MAPS: Optimizing Massively Parallel Applications using Device-Level Memory Abstraction

Presented by: Eri Rubin

Eri Rubin (HUJI), Ely Levy (HUJI),

Prof. Amnon Barak (HUJI), Tal Ben-Nun (HUJI)

Talk Outline

- Motivation - What's the problem ?
- Maps, written by CUDA programmers for CUDA programmers
- See the syntax !
- Very little theory ☺
- Some technical details
- Index mapper
- Code sample
- Index mapper code sample
- Performance

Motivation

- GPUs can achieve amazing acceleration for heavy compute tasks
- However getting this high performance is hard
- There are already many different tools to make development easier
- But something is missing
- After over 8 years and many projects of developing many applications, advising on many projects and teaching numerous courses in CUDA I got tired of:
  - Writing the same annoying pieces of code over and over again
  - Spending many hours debugging annoying bugs
  - Seeing my colleagues and students fall into the same potholes
- So I developed MAPS with the help of my colleagues at the lab

MAPS - Fun Facts:

- Most software is actually memory bound (if you wrote the compute in a smart way …)
- Memory optimizations on GPUs are not fun, and lead to the “Indexing Hell”
- These can induce hard to find bugs, and prolong development time significantly
- Most algorithms actually use a very small set of access patterns. So addressing this set is a feasible task.

MAPS – Goals

- Easily let a programmer utilize advanced memory optimizations without even knowing about it !
- Remove the “Indexing Hell” by using iterators, no need to calculate indexes at all !
- Doesn’t break the CUDA programming model, so if it stops working for you, there is no need to rewrite the whole algorithm.
- Familiar STL like Container/Iterator interface

MAPS – small taste

```
template<int RAD, int BLOCK_W>
__global__ void convMAPS(const float *in, float *out, int size) {
   int x = blockIdx.x * blockDim.x + threadIdx.x;
   if (x >= size) return;
   typedef maps::Window1D<float, BLOCK_W, RAD> window1DType;
   __shared__ window1DType wnd;
   wnd.init(in, size);
   float result = 0.f;

#pragma unroll
   for (window1DType::iterator iter = wnd.begin(); iter != wnd.end(); ++iter)
      result += (*iter) * dev_convKernel[iter.id()];
   out[x] = result;
}
```

Access patterns

- Similar patterns are used in multiple algorithms
- To validate we look at “Berkeley's parallel dwarfs”
- Berkeley's parallel dwarfs – a set of algorithmic building blocks with which any parallel algorithm can supposedly be built.
- We found a set of access patterns that are used by these parallel dwarfs

