Staggerd Mesh
elastic wave equations solver as an example

Takayuki Muranushi

Kyoto University → AICS Exascale project

May 28, 2014

1. Introduction

Section 1

Introduction

Mission

1. Read and udnerstand seismic wave-tsunami simulation programs currently used on the K-computer.
2. Translate the Fortran programs for the proposed exascale hardware prototypes.
3. Use automated code generation and optimization instead of manually doing so.
4. Convince funding agencies that performance figures for such exotic parallel hardwares are not just imaginary but are feasible and have useful applications.
5. Construct tsunami early warning system that will be run immediately after the earthquake, and provide local evacuation information before the tsunami arrives, and save souls.

Elastic wave equations

The evolution equations are just two lines:

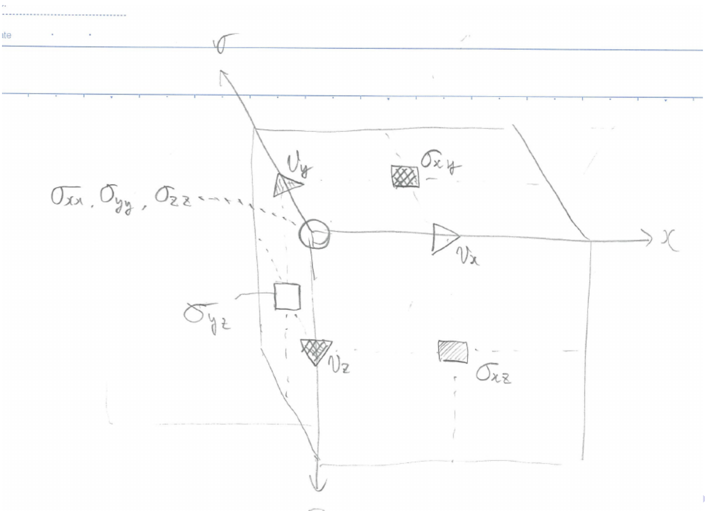
$$\frac{\partial v_i}{\partial t} = \frac{1}{\rho}\partial_j \sigma_{ij} \tag{1}$$

$$\frac{\partial \sigma_{ij}}{\partial t} = \mu(\partial_j v_i + \partial_i v_j) + \lambda \delta_{ij} \partial_k v_k \tag{2}$$

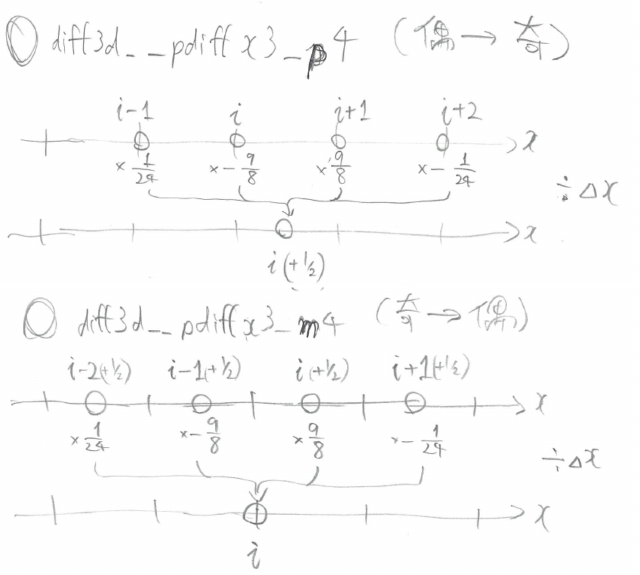
σ_{ij} : stress tensor (3x3 symmetric tensor / 6 independent components)
v_i : velocity (3-component vector)
The above information, plus that the spatial derivatives ∂_i is of 4th order, and that the time derivatives $\frac{\partial}{\partial t}$ are staggered-timestep 2nd order, is sufficient information for a human expert to specify the algorithm, in theory.

Staggered Mesh Structure

Each component of the variable reside in different possition, half-integer shifted in the unit lattice.



The 4th order space differentiation



Many instances of the single differential operator!

```
void diffx3_m4(double f[NZ][NY][NX], double df_dx[NZ][NY][NX]){
  for (int k = 0; k < NZ; ++k)
    for (int j = 0; j < NY; ++j)
      for (int i = 2; i < NX-1; ++i)
        df_dx[k][j][i]
          = ( (f[k][j][i  ] - f[k][j][i-1] ) * r40
            - (f[k][j][i+1] - f[k][j][i-2] ) * r41) / Dx;

void diffx3_p4(double f[NZ][NY][NX], double df_dx[NZ][NY][NX]){
  for (int k = 0; k < NZ; ++k)
    for (int j = 0; j < NY; ++j)
      for (int i = 1; i < NX-2; ++i)
        df_dx[k][j][i]
          = ( (f[k][j][i+1] - f[k][j][i  ] ) * r40
            - (f[k][j][i+2] - f[k][j][i-1] ) * r41) / Dx;
```

```
void diffy3_m4(double f[NZ][NY][NX], double df_dy[NZ][NY][NX]){
    for (int k = 0; k < NZ; ++k) {
      for (int j = 2; j < NY-1; ++j) {
        for (int i = 0; i < NX; ++i) {
   df_dy[k][j][i]
     = ( (f[k][j  ][i] - f[k][j-1][i] ) * r40
       - (f[k][j+1][i] - f[k][j-2][i] ) * r41) / Dy;

void diffy3_p4(double f[NZ][NY][NX], double df_dy[NZ][NY][NX]){
    for (int k = 0; k < NZ; ++k) {
      for (int j = 1; j < NY-2; ++j) {
        for (int i = 0; i < NX; ++i) {
   df_dy[k][j][i]
     = ( (f[k][j+1][i] - f[k][j  ][i] ) * r40
       - (f[k][j+2][i] - f[k][j-1][i] ) * r41) / Dy;
```

plus/minus

$$\mathrm{diff3d}\_\_\begin{Bmatrix}p\\ s\end{Bmatrix}\mathrm{diff}\begin{Bmatrix}x\\ y\\ z\end{Bmatrix}3\_\begin{Bmatrix}p\\ m\end{Bmatrix}4$$

