Второй слёт разработчиков CFD кодов
«Отечественные CFD коды – 2015»
Москва, 28-29 ноября 2015 г.

Краснов М.М.
ИПМ им. М.В. Келдыша РАН

Операторный метод программирования для задач математической физики

Операторы в математике и физике

- уравнение сохранения количества движения для ньютоновской несжимаемой жидкости без действия массовых сил:

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) - \nabla \cdot (\mu \nabla \boldsymbol{U}) = -\nabla p$$

- уравнение Эйлера в гидродинамике идеальной жидкости в консервативной векторной форме:

$$\partial \mathbf{m}/\partial t + \partial \mathbf{f}_x/\partial x + \partial \mathbf{f}_y/\partial y + \partial \mathbf{f}_z/\partial z = 0$$

- $\mathbf{E} = -\operatorname{grad} \phi$
- $\rho = \operatorname{div} \mathbf{D}$
- $\operatorname{rot} \mathbf{E} = -\partial \mathbf{B}/\partial t$

Примеры использования библиотеки

- E=-grad_(fi);
- ro=div_(D);
- dBdt=-rot_(E);
- v=(Sp*(I-P*AH*R*Ah)*Sp)(u);
- u=laplas(mu*v); (u=Δ(μv))

Сравнение операторов в математике и в программах

Операторы в математике

- $\mathbf{E}$=-grad ϕ
- ρ=div $\mathbf{D}$
- rot $\mathbf{E}$ = -$\partial\mathbf{B}/\partial$t
- $Q = S_p(I - PA_H^{-1}RA_h)\,S_p$

Примеры программ

- E=-grad_(fi);
- ro=div_(D);
- dBdt=-rot_(E);
- v=(Sp*(I-P*AH*R*Ah)*Sp)(u);

Цели, ставившиеся при разработке метода

- Приближение внешнего вида программы к формулам в теоретических работах
- Ленивые вычисления – правая часть вычисляется только при присваивании
- Сокрытие реализации оператора присваивания.
- Максимальная производительность вычислений

Особенности платформы CUDA

- Данные плотных сеточных функций хранятся в памяти CUDA.
- Основной оператор библиотеки – оператор присваивания сеточного вычислителя плотной сеточной функции выполняется путём вызова одного ядра.
- Возможно выполнение нескольких операторов присваивания в одном ядре.

Поддержка Intel Xeon Phi

Специальной поддержки (в отличие от CUDA) Xeon Phi в библиотеке нет. Есть поддержка OpenMP для последовательного кода, реализованного через три вложенных цикла. Перед этими циклами стоит оператор:

```
#pragma omp parallel for private(i,j,k)
```

Если компилятор поддерживает данную прагму, то для исполнения оператора будут использованы имеющиеся ядра процессора.

Пример оператора (Лапласа)

```
template<typename T> struct laplas : grid_operator<laplas<T> >
{
  laplas(T dx,T dy,T dz) : dx2(dx*dx), dy2(dy*dy), dz2(dz*dz){}
  template<class EOP>
  T operator()(size_t i,size_t j,size_t k,const EOP& eop)const{
    T v0 = eop(i, j, k) * 2.;
    return
      (eop(i + 1, j, k) + eop(i - 1, j, k) - v0) / dx2 +
      (eop(i, j + 1, k) + eop(i, j - 1, k) - v0) / dy2 +
      (eop(i, j, k + 1) + eop(i, j, k - 1) - v0) / dz2;
  }
private: T dx2, dy2, dz2;
};
```

Использованные технологии

- Шаблонный полиморфизм;
- Метавычисления (шаблонное метапрограммирование);
- Поддержка размерных величин;
- Объекты-заместители;
- Контекст исполнения.

Вычисление объёмных интегралов

$$I_{kj}^{1ev} = \int_{T_k} \left[(\rho u) \frac{\partial \varphi_j}{\partial x} + (\rho v) \frac{\partial \varphi_j}{\partial y} + (\rho w) \frac{\partial \varphi_j}{\partial z} \right] dV$$

$$I_{kj}^{2ev} = \int_{T_k} \left[(\rho u^2 + p) \frac{\partial \varphi_j}{\partial x} + (\rho uv) \frac{\partial \varphi_j}{\partial y} + (\rho uw) \frac{\partial \varphi_j}{\partial z} \right] dV$$

$$I_{kj}^{3ev} = \int_{T_k} \left[(\rho vu) \frac{\partial \varphi_j}{\partial x} + (\rho v^2 + p) \frac{\partial \varphi_j}{\partial y} + (\rho vw) \frac{\partial \varphi_j}{\partial z} \right] dV$$

$$I_{kj}^{4ev} = \int_{T_k} \left[(\rho wu) \frac{\partial \varphi_j}{\partial x} + (\rho wv) \frac{\partial \varphi_j}{\partial y} + (\rho w^2 + p) \frac{\partial \varphi_j}{\partial z} \right] dV$$

$$I_{kj}^{5ev} = \int_{T_k} \left[(E + p)u \frac{\partial \varphi_j}{\partial x} + (E + p)v \frac{\partial \varphi_j}{\partial y} + (E + p)w \frac{\partial \varphi_j}{\partial z} \right] dV$$

$$I_{kj}^{nev} = \int_{T_k} \left[f_x^n \frac{\partial \varphi_j}{\partial x} + f_y^n \frac{\partial \varphi_j}{\partial y} + f_z^n \frac{\partial \varphi_j}{\partial z} \right] dV$$

где $f^n = f^n(\rho, u, v, w, p, E)$, ϕ_j – базисные функции.

Интеграл по тетраэдру вычисляется методом Гаусса, для чего нужно найти значение выражения в квадратных скобках в Гауссовых точках. На входе в процедуру имеются коэффициенты разложения по базисным функциям величин ρ, ρu, ρv, ρw, E.

В начале по этим коэффициентам находятся сами величины в Гауссовых точках, затем вычисляются значения переменных $u=\rho u/\rho$, $v=\rho v/\rho$, $w=\rho w/\rho$. После этого вычисляется значение переменной e: $e=E/\rho-(u^2+v^2+w^2)/2$. Давление p вычисляется по уравнению состояния $p=p(\rho, e)$.

В результате основная процедура метода Галёркрина выглядит следующим образом:

```
R = A * (join_polynom_operator<grid_type>(ugrid)(
  volume_integral<integrate_tetra_t>(ugrid, eos,
    _rou, _rov, _row)(U),
  volume_integral<integrate_tetra_t>(ugrid, eos,
    _rou*_u+_p, _rou*_v, _rou*_w)(U),
  volume_integral<integrate_tetra_t>(ugrid, eos,
    _rov*_u, _rov*_v+_p, _rov*_w)(U),
  volume_integral<integrate_tetra_t>(ugrid, eos,
    _row*_u, _row*_v, _row*_w+_p)(U),
  volume_integral<integrate_tetra_t>(ugrid, eos,
    (_E+_p)*_u, (_E+_p)*_v, (_E+_p)*_w)(U)
) - hflow_op(U));
```

Преимущества операторного метода программирования

- Наглядность. Из короткой записи сразу видно, что делается для каждой ячейки;
- Не нужны промежуточные массивы (сеточные функции). Легко выполнять выражения любой сложности;
- Автоматический (простой перекомпиляцией) перенос на параллельные архитектуры, такие, как NVidia CUDA и Intel Xeon Phi.

Спасибо за внимание