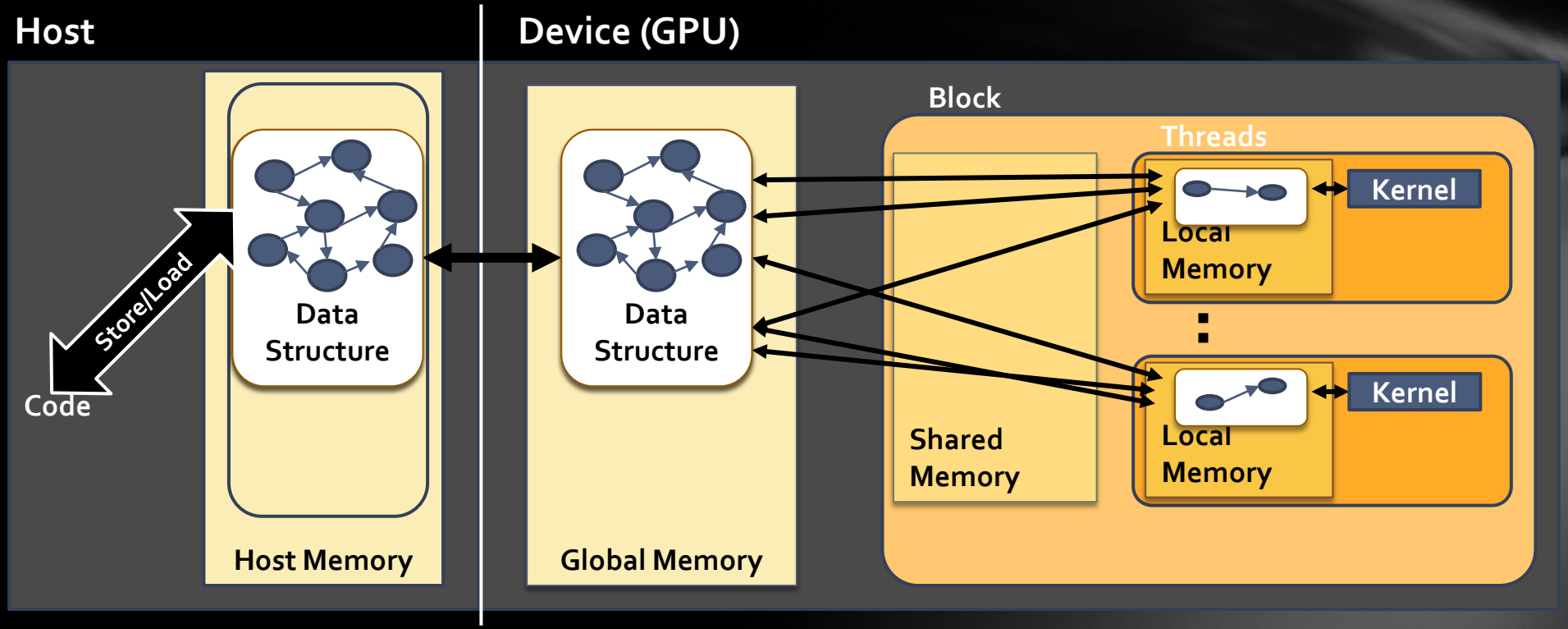
Parallel Dwarfs access pattern

Parallel Dwarf	Data Structure	Access Patterns	Typical Example
Dense Linear Algebra	Vector-Vector	Block (1D)	Dot product
	Matrix-Vector	Block (2D, 1D)	Matrix-Vector Mult.
	Matrix-Matrix	Block (2D, Transposed)	Matrix Mult.
Sparse Linear Algebra	CSR/CSC Matrix	Adjacency	SpMV
	Banded Matrix	Block (1D)	Banded Solver
Spectral Methods	Vector/Matrix	Permutation	FFT
N-Body Methods	Array	Block (1D)	All pairs N-Body
	Octree	Traversal	Barnes-Hut N-Body
Structured Grids	Grid Matrix	Window	Convolution
Unstructured Grids	Graph	Adjacency	Cloth Simulation
Graph Traversal	Graph	Varies	BFS
MapReduce	Varies	Varies	Histogram
Combinational Logic	Varies	Varies	CRC
Dynamic Programming	Varies	Varies	Needleman-Wunsch
Backtrack/Branch-and-Bound	Varies	Varies	A*, DFS
Graphical Models	Varies	Varies	HMM
Finite State Machine	Varies	Varies	Any FSM