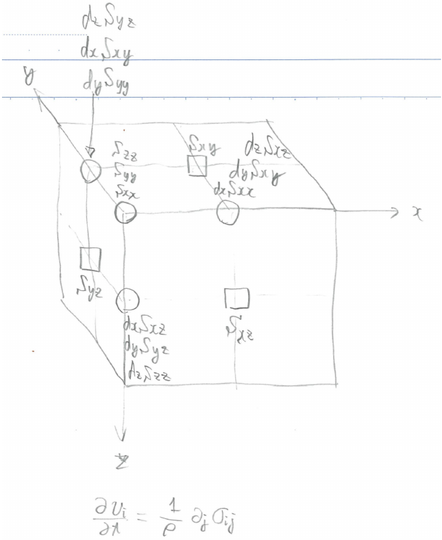
p：主に用いられる.
s：z方向成分のみ存在. 端点で何次の微分かの固定

The code for space differentiation of the stress tensor

The positions where the differentiation result is generated and where needed, coincide on the half-integer lattice.

```
subroutine kernel__stressderiv()
  call diff3d__pdiffx3_p4( Sxx, dxSxx )
  call diff3d__pdiffy3_p4( Syy, dySyy )
  call diff3d__pdiffx3_m4( Sxy, dxSxy )
  call diff3d__pdiffx3_m4( Sxz, dxSxz )
  call diff3d__pdiffy3_m4( Sxy, dySxy )
  call diff3d__pdiffy3_m4( Syz, dySyz )
  call diff3d__sdiffz3_p4( Szz, dzSzz )
  call diff3d__sdiffz3_m4( Sxz, dzSxz )
  call diff3d__sdiffz3_m4( Syz, dzSyz )
end subroutine kernel__stressderiv
```

$$\frac{\partial v_i}{\partial t} = \frac{1}{\rho}\partial_j \sigma_{ij}$$

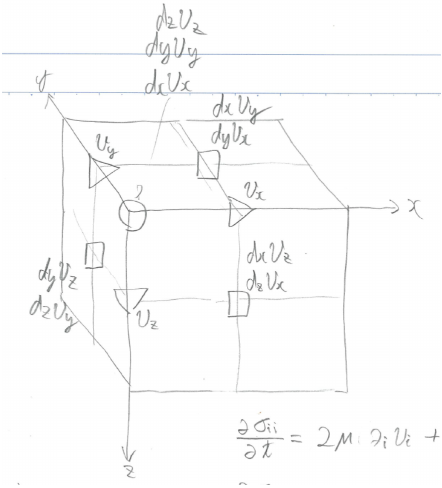


The code for space differentiation of the velocity

The positions where the differentiation result is generated and where needed, coincide on the half-integer lattice.

```
subroutine kernel__velderiv()
  call diff3d__pdiffx3_m4( Vx, dxVx )
  call diff3d__pdiffy3_p4( Vx, dyVx )
  call diff3d__sdiffz3_p4( Vx, dzVx )
  call diff3d__pdiffx3_p4( Vy, dxVy )
  call diff3d__pdiffy3_m4( Vy, dyVy )
  call diff3d__sdiffz3_p4( Vy, dzVy )
  call diff3d__pdiffx3_p4( Vz, dxVz )
  call diff3d__pdiffy3_p4( Vz, dyVz )
  call diff3d__sdiffz3_m4( Vz, dzVz )
end subroutine kernel__velderiv
```

$$\frac{\partial \sigma_{ij}}{\partial t} = \mu(\partial_j v_i + \partial_i v_j) + \lambda \delta_{ij} \partial_k v_k$$



Challenges

- Given that the original equations was just two lines, we should be able to generate the Fortran code from much compact algorithm descriptions. Can we?
- Optionally, prove that symmetries of the equations are retained in the discretized code. e.g. Result is invariant over swap of x and y axes; swap of x axis to $-x$.

Problem source codes are in `https://github.com/StagedHPC/shonan-challenge/tree/master/problems/staggered-mesh`

Type system can do that

You can imagine type system handling this problem, but I feel staging can do better!

```
newtype Even = Even Int
newtype Odd  = Odd  Int

succHalf (Even n) = Odd n; succHalf (Odd n) = Even (n+1)
predHalf (Even n) = Odd (n-1); predHalf (Odd n) = Even n

type Array100 = Array3d Odd Even Even
type Array010 = Array3d Even Odd Even
type Array001 = Array3d Even Even Odd

type family MkArr (n :: Nat) :: * where
  MkArr 0 = Array100 Double
  MkArr 1 = Array010 Double
  MkArr 2 = Array001 Double

data Vector3 = Vector3 (f :: Nat -> *) :: *
type VelocityField = Vector3 MkArr
```