Programming Techniques for Supercomputers: Performance Modelling for GPUs

Stream Benchmark

Jacobi 2D & 3D



Prof. Dr. G. Wellein[a,b], Dr. G. Hager[a], D. Ernst[b], Prof. Dr. Harald Köstler[a,b], Dr. S. Kuckuk[a,b]
[a]Erlangen National High Performance Computing Center (NHR@FAU)
[b]Department für Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg, Sommersemester 2022

Target GPU (V100)

- Throughout this lesson: target GPU is an NVIDIA V100
  - 80 SMs with 32 FP64 and 64 FP32 cores each
  - Memory
    - $16\ GB$ DRAM
    - $6\ 144\ KB$ L2 cache (shared by all SMs)
    - $128\ KB$ combined L1 cache and shared memory *per SM*
  - Theoretical peak performance:
    - $P_{peak} = 7 \frac{TF}{s}$ for double precision and
    - $P_{peak} = 14 \frac{TF}{s}$ for single precision
  - Theoretical peak main memory bandwidth:
    - $b_s = 900 \frac{GB}{s}$

Stream Benchmark (V100)

```
__global__ void scale_kernel(double *A __restrict__, const double *__restrict__ B, const size_t N) {
  int tidx = threadIdx.x + blockIdx.x * blockDim.x;

  if( tidx >= N )
    return;

  A[tidx] = 0.2 * B[tidx];
}
```

Ensure thread index is within bounds

Data does not overlap

GPU kernel

Courtesy of D. Ernst, https://github.com/te42kyfo/cuda-benches

Stream Benchmark (V100)

```
__global__ void scale_kernel(double *A __restrict__, const double *__restrict__ B, const size_t N) {
  int tidx = threadIdx.x + blockIdx.x * blockDim.x;

  if( tidx >= N )
    return;

  A[tidx] = 0.2 * B[tidx];
}
```

Ensure thread index is within bounds

Data does not overlap

GPU kernel

```
int main(int argc, char **argv) {
  const size_t N = 640 * 256;

  // allocate A, B and their device counter-parts dA, dB; synchronize data

  scale_kernel<<<640, 256>>>(dA, dB, N);

  // synchronize data & clean-up
}
```