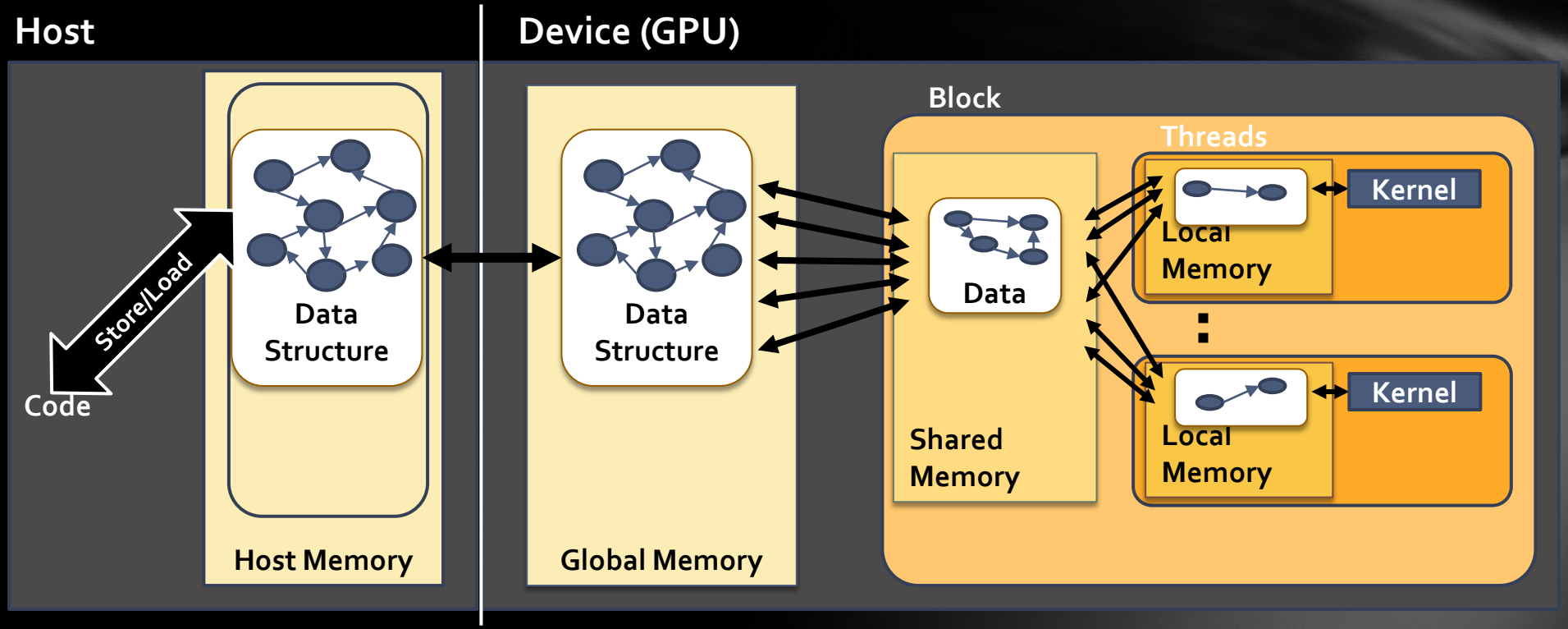
Proposed Framework - MAPS

- Base on the “Memory Dwarfs”
- Using a familiar STL style Container/Iterator interface
- Hide the “*Indexing Hell*”
- Does not limit or hinder the developer in any way
- Maintain optimized performance

