. Всего имеется 870 блоков. Минимальное количество ячеек в блоке составляет 2304, максимальное — 589824 ячейки. В среднем 60735 ячеек на блок. Из графика на рисунке 3 видно, что имеется небольшое количество (около десятка) очень больших блоков. Вероятно, эти блоки будут мешать сбалансировать загрузку при запуске расчёта.

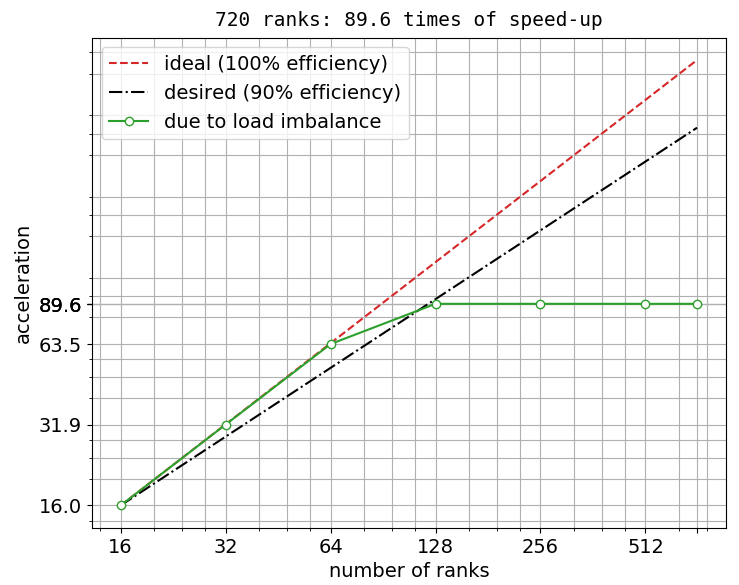


Рисунок 4 — Кривая масштабируемости (без учёта обменов)

Кривая масштабируемости задачи, изображенная на рисунке 4 подтверждает это предположение. Видно, что, начиная с 64-х рэнков, потенциальное ускорение расчёта заметно снижается и совсем исчезает при запуске свыше 128-ми рэнков. То есть, без декомпозиции имеет смысл запускать расчёт не более, чем на 64-х ядрах CPU.

Примерное ускорение, которое возможно получить, — до 90 раз в сравнении с запуском на одном ядре CPU.

