ՀԱՅԱՍՏԱՆԻ ԳԻՏՈՒԹՅՈՒՆՆԵՐԻ ԱԶԳԱՅԻՆ ԱԿԱԴԵՄԻԱ
НАЦИОНАЛЬНАЯ АКАДЕМИЯ НАУК АРМЕНИИ
NATIONAL ACADEMY OF SCIENCES OF ARMENIA
ДОКЛАДЫ ԶԵԿՈՒՅՑՆԵՐ REPORTS

Հատոր Том Volume 117 2017 № 2

ФИЗИКА

УДК 539.12

Д. А. Осипян, член-корреспондент НАН РА Г. Г. Матевосян, О. С. Ароян

Построение гибридных программных систем PIC-моделирования

(Представлено 10/IV 2017)

Ключевые слова: *метод частиц в ячейке, PIC-моделирование плазмы, гибридное программирование.*

1. Введение. Численное моделирование нестационарных плазменных процессов основано на физико-математических моделях различной степени детальности. В настоящее время считается общепринятым, что хорошей исходной моделью плазмы является система уравнений, состоящая из кинетических уравнений Власова для функций распределения ионов и электронов и уравнений Максвелла с самосогласованными электромагнитными полями. Согласно кинетическим уравнениям функции распределения изменяются под действием электрического и магнитного полей, которые, в свою очередь, определяются из уравнений Максвелла через моменты функции распределения. Таким образом, исходная система уравнений является нелинейной, и для её решения необходимо использование численных методов. Определению численного эксперимента в рамках этой модели наиболее полно удовлетворяет дискретное моделирование плазмы методом частиц в ячейке[1] [1-3].

Общая схема метода такова. Плазма представляется набором достаточно большого числа модельных частиц, траектории которых являются характеристиками уравнения Власова. Движение частиц происходит в расчётной области, в соответствии с законами классической механики в самосогласованном электромагнитном поле, которое определяется из уравнений Максвелла с использованием зарядов и токов в качестве источников. Плотности этих зарядов и токов, в свою очередь, вычисляются по координатам и скоростям частиц с помощью

[1] Метод, разработанный Харлоу [1], был назван Particle-in-Cell. На русский язык часто переводится как «метод частиц в ячейках» и для краткости именуется PIC, а основанные на нём программы принято называть PIC-кодами.

какой-либо процедуры интерполяции массы, заряда и скорости частиц на узлы пространственной сетки, покрывающей расчётную область.

На данный момент существует ряд эффективных реализаций метода частиц в ячейках [4]. Во многих из них в качестве описания начальных данных используется файл, содержащий параметры задачи в виде списка пар «ключ – значение». В таком формате задаются размеры расчётной области, начальная конфигурация электромагнитных полей, заряд и масса частиц, их распределение по координатам и скоростям, граничные условия, параметры алгоритма моделирования и диагностики. Фрагмент типичного файла входных данных выглядит следующим образом:

```
nx = 100          ; размер сетки вдоль оси X
ny = 100          ; размер сетки вдоль оси Y
nz = 150          ; размер сетки вдоль оси Z
mf = D            ; конфигурация магнитного поля.
                  ; D - поле магнитного диполя,
                  ; U - однородное магнитное поле.
m   = 20          ; момент диполя если mf = D, иначе -
                    игнорируется.
mx = 50          ; x-координата магнитного диполя, если
                    mf = D,
                 ; x-компонента магнитного поля, если
mf = U
my = 50          ; y-координата магнитного диполя если
mf = D,
                 ; y-компонента магнитного поля, если
mf = U
mz = 50          ; z-координата магнитного диполя, если
mf = D,
                 ; z-компонента магнитного поля, если
                   mf = U
pq = 1.0         ; заряд иона в единицах заряда протона
pm = 1.0         ; масса иона в единицах массы протона
Np = 1.0e+16     ; количество ионов
```

Форматы файлов входных данных для большинства программ моделирования плазмы существенно сложнее приведённого выше примера. Они существенно отличаются друг от друга, и их описания представляют собой объёмные руководства. Основным недостатком такого описания задачи является ограниченность возможных начальных конфигураций полей и распределения частиц. Если в программе не предусмотрено, например, задание тороидального магнитного поля, то без изменения исходного кода такую конфигурацию добавить невозможно.