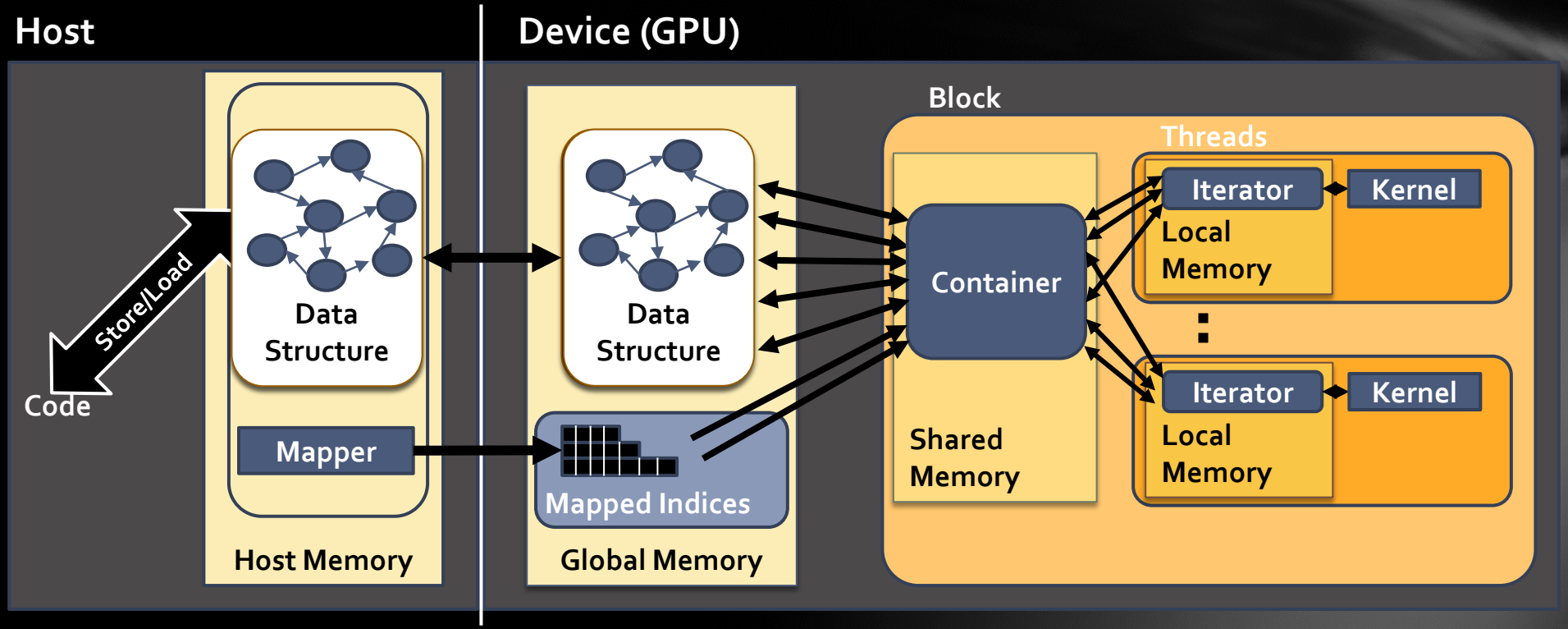
Naïve flow


Host
Device (GPU)
Store/Load
Code
Data Structure
Host Memory
Data Structure
Global Memory
Block
Threads
Kernel
Local Memory
Kernel
Local Memory
Shared Memory

Optimized flow

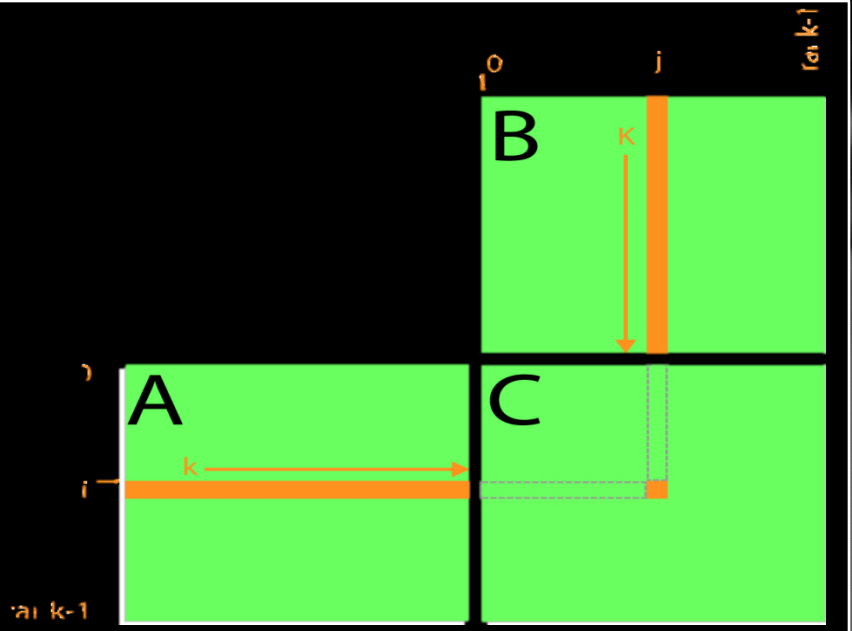


MAPS Framework



Matrix Multiplication Sample: Naïve

```
int bx = blockIdx.x;
int by = blockIdx.y;
int tx = threadIdx.x;
int ty = threadIdx.y;

int aBegin = n * BLK_SIZE * by;
int bBegin = BLK_SIZE * bx;

for (int i = 0; i < n; ++i)
{
    Csub += A[aBegin + n * ty + i] * B[bBegin + k * i + tx];
}
```



Matrix Multiplication Sample: Optimized

```
__shared__ float As[BLK_SIZE*BLK_SIZE];
__shared__ float Bs[(BLK_SIZE+1)*BLK_SIZE];
float Csub = 0;

for (a = wA * BLK_SIZE * blockIdx.y, b = BLK_SIZE * blockIdx.x;
       a <= wA * BLK_SIZE * blockIdx.y + wA - 1;
       a += BLK_SIZE, b += BLK_SIZE * wB)
{
    As[threadIdx.y*BLK_SIZE+threadIdx.x] = A[a + wA * threadIdx.y + threadIdx.x];
    Bs[threadIdx.y*(BLK_SIZE+1)+threadIdx.x] = B[b + wB * threadIdx.y + threadIdx.x];

    __syncthreads();

#pragma unroll
    for (int k = 0; k < BLK_SIZE; ++k){
        Csub += As[threadIdx.y*BLK_SIZE+k] * Bs[k*(BLK_SIZE+1)+threadIdx.x];
    }
    __syncthreads();
}
```