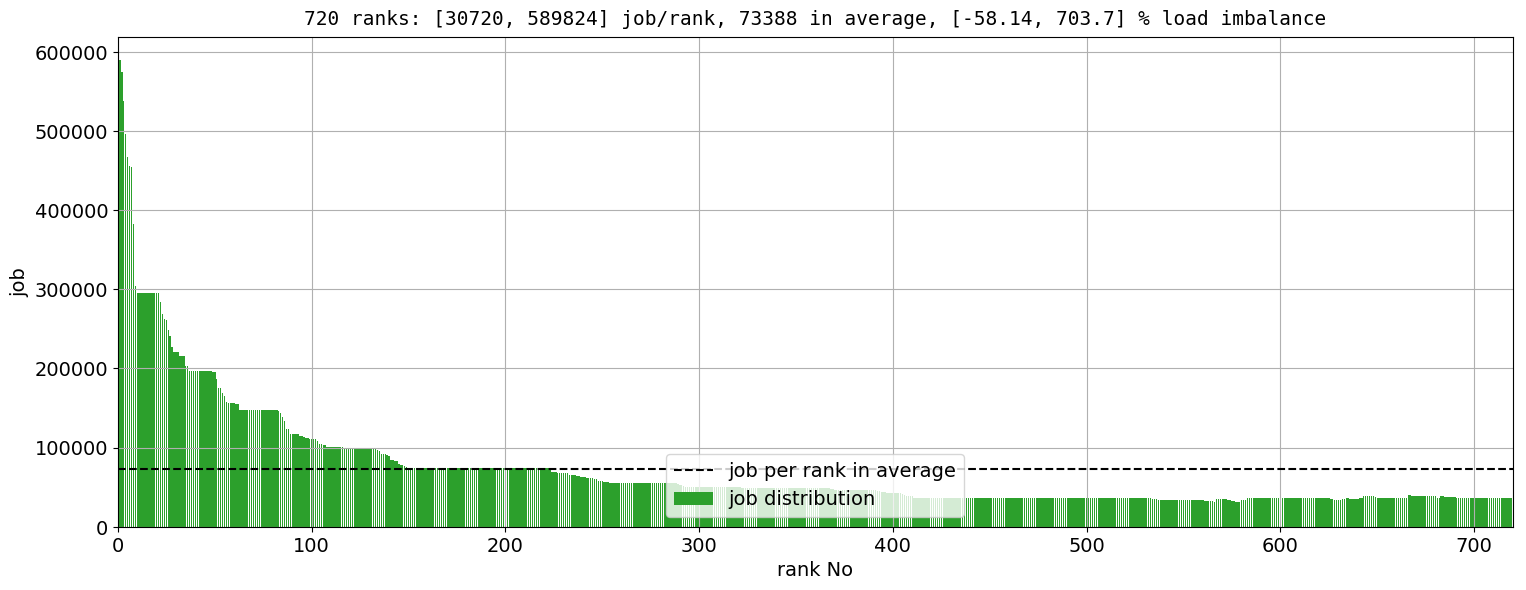


Рисунок 5 — Распределение загрузки по рэнкам

На рисунке 5 представлено начальное распределение загрузки по рэнкам. Всего имеется 720 рэнков, минимальная загрузка рэнка составляет 30720 ед. На данный момент единица измерения — это просто количество ячеек. Максимальная — 589824 ед. Заметим, что это как раз размер максимального блока. В среднем загрузка равна 73388 ед. Относительно этой средней загрузки (чёрная пунктирная линия) имеются рэнк с максимальной недогрузкой -58.14% и рэнк с максимальной перегрузкой 703.7%.

Распределение показывает, что небольшая часть рэнков сильно перегружена, и в процессе счёта все остальные рэнки будут вынуждены ждать их. Это и составляет причину низкой эффективности на указанных вычислительных ресурсах. Исходя из разумных соображений, декомпозицию имеет смысл делать, если максимальная перегрузка рэнков превосходит 10–20%.

Продолжим анализ. Попросим скрипт выполнить декомпозицию топологии и вывести статистику:

$ decomp.py -r 720 graph z.zboco

...

Potential speed-up: 614.3 times

Load imbalance: [-18.37, 17.21] %

...

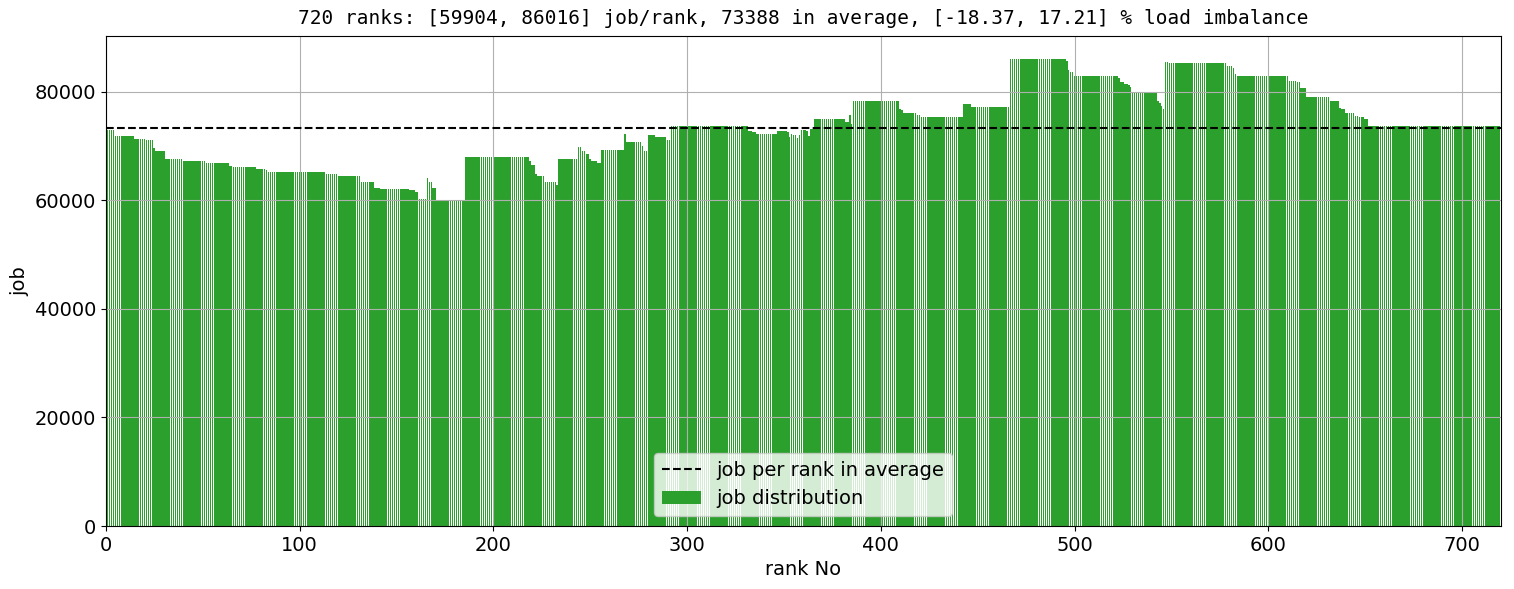


Рисунок 6 — Распределение загрузки по рэнкам после декомпозиции с параметрами по умолчанию