Коммерческие пакеты предоставляют определённую гибкость за счёт поддержки дополнительных синтаксических конструкций во входных файлах, однако это усложняет практическое использование программы.

Мы рассмотрим альтернативный способ задания начальных и граничных условий, основанный на использовании интерпретатора скриптового языка. Суть метода состоит в разделении расчётной программы на компи-

лируемый и интерпретируемый модули. В компилируемом модуле реализуются наиболее требовательные к вычислительным ресурсам алгоритмы и структуры данных: массивы для хранения частиц и полей, алгоритм решения уравнения движения частицы, газодинамических уравнений и уравнений Максвелла. Это расчётный модуль. В интерпретируемом модуле находятся функции задания начальных и граничных условий, которые, как правило, не требуют значительных ресурсов. Интерпретируемый модуль, таким образом, играет роль конфигурационного скрипта. Расчётный модуль предоставляет скрипту доступ к алгоритмам и структурам данных через интерпретатор скриптового языка. Это позволяет задавать в скрипте произвольные начальные условия задачи, но его возможности этим не ограничиваются. В конфигурационный скрипт легко добавить, например, управление вычислительным процессом. Подобные вычислительные системы, сочетающие в себе различные программные компоненты, называются гибридными. В следующем параграфе построение такой программы будет продемонстрировано на практическом примере.

2. Взаимодействие программ на языках C++ и Python. Большинство современных пакетов PIC-кодов реализовано на языке C++. Для научных вычислений широко используется также язык Python, для которого имеются бесплатно доступные интерпретаторы и обширная библиотека программ. В стандартный пакет Python входит также Python/C API [5] – множество подпрограмм на языке C для работы с интерпретатором языка Python, что позволяет использовать его в качестве интерпретатора конфигурационного скрипта в C++ программах. Существует два механизма взаимодействия C++ и Python: расширение и встраивание. В первом случае из Python вызываются элементы C++ программы. Для этого реализуется модуль расширения для Python, в который экспортируются элементы C++ программы. Этот модуль можно импортировать в Python с помощью стандартного оператора import. В пользовательской системе необходимо установить пакет Python и предусмотреть процедуру инсталляции модуля расширения. В случае встраивания в C++ программе создаётся интерпретатор Python, на выполнение которому передаётся скрипт. Интерпретатор является частью C++ программы, и устанавливать Python на компьютере пользователя не обязательно.

Для вычислительных задач типа метода частиц в ячейке более эффективным является комбинирование этих механизмов. В этом случае расчётный модуль предварительно создаёт модуль расширения Python, затем с помощью интерпретатора выполняет скрипт, использующий этот модуль. После выполнения скрипта, когда все начальные и граничные условия заданы, расчётный модуль приступает к выполнению алгоритма моделирования. В процессе работы расчётный модуль может обращаться к отдельным функциям конфигурационного скрипта. Например, после каждой итерации расчётный модуль может вызывать определённую функцию скрипта для инжекции частиц в расчётную область, вывода промежуточных данных, графиков и дополнительной диагностики, что существенно расширяет возможности программы. В зависимости от характера числен-

ного эксперимента в конфигурационном скрипте могут быть реализованы процедуры мониторинга вычислительного процесса, фильтрации шумов, обработки ошибок. Тогда, скрипт будет играть роль драйвера вычислительного процесса.

3. Реализация на языках C++ и Python. Продемонстрируем комбинирование встраивания интерпретатора языка Python и расширения на простом примере. Для связывания C++ и Python используем библиотеку Boost.Python [6], которая является обёрткой функций Python/C API, предназначенных для создания модулей расширения, инициализации интерпретатора и выполнения Python программ. В листинге 1 показаны основные этапы экспорта C++ структур в Python, что соответствует созданию модуля расширения Python, и выполнения скрипта.