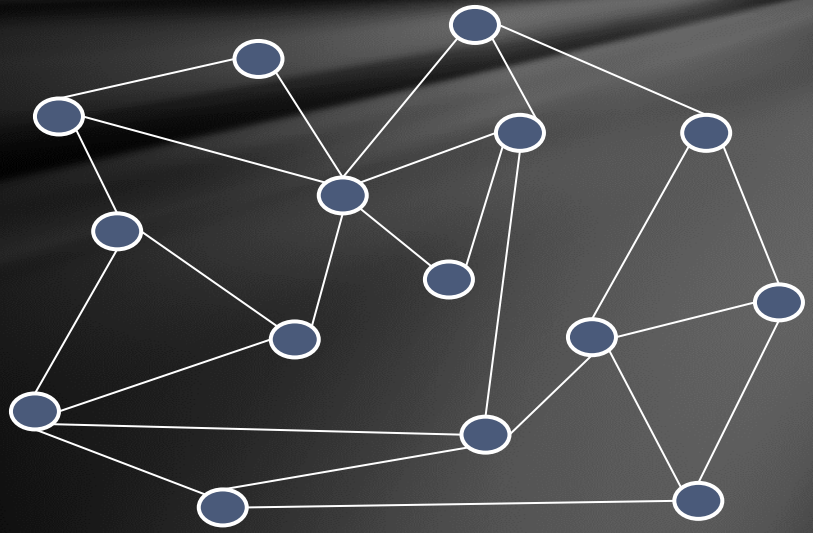
Matrix Multiplication Sample: MAPS

```
Block2D <float, BLK_SIZE> matConA;
Block2DT<float, BLK_SIZE> matConB;
matConA.init(A, m, n,As);
matConB.init(B, n, k,Bs);
Block2D <float, BLK_SIZE>::iterator matAIt;
Block2DT<float, BLK_SIZE>::iterator matBIt;

while (!matConA.isDone())
{
#pragma unroll
    for (matAIt = matConA.begin(),  matBIt = matConB.begin();  matAIt != matConA.end();
            ++matAIt, ++matBIt)
    {
        Csub += (*matAIt) * (*matBIt);
    }
    matConA.nextChunk();   matConB.nextChunk();
}
```

Host Index Mapping

A graph has a topology which in many cases is unstructured, thus cannot be known in compile time, yet in many cases it is fairly constant for a certain use case.

- Naïve access from a node to its neighbors leads to random memory access
- Yet if index locality is maintained, a group of near by nodes will have allot of overlap with their neighbors.
- Caching these items beforehand can significantly reduce the overhead of random access



Host Index Mapping

For these cases, the MAPS framework includes the **index mapper** component.

This component processes data structures on the host to find an optimal caching strategy for each thread-block.

Vertex Data

Id	Position	Normal
0	x,y,z	x,y,z
1	x,y,z	x,y,z
2	x,y,z	x,y,z
3	x,y,z	x,y,z
4	x,y,z	x,y,z
5	x,y,z	x,y,z
6	x,y,z	x,y,z

Face Data

Id	Vert Ids	Normal
0	0,1,2	x,y,z
1	2,3,0	x,y,z
2	3,4,6	x,y,z
3	4,5,6	x,y,z
4	0,6,3	x,y,z
5	3,5,2	x,y,z
6	4,5,2	x,y,z