Из вывода программы и графика на рисунке 6 видно, что ситуация заметно улучшилась и загрузка стала более равномерной. В принципе, на этом можно остановиться, тем более, если мы хотим сэкономить на количестве блоков. Слишком большое количество блоков приводит к излишним затратам памяти и увеличению числа обменов данными между блоками.

Предположим, что теперь в нашем распоряжении стало 125 расчётных узлов, то есть 1500 ядер CPU. Декомпозиция по умолчанию приводит к следующему результату:

$ decomp.py -r 1500 graph z.zboco

...

Potential speed-up: 1096.4 times

Load imbalance: [-21.51, 36.81] %

...

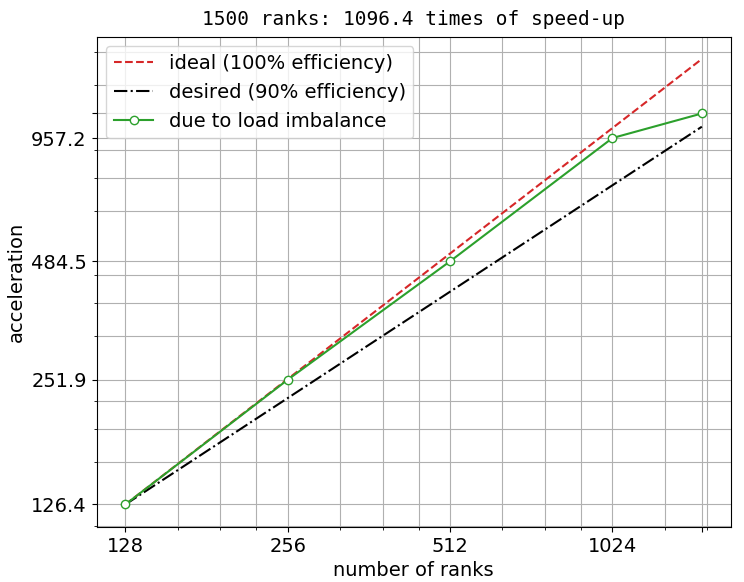


Рисунок 7 — Кривая масштабируемости после изменения числа доступных рэнков

Очевидно, при таком разбиении считать на 1500 ядрах CPU становится неэффективно. Предел около – 1024 ядер CPU.

Понятно, что блоки достаточно велики, а их количество в пересчёте на один рэнк мало, чтобы суметь их хорошо сбалансировать. Заметим, что блок подлежит разбиению, если количество ячеек в нём превосходит некую заданную величину. До сих пор мы никак не указывали этот параметр явно. По умолчанию его значение равно среднему количеству ячеек в расчёте на один MPI рэнк.

Попробуем уменьшить этот параметр, перебрав несколько вариантов. Отталкиваться будем от среднего, которое указано над графиком распределения загрузки по рэнкам. В нашем случае это 35226 ячеек:

$ decomp.py -r 1500 -c 30000 graph z.zboco

...

Potential speed-up: 1118.7 times

Load imbalance: [-25.06, 34.08] %

...

Получено незначительное улучшение баланса. Сделаем еще одну итерацию, уменьшив пороговое значение числа ячеек:

$ decomp.py -r 1500 -c 25000 graph z.zboco

...

Potential speed-up: 1253.8 times

Load imbalance: [-30.23, 19.64] %

...

Балансировка улучшилась, повторим предыдущий шаг:

$ decomp.py -r 1500 -c 15000 graph z.zboco

...

Blocks number after decomposition: 4116

...

Potential speed-up: 1355.7 times

Load imbalance: [-22.6, 10.64] %

...

Отметим, что блоков становится больше, это может привести к уменьшению скорости счета из-за повышения интенсивности обменов. Приняв во внимание этот факт, будем считать, что компромиссное разбиение достигнуто, см. рисунок 8.