Макрос `BOOST_PYTHON_MODULE` создаёт модуль расширения Python, содержащий структуры `Vector` и `Particle`. Класс `boost::python::class_` и, в частности, его функция-член `def()` обеспечивают экспорт этих типов данных с помощью Python/C API. В строке 22 экспортируется массив частиц, который будет доступен в скрипте как переменная `parts`. На этом завершается создание модуля расширения, который, в соответствии с соглашением имён в Python [7], мы назвали `pic`. Аналогичным образом необходимо экспортировать расчётную сетку и другие объекты, настраиваемые для конкретной задачи.

```
1. #include <boost/python.hpp>
2. #include <vector>
3. struct Vector {
4.     double x;
5.     double y;
6.     double z;
7. }
8. struct Particle {
9. Vector r; // particle position
10.  Vector v; // particle velocity
11.  double q; // particle charge
12.  double m; // particle mass
13.  };
14.  typedef std::vector<Particle> Particles;
15.  Particles particles;
16.  namespace py = boost::python;
17.  BOOST_PYTHON_MODULE(pic) {
18.  py::class_<Vector>("Vector")
               .def_readwrite("x", &Vector::x)
               .def_readwrite("y", &Vector::y)
               .def_readwrite("z", &Vector::z),
19.  py::class_<Particle>("Particle")
               .def_readwrite("r", &Particle::r)
               .def_readwrite("v", &Particle::v)
               .def_readwrite("q", &Particle::q),
```

```
            .def_readwrite("m", &Particle::m),
20.  py::class_<Particles>("Particles")
21.  .def(py::vector_indexing_suite<Particles>());
22.  py::import("__main__")
23.  .attr("parts") = py::ptr(&particles);
24.   }
25.   int main() {
26.  PyImport_AppendInittab("pic", initpic);
27.  Py_Initialize();
28.  py::object main_module = py::import("__main__");
29.  py::dict
30.  main_namespace = py::extract<py::dict>(
31.  main_module.attr("__dict__"));
32.  py::exec_file("init_pic.py", main_namespace,
33.  main_namespace);
34.  std::cout << particles.size()
35.  << " particle(s) created initially."
36.  << std::endl;
37.  // На очередном шаге алгоритма моделирования
38.  // производится инжекция новых частиц.
39.  py::object
40.  inject_parts = main_namespace["inject_parts"];
41.  size_t
42.  parts_num = py::extract<double>(inject_parts());
43.  std::cout << parts_num
44.  << " particle(s) after injection."
45.            << std::endl;
46.  }
```

Листинг 1. В функции main производится встраивание интерпретатора Python. Для этого модуль pic добавляется во множество импортируемых модулей, затем инициализируется интерпретатор Python и выполняется конфигурационный скрипт init_pic.py:

```
1.    from pic import *
2.    p = Particle()
3.    p.r.x = p.r.y = p.r.z = 0
4.    parts[:] = [p, p, p]
5.    def inject_parts():
6.    p = Particle()
7.        p.r.x = p.r.y = p.r.z = 1
8.        for i in range(0, 3):
          i.  parts.append(p)
9.        return len(parts)
```

Листинг 2. На этом шаге, в результате выполнения строк 1 – 4 конфигурационного скрипта (листинг 2), массив частиц (parts) содержит три элемента. Это соответствует заданию начального распределения частиц по координатам и скоростям. Затем расчётная программа вызывает функцию

inject_parts() для модификации массива частиц, в данном случае – для добавления новых частиц в область моделирования.

Для обеспечения максимальной гибкости расчётная программа может вызывать функции конфигурационного скрипта перед выполнением и после завершения этапов моделирования. Стандартный набор вызываемых предопределённых функций может выглядеть так:

```
on_iteration_begin()     -- перед очередной итерацией алгоритма
on_iteration_end()       -- после очередной итерации алгоритма
on_boundary_condition()-- для применения граничных условий
on_particles_moved()     -- после продвижения частиц на очередной
                            шаг по времени
```

В этих функциях удобно реализовать процедуры диагностики, сохранения данных в файлах, инжекции новых частиц, обновления переменных электромагнитных полей. Существует богатый выбор модулей расширений Python, предназначенных для анализа и визуализации данных, которые могут быть использованы в конфигурационном скрипте для создания полноценного инструмента моделирования на базе расчётного модуля. Библиотека Boost.Python позволяет легко добавить поддержку Python в имеющиеся расчётные программы. Возможно создание унифицированного интерфейса с Python для различных программ моделирования, что упростило бы их использование.