SpMV Sample: Host – build graph

First we need to build the graph, and create the index maps

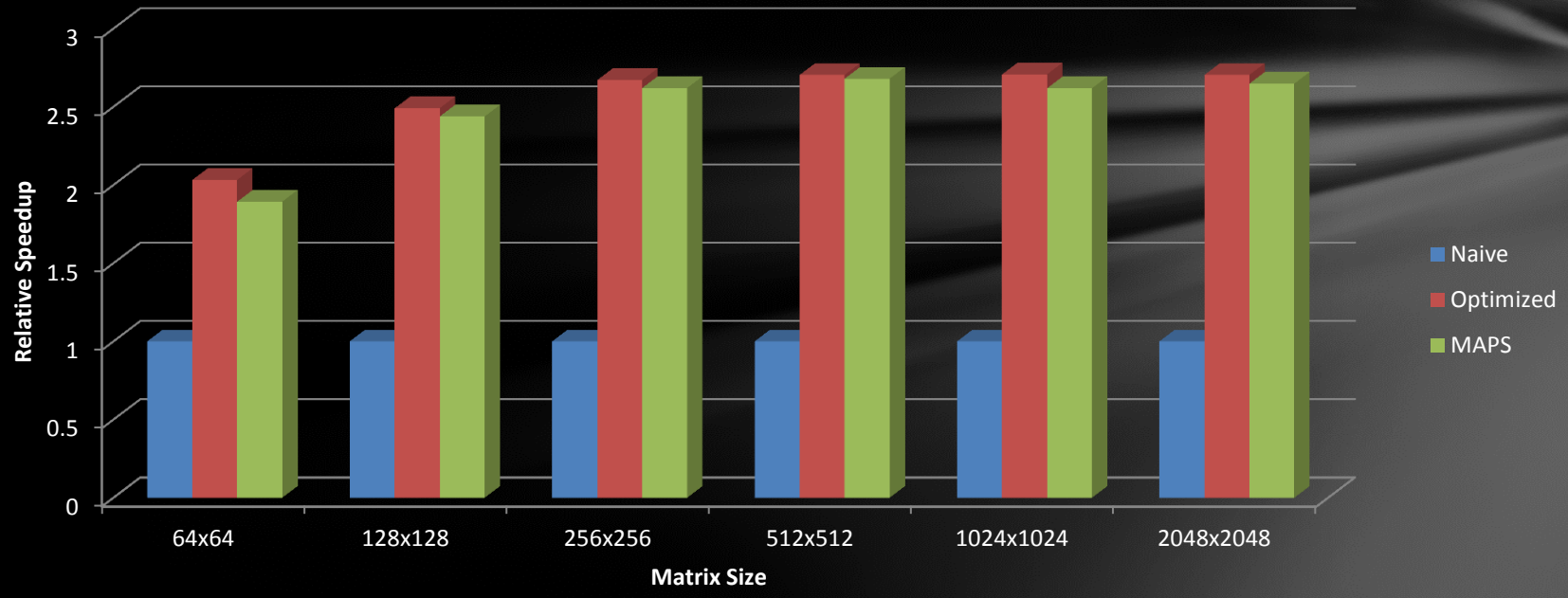
```
maps::GraphMapper indexMapper(blockSize,false);
indexMapper.init(rows);

std::vector<matCell>::iterator matIt;
for (matIt = spMat.begin(); matIt != spMat.end(); ++matIt)
    indexMapper.addEdge(matIt->i,matIt->j);

indexMapper.setMaxNodeRankSize(maxNRank);

indexMapper.createIndexMap();
int sharedMemSize_con = sizeof(float)*indexMapper._MaxNumOfConstVecsInBlock;
unsigned int numPartRoundUp = maps::RoundUp(cols,512)*512;

SPmV_maps_kernel_maps <<<gridDim, blockDim, sharedMemSize_con>>> (rows, d_A_val, d_A_j_ind,
   d_A_lineStartInd, d_x, d_b, indexMapper._gpuData, cols, indexMapper._MaxNumOfConstVecsInBlock,
   numPartRoundUp);
```

