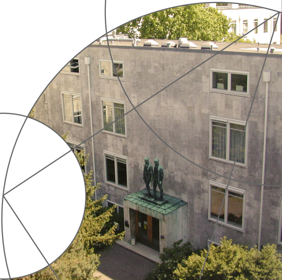
PMPH project

Malte Stær Nissen
René Løwe Jacobsen
Erik John Partridge
Department of Computer Science



Agenda

1. Introduction and OpenMP
2. Naïve CUDA
3. Optimized CUDA
4. Results

Agenda

1. Introduction and OpenMP
2. Naïve CUDA
3. Optimized CUDA
4. Results

Introduction and OpenMP

High level code structure:

```
for each i in outer {    // parallel loop
    initGlob
    initOperator dxx
    initOperator dyy
    setPayoff

    for each t in time { // sequential loop
        updateParams
        explicitX
        explicitY
        implicitX
        tridag
        implicitY
        tridag
    }
}
```

Agenda

1. Introduction and OpenMP
2. Naïve CUDA
3. Optimized CUDA
4. Results

Loop distribution

```
for each i in outer {     // parallel loop
    initGlob
    initOperator dxx
    initOperator dyy
    setPayoff

    for each t in time { // sequential loop
        updateParams
        explicitX
        explicitY
        implicitX
        tridag
        implicitY
        tridag
    }
}
```

Loop distribution

```
deviceInitGlob
deviceInitOperator dxx
deviceInitOperator dyy
deviceSetPayoff

for each i in outer {     // parallel loop
    for each t in time { // sequential loop
        updateParams
        explicitX
        explicitY
        implicitX
        tridag
        implicitY
        tridag
    }
}
```

Loop interchange

```
deviceInitGlob
deviceInitOperator dxx
deviceInitOperator dyy
deviceSetPayoff

for each t in time {        // sequential loop
    for each i in outer {   // parallel loop
        updateParams
        explicitX
        explicitY
        implicitX
        tridag
        implicitY
        tridag
    }
}
```

Loop distribution

```
deviceInitGlob
deviceInitOperator dxx
deviceInitOperator dyy
deviceSetPayoff

for each t in time {        // sequential loop
    deviceUpdateParams
    deviceExplicitX
    deviceExplicitY
    deviceImplicitX
    tridag_solver
    deviceImplicitY
    tridag_solver
}
```

Agenda

1. Introduction and OpenMP
2. Naïve CUDA
3. Optimized CUDA
4. Results

Memory coalescing

```
deviceExplicitX // computes the transpose of u
deviceExplicitY // writes directly to the transpose of u
deviceImplicitX // computes the transpose of a, b and c
transpose u, a, b and c
tridag_solver
transpose u
deviceImplicitY // reads from the transpose of u
tridag_solver
```

Other optimizations

- Re-usage of shared memory (tridag)
- Reduction of global memory accesses
- Reduction of floating point operations

ExplicitX naïve

```
// u[outer][numY][numX]
int uindex = i*numY*numX + k*numX + j;
// myVarX [outer][numX][numY]
int myVarXindex = i*numX*numY + j * numY + k;
// myResult[outer][numX][numY]
u[uindex] = dtInv * myResult[myVarXindex];

// Dxx [outer][numX][4]
int Dxxindex = i*numX*4 + j*4;
REAL varX = myVarX[myVarXindex];
if (j > 0) {
    u[uindex] +=  0.5*(0.5*varX*myDxx[Dxxindex])
                    * myResult[i*numX*numY + (j-1)*numY + k];
}
u[uindex] +=      0.5*(0.5*varX*myDxx[Dxxindex+1])
                    * myResult[myVarXindex];
if (j < numX) {
    u[uindex] +=  0.5*(0.5*varX*myDxx[Dxxindex+2])
                    * myResult[i*numX*numY + (j+1)*numY + k];
}
```

ExplicitX coalesced

```
// myVarX [outer][numX][numY]
int idx = i*numX*numY + j * numY + k;
// myResult[outer][numX][numY]
u[idx] = dtInv * myResult[idx];

// Dxx [outer][numX][4]
int Dxxindex = i*numX*4 + j*4;
REAL varX = myVarX[idx];
if (j > 0) {
    u[idx] +=  0.5*(0.5*varX*myDxx[Dxxindex])
                     * myResult[i*numX*numY + (j-1)*numY + k];
}
u[idx] +=      0.5*(0.5*varX*myDxx[Dxxindex+1])
                     * myResult[idx];
if (j < numX) {
    u[idx] +=  0.5*(0.5*varX*myDxx[Dxxindex+2])
                     * myResult[i*numX*numY + (j+1)*numY + k];
}
```

ExplicitX optimized

```
// u[outer][numX][numY]
int idxO = i*numX*numY;
int idx = idxO + j*numY + k;
// myResult[outer][numX][numY]
REAL uval;
uval = dtInv * myResult[idx];

// Dxx [outer][numX][4]
int Dxxindex = i*numX*4 + j*4;
REAL varX = 0.25*myVarX[idx];
if (j > 0) {
    uval +=   (varX*myDxx[Dxxindex])
                    * myResult[idxO + (j-1)*numY + k];
}
uval +=       (varX*myDxx[Dxxindex+1])
                     * myResult[idx];
if (j < numX) {
    uval +=   (varX*myDxx[Dxxindex+2])
                     * myResult[idxO + (j+1)*numY + k];
}
u[idx] = uval;
```

Agenda

Results

- All versions validate on the small and large datasets
- Results obtained by executing on one of the APL GPU machines

Version	Total execution time (microseconds) Small	Large
Sequential CPU	2,297,659	216,305,907
OpenMP CPU	213,948	10,132,446
Naïve CUDA	92,787	6,975,193
Optimized CUDA	60,867	3,647,836

Table: Results of the three different implementations.