Отметим, что небольшие “провалы” в загрузке на рисунке 8 слабо сказываются на производительности (немногие ждут многих). А вот небольшие “выступы” (перегрузка части рэнков) — очень сильно (многие ждут немногих).

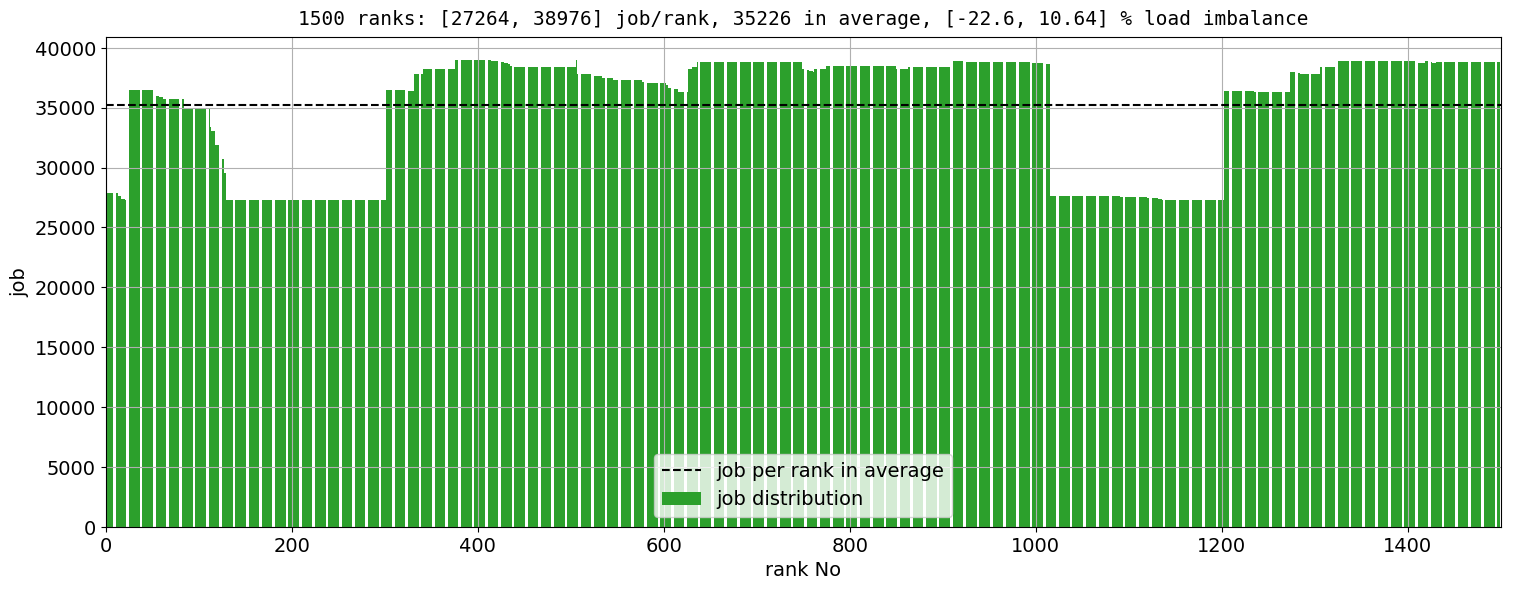


Рисунок 8 — Окончательное распределение загрузки по рэнкам

Когда параметры декомпозиции подобраны удовлетворительным образом, необходимо сохранить копию исходной сетки на случай, если что-то пойдёт не так, а также просто для повторного использования, например, [передекомпозиции](http://192.168.103.124/decomp-guide/#Usage-decomposition-repeat) с другими параметрами.

Теперь запускаем команду ([mesh](http://192.168.103.124/decomp-guide/#Decomp-subcommands-decomposition)):

$ decomp.py -r 1500 -c 15000 mesh -n 8 z.zboco

Флаг -n 8 указывает, что следует запустить 8 нитей исполнения. Это заметно ускоряет процесс, поскольку блоки обрабатываются параллельно.

Если уже есть готовое начальное поле, можно разбить блоки сетки и блоки поля за один раз командой ([split](http://192.168.103.124/decomp-guide/#Decomp-subcommands-decomposition)):

$ decomp.py -r 1500 -c 15000 split -n 8 m085a190\_ltt/z.zproj z.zboco

**Отдельная декомпозиция полей**

Как правило, необходимость в отдельной декомпозиции полей возникает, когда имеется несколько режимов, на которых требуется провести расчёт.