SpMV Sample: Device – build graph

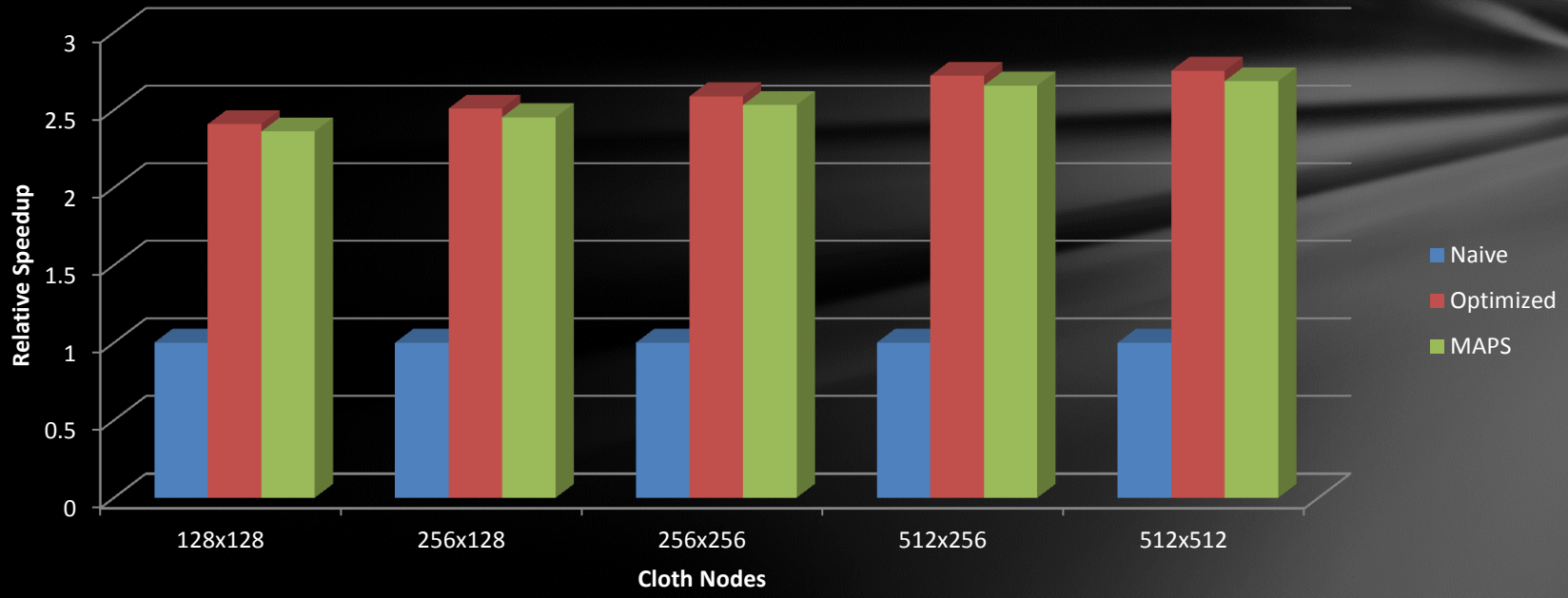
First we need to build the graph, and create the index maps

```
extern __shared__ float sdata[];

maps::Adjacency<float,false> MyGraph;

MyGraph.init(threadIdx.x, blockDim.x, g_x ,sdata , GraphGPUData, MaxNumOfConstVecsInBlock, global_ind,
             numPartRoundUp);

if (global_ind < N) {
    int lineStartIndex = g_A_lineStartInd[global_ind];
    int nextLineStartInd = g_A_lineStartInd[global_ind+1];

    float res=0.f;

    maps::Adjacency<float,false>::iterator gIter = MyGraph.begin();
    for (int i=lineStartIndex; i<nextLineStartInd; ++i,++gIter)
         res += g_A_val[i] * (*gIter);

    g_b[global_ind] = res;
}
```