4. Заключение. Преимуществом описанной гибридной технологии по сравнению с использованием традиционного файла входных данных является возможность задания различных начальных и граничных условий, управления ходом численного эксперимента без внесения изменений в расчётный модуль. Выбор Python в качестве драйвера позволяет использовать имеющиеся для него библиотеки интерактивной визуализации, анализа и обработки результатов моделирования. Подобная техника реализована в разработанной в ИРФЭ НАН РА программе OpenPIC [8], предназначенной для исследования широкого класса нестационарных задач динамики бесстолкновительной плазмы.Предложенный механизм может быть использован также в других системах программного обеспечения, требующих мониторинга потока управления.

Исследование выполнено при финансовой поддержке Государственного комитета по науке МОН РА в рамках научного проекта № 15T-1C231.

Институт радиофизики и электроники НАН РА
e-mail: sd_fund@sci.am

Д. А. Осипян, член-корреспондент НАН РА Г. Г. Матевосян, О. С. Ароян

Построение гибридных программных систем PIC-моделирования

Предложен способ разделения программы PIC-моделирования на компилируемый расчётный модуль и интерпретируемый конфигурационный скрипт, который задаёт начальные и граничные условия, а также взаимодействует с расчётным модулем на определённых этапах численного эксперимента. Показано преимущество данного подхода по сравнению с использованием традиционного файла входных данных. Приведён пример реализации описанной технологии на языках C++ и Python.

Դ. Ա. Օսիպյան, ՀՀ ԳԱԱ թղթակից անդամ Հ. Հ. Մաթևոսյան, Հ. Ս. Արոյան

Հիբրիդային PIC-մոդելավորման ծրագրային համակարգերի կառուցում

Ներկայացված է PIC-մոդելավորման ծրագրային համակարգերի տարրալուծման եղանակ՝ հաշվարկային մոդուլի և ինտերպրետացվող կոնֆիգուրացիոն սկրիպտի, որը նկարագրում է նախնական և սահմանային պայմանները։ Սկրիպտը նաև համագործակցում է հաշվարկային մոդուլի հետ թվային էքսպերիմենտի որոշակի փուլերում։ Ցույց է տրված առաջարկված մոտեցման առավելությունն ավանդական մուտքագրման ֆայլի համեմատ։ Բերված է C++ և Python լեզուների միջոցով նկարագրված տեխնոլոգիայի ծրագրային իրականացման օրինակ։

D. A. Osipyan, corresponding member of NAS RA H. H. Matevosyan, H. S. Aroyan

Building Hybrid PIC-Simulation Codes

The decomposition method of PIC simulation codes into the compiled simulation module and the interpreted configuration script which sets initial and boundary conditions is presented. The script is also intended to interact with simulation module at certain stages of a numerical experiment. The benefit of this approach in comparison with the traditional input file usage is shown. The example of the described technique implementation in C ++ and Python is given.

Литература

1. *Harlow F. H.* A Machine Calculation Method for Hydrodynamic Problems, Los Alamos Scientific Laboratory report LAMS-1956, 1955.
2. *Хокни Р., Иствуд Дж.* Численное моделирование методом частиц. М. Мир,1987. 638 с.
3. *Birdsall C. K., Langdon A. B.* Plasma Physics via Computer Simulation. CRC Press. 2004. 504 p.

4. *Smith J.,* Advanced simulation tools for next generation particle acceleration: PIC and beyond. Advanced school on laser applications and accelerators, 2014.
5. Python/C API Reference Manual, https://docs.python.org/2/c-api
6. *Abrahams D., Grosse-Kunstleve Ralf W.* -, C/C++ Users Journal, 2003, July.
7. PEP 8, Style Guide for Python Code, https://www.python.org/dev/peps/pep-0008.
8. *Nersisyan H. B., Sargsyan K. A., Osipyan D. A., Sargsyan M. V., Matevosyan H. H.*- Armenian Journal of Physics. 2011. N 4 V. 2. P. 74-89.