Итак, положим, что уже выполнена декомпозиция сетки. В результате её выполнения запущенная команда сохранена в лог-файле \*\_decomp.log, а также сохранён оригинальный файл с топологией под именем \*\_orig.zboco. При необходимости из лог-файла легко восстановить параметры декомпозиции.

Декомпозиция полей должна быть выполнена точно с теми же параметрами (максимальное количество ячеек в блоке и алгоритм разбиения), как и декомпозиция сетки.

Используя эту информацию, запустим команду ([field](http://192.168.103.124/decomp-guide/#Decomp-subcommands-decomposition)):

$ decomp.py -r 1500 -c 15000 field -n 8 m085a190\_ltt/z.zproj z\_orig.zboco

**Восстановление полей**

Предположим, что после декомпозиции проведён расчёт, получены поля течения. И теперь необходимо продолжить расчёт на другом количестве ресурсов, то есть произвести передекомпозицию с другими параметрами. Или необходимо использовать это поле для пост-обработки в рамках исходной топологии блоков.

В таких случаях необходимо восстановить поля в исходной топологии, запустив команду ([recomp.py](http://192.168.103.124/decomp-guide/#Recomp-py)):

$ recomp.py -n 8 m085a190\_ltt/z.zproj z.zboco

Здесь \*.zboco — это с декомпозированной топологией.

**Повторная декомпозиция**

Повторная декомпозиция запрещена. То есть выполнение декомпозиции над уже декомпозированной задачей по техническим причинам не допускается. Скрипт выдаст сообщение об ошибке.

Однако, скорее всего, это не то, что на самом деле необходимо пользователю. Для передекомпозиции и некоторых других задач используйте восстановление полей.

# Подготовка и запуск расчета в EWT решателем ZEUS

В этом разделе приведена пошаговая инструкция по запуску расчета решателем ZEUS пакета EWT-128-ЦАГИ. В качестве примера рассмотрим простую двумерную задачу – обтекание профиля NACA 23012. Сетка для этого примера построена при помощи специализированных программ. В программном пакете EWT-128-ЦАГИ используются многоблочные структурированные сетки. Для импорта используется общепринятый формат CGNS.

## Конвертация CGNS во внутренний формат zboco/zmesh

Для конвертации сетки из формата CGNS во внутренний формат zboco/zmesh используется программа z\_import\_cgns.

Все программы пакета EWT-128-ЦАГИ, относящиеся к решателю ZEUS, являются консольными приложениями. Для их выполнения следует использовать интерпретатор командной строки. В случае операционной системы Ubuntu таким интерпретатором является, например, bash.

Будем считать, что все исполняемые файлы решателя ZEUS находятся в директории с адресом ~/zeus-3.28/bin-lin.

* + Запустите терминал и перейдите в директорию, где находится файл CGNS
  + Выполните команду  
    ~/zeus-3.28/bin-lin/z\_import\_cgns -s 0.001 -d naca23012 -v naca\_23012\_3d\_block\_3d\_geo\_2015.cgns

Рассмотрим используемые в примере параметры команды, которые приведены в таблице 17.

Таблица 17 — Используемые в примере параметры команды z\_import\_cgns

|  |  |
| --- | --- |
| Параметр | Действие |
| -s 0.001 | масштабирование сетки, перевод из миллиметров в метры |
| -d naca23012 | создание папки naca23012, в которую будут помещены создаваемые файлы |
| -v | вывод информации о ходе выполнения команды на экран |
|  |  |

В результате выполнения программы z\_import\_cgns будут созданы файлы сетки zmesh и файл с деревом граничных условий z.zboco

## Проверка качества расчетной сетки

После конвертации сетки бывает полезно проверить ее пригодность к расчету. Обязательным условием для успешного запуска расчета является обеспечение положительных объемов ячеек расчетной сетки. Для проверки минимальных параметров качества можно использовать скрипт mesh\_test.py, написанный на языке Python.

В терминале перейдем в директорию, содержащую файлы сетки и файл с деревом граничных условий z.zboco:

* + cd naca23012

В указанной папке выполните команду  
 python ~/zeus-3.28/bin-lin/mesh\_test.py ./z.zboco

Программа выведет на экран информацию о каждом блоке расчетной сетки и обобщенные данные:

* + число узлов
  + число ячеек
  + число граней
  + минимальный и максимальный объемы ячейки

В случае, когда сообщения об отрицательных объемах отсутствуют, можно продолжить процедуру подготовки расчета.