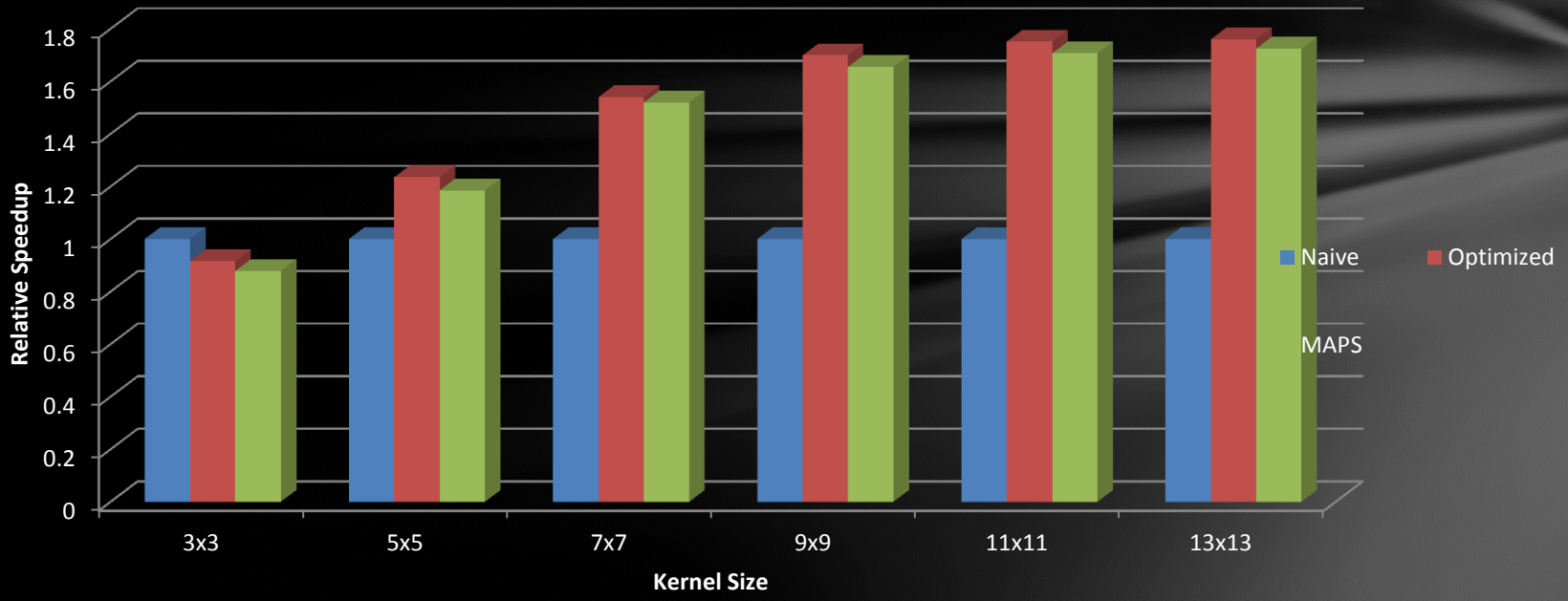
Dense Matrix Multiplication – Block 2D



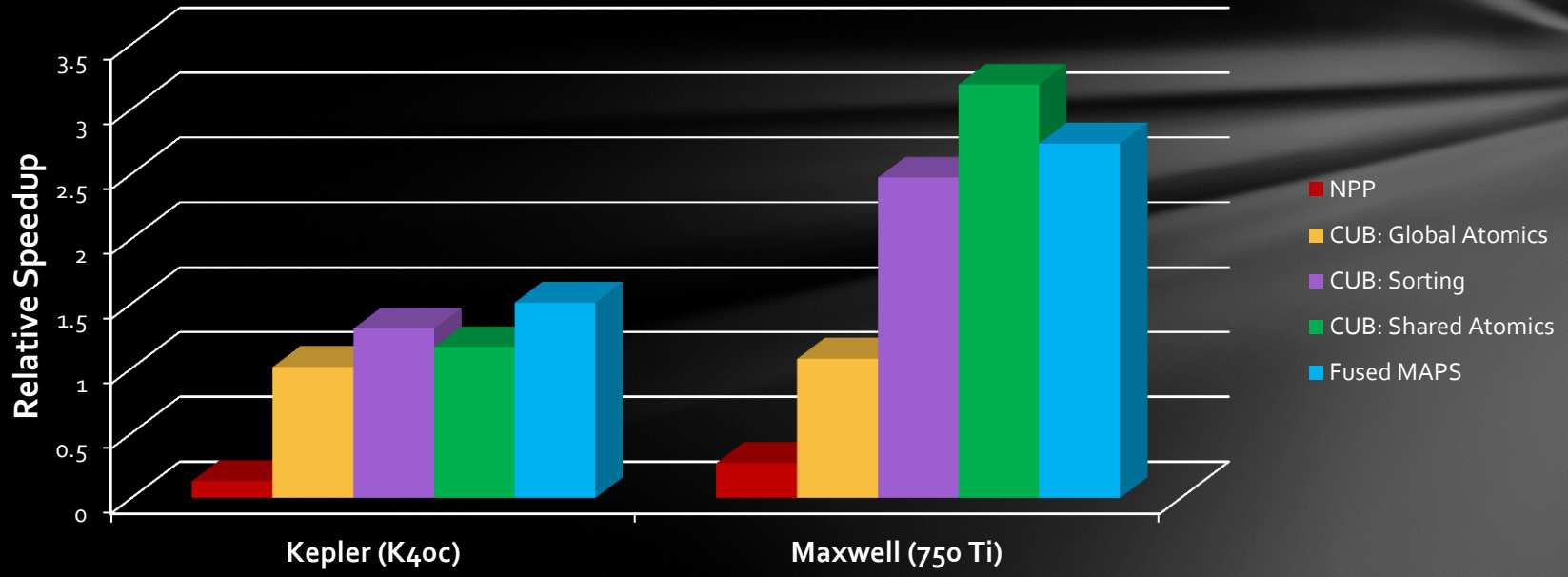
Cloth Simulation - Adjacency



2D Convolution



Cross Platform Benchmark – Fused Convolution + Histogram



Conclusion

This work presented a novel framework

Defined a set of “Memory Dwarfs” based on Parallel Dwarfs

The MAPS framework creates an abstraction, exposing a familiar STL-like API

An implementation has been written and is publicly available

Future Work

Writing a port of the library over higher level languages such and C++ AMP, Python

Enhance the model to allow for even more automatic optimizations (e.g. ILP)

Integration with other libraries

Thank you

Questions ?

Questions can be sent to:

- eri.rubin@gmail.com
- talbn@cs.huji.ac.il

Library to be published at http://www.cs.huji.ac.il/~talbn/maps

This research was partially supported by the Ministry of Science and Technology, Israel.