



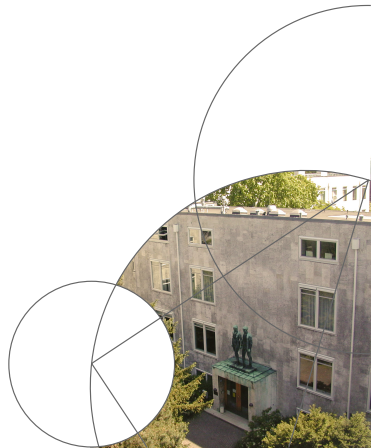
Faculty of Science



PMPH Project

Daniel Egeberg Sebastian Paaske Tørholm
Department of Computer Science

November 6, 2014
Slide 1/14



After time loop extraction

- Time loop has been moved out, what do our variables look like?

```
REAL myX[outer][numX];  
REAL myY[outer][numY];  
REAL myTimeline[outer][numT];  
REAL myResult[outer][numX][numY];  
REAL myVarX[outer][numX][numY];  
REAL myVarY[outer][numX][numY];  
REAL myDxx[outer][numX][4];  
REAL myDyy[outer][numY][4];
```

- Do we need the outer dimension on all of them?



How to determine dependencies

- Need to determine which variables are dependent on the outer dimension, and which that are independent.



How to determine dependencies

- Need to determine which variables are dependent on the outer dimension, and which that are independent.
- To accomplish this: Look at when each variable is read from or written to.



initGrid

```
void initGrid(...) {  
    for(unsigned i=0;i<numT;++i)  
        myTimeline[o][i] = t*i/(numT-1);  
    // ...  
    for(unsigned i=0;i<numX;++i)  
        myX[o][i] = i*dx - myXindex*dx + s0;  
    // ...  
    for(unsigned i=0;i<numY;++i)  
        myY[o][i] = i*dy - myYindex*dy + logAlpha;  
}
```

- Writes to myTimeline, myX, myY.
- Data written is independent of outer dimension.
- myTimeline, myX, myY never written to outside of initGrid.



initOperator

```
void initOperator(...) {  
    // ...  
    for(unsigned i=1;i<n-1;i++) {  
        dxl      = x[o][i]   - x[o][i-1];  
        dxu      = x[o][i+1] - x[o][i];  
        Dxx[o][i][0] = 2.0/dxl/(dxl+dxu);  
        Dxx[o][i][1] = -2.0*(1.0/dxl + 1.0/dxu)/(dxl+dxu);  
        Dxx[o][i][2] = 2.0/dxu/(dxl+dxu);  
        Dxx[o][i][3] = 0.0;  
    }  
    // ...  
}
```

- Reads from myX, writes to myDxx. (Ditto y.)
- Data written is independent of outer dimension. (If myX is.)
- myDxx never written to outside of initGrid.



setPayoff

```
void setPayoff(...) {  
    for (unsigned i=0; i < numX; ++i) {  
        REAL payoff = max(myX[o][i]-strike, (REAL)0.0);  
        for (unsigned j=0; j < numY; ++j)  
            myResult[o][i][j] = payoff;  
    }  
}
```

- Reads from myX, writes to myResult.
- Data written is dependent on outer dimension.
 - strike is a function of o!



updateParams

```
void updateParams(...) {  
    for (unsigned i=0; i < numX; ++i)  
        for (unsigned j=0; j < numY; ++j) {  
            myVarX[o][i][j] = f(myX[o][i], myY[o][j], myTimeline←  
                                [o][g]);  
            myVarY[o][i][j] = g(myX[o][i], myY[o][j], myTimeline←  
                                [o][g]);  
        }  
}
```

- Reads from myX, myY, myTimeline, writes to myVarX, myVarY.
- Data written is not dependent on outer dimension.¹
- myVarX and myVarY never written to outside of initGrid.