## Генерация служебных файлов с информацией о стыковке блоков

Блоки расчетной сетки могут стыковаться друг с другом "узел-в-узел" (при помощи граничного условия joint) и произвольно (при помощи граничного условия connect). Для того чтобы не выполнять поиск соседей на каждой итерации, информация о стыковке записывается в служебные файлы zjoint и zconnect. Для генерация служебных файлов с информацией о стыковке блоков используется программа z\_make\_connection.

В интерпретаторе командной строки cmd.exe в папке с сеткой и деревом граничных условий выполните команду:  
 ~/zeus-3.28/bin-lin/z\_make\_connection ./z.zboco

На экран будет выведена информация о максимальных расстояниях между ячейками для граничных условий join и connect, а в папке появятся файлы z.zjoint и z.zconnect

## Генерация служебных файлов с информацией о расстоянии до твердой стенки

Для корректной работы модели турбулентности необходима информация о расстоянии от центра ячейки расчетной сетки до твердой поверхности. Поскольку в процессе расчета это расстояние не изменяется, его можно вычислить в процессе подготовки к запуску задачи. Для этого используется программа z\_dwall . Команда может выполняться параллельно, для этого ее нужно вызвать при помощи MPI.

* + В терминале в директории с сеткой и деревом граничных условий выполните команду  
     mpiexec -n 8 ~/zeus-3.28/bin-lin/z\_dwall ./z.zboco
  + После ключа -n указывается количество потоков (рэнков, ranks). Оно не должно быть больше количества блоков расчетной сетки. Не имеет смысла использовать число потоков, превышающее количество ядер.
  + Результатом работы программы z\_dwall являются файлы zdwall. Их количество равно числу блоков расчетной сетки.

## Параметры расчетов (файл проекта)

Вся информация, касающаяся расчетов хранится в файле проекта – с расширением zproj. Под каждый режим расчета целесообразно создать собственную папку и поместить туда файл проекта. Содержание базового файла проекта приведено в листинге.

z\_project {  
 version 3.00  
 solver {  
 solver nut  
 tau\_type local\_implicit  
 use\_wl no  
 reconstr\_fun component  
 }  
 nut {  
 version sa-zeus  
 }  
 diminish\_tau {  
 stab\_coeff 100  
 }  
 steps {  
 end\_iter 10000  
 write\_field\_iter\_period 500  
 first\_averaging\_iter 9800  
 }  
 aircraft\_features {  
 specific\_square 0.3  
 Focus "0.075 0 0"  
 average\_chord 0.3  
 specific\_length 1  
 }  
 set\_flow {  
 family outer\_flow  
 family bottom  
 family top  
 family in  
 family out  
 Mach 0.4  
 alpha 4  
 beta 0.0  
 p\_total 101325  
 T\_total 288  
 k 4.0  
 om 7777.778  
 nut 1.8e-6  
 }  
 files {  
 boco ../z.zboco  
 }  
 }

В блоке solver содержится базовая информация о решателе: модель турбулентности (nut, Спалларт-Алмарас), тип шага по времени (local-implicit , локальный шаг по времени, неявная схема)...

Далее идет блок, описывающий настройки выбранной модели турбулентности. В нашем случае это nut. Будет использоваться версия sa-zeus. Бывают и другие, подробности описаны в разделе 4.3.

В блоке diminish\_tau описаны настройки шага по времени. Мы только указываем значение локального коэффициента устойчивости схемы (аналог числа CFL).

Число шагов по времени, частота записи и параметры осреднения решения указаны в блоке steps.

Блок aircraft\_features содержит геометрические параметры модели, используемые для вычисления коэффициентов аэродинамических сил и моментов.

Наконец, в блоках set\_flow указываются значения газодинамических величин для семейств с граничными условиями, которые позволяют это сделать. В приведенном примере задаются параметры для семейств, у которых в файле zboco указаны граничные условия riemann , а также для семейства outer\_flow, к которому принадлежат ячейки (объемы) расчетной сетки. То есть, в данном случае, в этом блоке инициализируются параметры на границах и внутри расчетной области.

В блоке files прописываются пути к служебным файлам, в данном случае к файлу с деревом граничных условий zboco. Он находится в родительской директории.

Создадим в рабочей директории другую директорию, в которой будут храниться файл проекта и поля. Исходя из режима расчета, назовем ее m04a000. Поместим в нее файл проекта с описанным содержимым z.zproj.

## Генерация начального поля

В случае, когда расчет производится впервые (новая геометрия, новая расчетная сетка) необходимо инициализировать поле. Это можно сделать при помощи программы z\_init\_field. Команда может выполняться параллельно, для этого ее нужно вызвать при помощи MPI.

В терминале в директории с файлом проекта z.zproj выполните команду:  
mpiexec -n 8 ~/zeus-3.28/bin-lin/z\_init\_field ./z.zproj ../z.zboco

Программа z\_init\_field получает на вход пути к файлу проекта zproj и файлу с деревом граничных условий zboco

После ключа -n указывается количество потоков (рэнков, ranks). Оно не должно быть больше количества блоков расчетной сетки. Также не имеет смысла использовать число потоков, превышающее количество ядер.

Результатом работы программы z\_init\_field являются файлы zfield, содержащие значения газодинамических переменных в каждом узле расчетной сетки. Количество файлы zfield равно числу блоков расчетной сетки. Изначально в файлах zfield содержится равномерное поле со значениями, указанными в соответствующих семействах файла проекта z.zproj. При дальнейшей работе решателя с периодичностью, указанной в файле проекта (в нашем случае в переменной write\_field\_iter\_period) в файлы zfield будут записываться новые значения полей.

## Запуск расчета

Проведение расчета обеспечивается программой-решателем (solver) z\_run . Команда может выполняться параллельно, для этого ее нужно вызвать при помощи MPI.

В терминале в директории с файлом проекта z.zproj выполните команду:  
 mpiexec -n 8 ~/zeus-3.28/bin-lin/z\_run ./z.zproj ../z.zboco

Программа z\_run получает на вход пути к файлу проекта zproj и файлу с деревом граничных условий zboco

После успешного запуска на экран будет выводиться краткая информация о каждой итерации (периодичность вывода можно изменить в файле zproj). Пример такого вывода приведен на рисунке 9. Выводятся номер итерации, значение локального коэффициента устойчивости схемы, величина шага по времени, максимальная и средняя невязки плотности и значения аэродинамических коэффициентов от сил давления и трения.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 — Вывод программы z\_run

С периодичностью write\_field\_iter\_period в директории проекта будут обновляться файлы полей zfield и zstat. Последний содержит информацию по итерациям, выводимую на экран, плюс коэффициенты моментов. Содержимое файла zstat может быть визуализировано любым доступным способом. В составе пакета содержится программа zglance, которая позволяет это сделать. Скриншот окна программы zglance с визуализацией невязки решения приведен на рисунке 10.

В директории проекта содержатся также служебные файлы с расширением .key, их удаление позволяет отдать следующие команды решателю:

* + read.key - обновить информацию из файла проекта z.zproj. Таким образом можно, например, поменять число итераций, периодичность записи, величину локального коэффициента устойчивости схемы или шага по времени;
  + write.key - выполнить сохранение поля, не дожидаясь окончания write\_field\_iter\_period и продолжить расчет;
  + run.key - остановить расчет и записать поле.

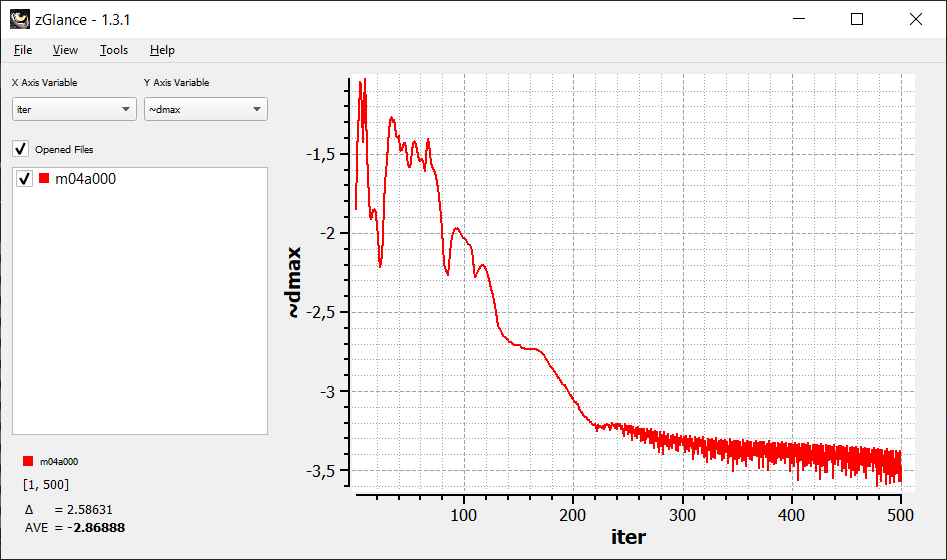


Рисунок 10 — Визуализация невязки в программе zglance

## Визуализация поля

Программа z\_export позволяет экспортировать поле из внутреннего формата zfield в широко используемый формат cgns. Опции программы подробно описаны в соответствующем разделе документации.