¹The variables used are independent of it, per our previous slides.



rollback

- Partitioned into four logical parts: Explicit X, explicit Y, implicit X, implicit Y.
- We analyse each of these separately.



rollback - Explicit X

```
REAL dtInv = 1.0/(myTimeline[o][g+1]-myTimeline[o][g]);  
// ...  
for(i=0;i<numX;i++) {  
  for(j=0;j<numY;j++) {  
    u[j][i] = dtInv*myResult[o][i][j];  
    if(i > 0) {  
      u[j][i] += f(myVarX[o][i][j],myDxx[o][i][0],myResult[o][i-1][j]);  
    }  
    u[j][i] += g(myVarX[o][i][j],myDxx[o][i][1],myResult[o][i][j]);  
    if(i < numX-1) {  
      u[j][i] += h(myVarX[o][i][j],myDxx[o][i][2],myResult[o][i+1][j]);  
    }  
  }  
}
```

- Reads from a number of globs, but doesn't write to any!



rollback - Explicit Y

```
for(j=0;j<numY;j++) {  
  for(i=0;i<numX;i++) {  
    v[i][j] = 0.0;  
    if(j > 0) {  
      v[i][j] += f(myVarY[o][i][j],myDyy[o][j][0],myResult[o][i][j-1]);  
    }  
    v[i][j] += g(myVarY[o][i][j],myDyy[o][j][1],myResult[o][i][j]);  
    if(j < numY-1) {  
      v[i][j] += h(myVarY[o][i][j],myDyy[o][j][2],myResult[o][i][j+1]);  
    }  
    u[j][i] += v[i][j];  
  }  
}
```

- Reads from a number of globs, but doesn't write to any!



rollback - implicit X

```
REAL dtInv = 1.0/(myTimeline[o][g+1]-myTimeline[o][g]);  
// ...  
for(j=0;j<numY;j++) {  
  for(i=0;i<numX;i++) {  
    a[i] =      - 0.5*(0.5*myVarX[o][i][j]*myDxx[o][i][0]);  
    b[i] = dtInv - 0.5*(0.5*myVarX[o][i][j]*myDxx[o][i][1]);  
    c[i] =      - 0.5*(0.5*myVarX[o][i][j]*myDxx[o][i][2]);  
  }  
  tridag(a,b,c,u[j],numX,u[j],yy);  
}
```

- Reads from a number of globs, but doesn't write to any!



rollback - implicit Y

```
REAL dtInv = 1.0/(myTimeline[o][g+1]-myTimeline[o][g]);  
// ...  
for(i=0;i<numX;i++) {  
  for(j=0;j<numY;j++) {  
    a[j] = - 0.5*(0.5*myVarY[o][i][j]*myDyy[o][j][0]);  
    b[j] = dtInv - 0.5*(0.5*myVarY[o][i][j]*myDyy[o][j][1]);  
    c[j] = - 0.5*(0.5*myVarY[o][i][j]*myDyy[o][j][2]);  
    y[j] = dtInv*u[j][i] - 0.5*v[i][j];  
  }  
  tridag(a,b,c,y,numY,globs.myResult[i],yy);  
}
```

- Reads from a number of globs, writes to myResult.



Overview of dependencies

- We can create a table of these dependencies:

Function	myX	myY	myTimeline	myResult	myVarX	myVarY	myDxx	myDyy
initGrid	W	W	W					
initOperator, x	R						W	
initOperator, y		R						W
setPayoff	R			W				
updateParams	R	R	R		W	W		
rollback, explicit x				R	R		R	
rollback, explicit y				R		R		R
rollback, implicit x					R		R	
rollback, implicit y				W		R		R

- Only `myResult` actually needs to be computed separately for each outer iteration.



After reduction of duplicate computations

- After this reduction, our variables look as follows:

```
REAL myX[numX];  
REAL myY[numY];  
REAL myTimeline[numT];  
REAL myResult[outer][numX][numY];  
REAL myVarX[numX][numY];  
REAL myVarY[numX][numY];  
REAL myDxx[numX][4];  
REAL myDyy[numY][4];
```