Вызов:  
 ~/zeus-3.28/bin-lin/z\_export.exe -p -n -f sym\_left -s solid -o symmetry ./z.zproj ../z.zboco   
создаст в директории проекта файл symmetry.cgns, который может быть визуализирован, например, при помощи открытой программы ParaView

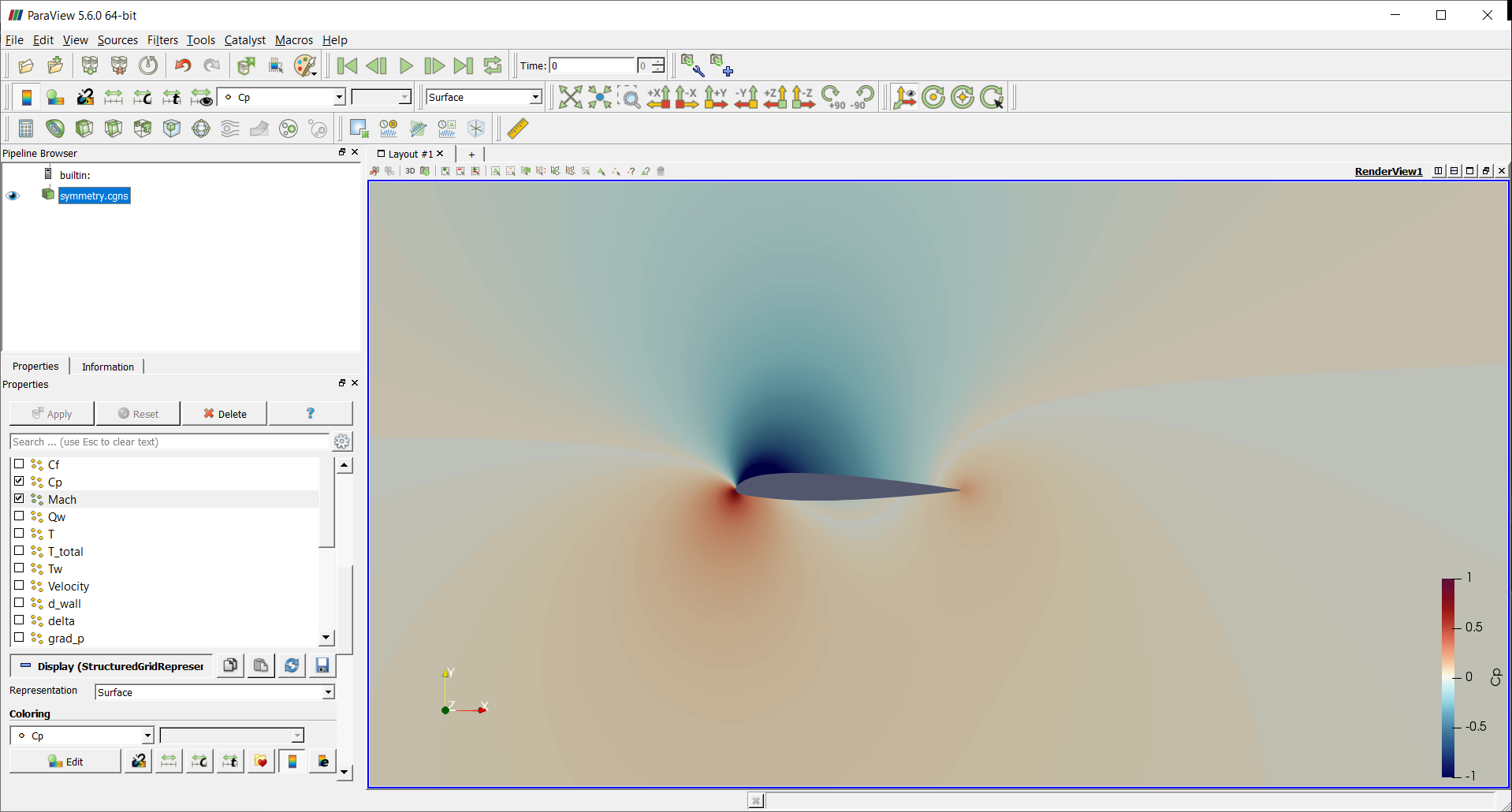


Рисунок 11 — Окно программы ParaView