GPU kernel call

Number of CUDA blocks

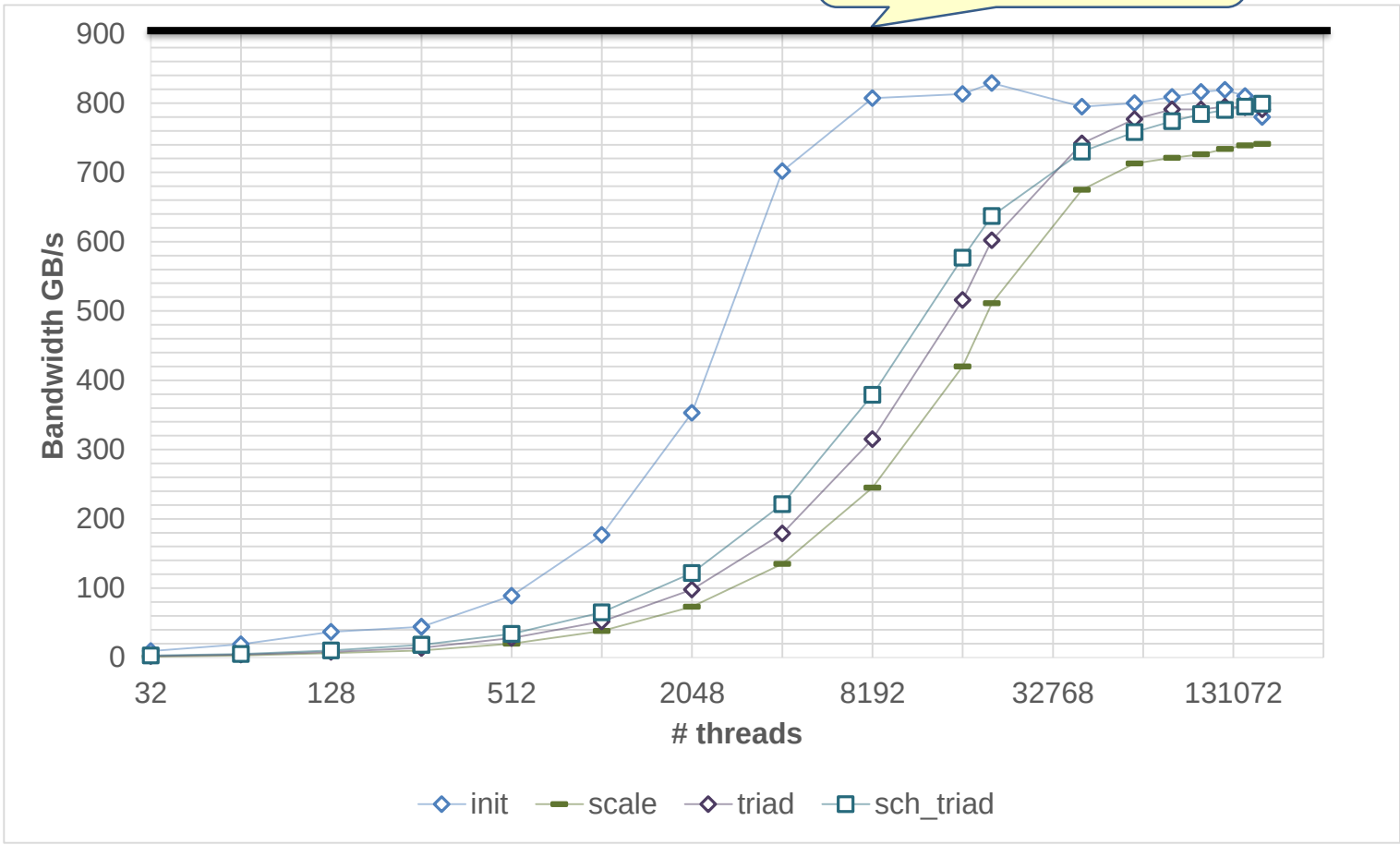
Size of each CUDA block

Host code

Courtesy of D. Ernst, https://github.com/te42kyfo/cuda-benches

Stream Benchmark (V100)

- Measured memory bandwidth for different test cases
  - Init
    - `A[i] = c`
  - Scale
    - `A[i] = c*B[i]`
  - Triad
    - `A[i] = B[i] + c*C[i]`
  - Sch_triad
    - `A[i] = B[i] + C[i]*D[i]`



Courtesy of D. Ernst, https://github.com/te42kyfo/cuda-benches

NHR FAU

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Case Study: 2D Stencil

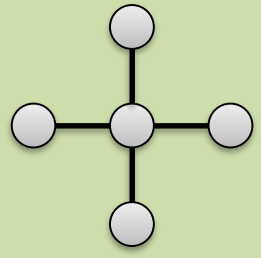


Case Study: 2D Stencil

- CPU code to GPU code

Data does not overlap

```
inline void jacobi(int nx, int ny, double * __restrict__ u, double * __restrict__ u_new) {
  for (int j = 1; j < ny - 1; ++j) {
    for (int i = 1; i < nx - 1; ++i) {
      u_new[j*nx + i] = 0.25 * ( u[ j   *nx + i-1] + u[ j   *nx + i+1]
                               + u[(j-1)*nx + I  ] + u[(j+1)*nx + i  ] );
    }
  }
}
```



CPU version

```
__global__ void jacobi(int nx, int ny, double * __restrict__ u, double * __restrict__ u_new) {
  int i = blockIdx.x * blockDim.x + threadIdx
  int j = blockIdx.y * blockDim.y + threadIdx

  if (i < 1 || j < 1 || i >= nx - 1 || i
    return;

  u_new[j*nx + i] = 0.25 * ( u[ j   *nx + i-1] + u[ j   *nx + i+1]
                           + u[(j-1)*nx + I  ] + u[(j+1)*nx + i  ] );
}
```

Iteration indices derived from loop indices

Ensure iteration space bounds (loop bounds)

GPU kernel

Update as in the CPU version

Case Study: 2D Stencil

- CPU code to GPU code continued

```
/// original CPU version ///
for (int i = 0; i < n_it; ++i) {
  jacobi(nx, ny, u, u_new);
  std::swap(u, u_new);
}
```

```
const size_t buffer_size = sizeof(double) * nx * ny;

double *d_u, *d_u_new;
cudaMalloc((void**)&d_u, buffer_size);
cudaMalloc((void**)&d_u_new, buffer_size);                    Allocate GPU memory

cudaMemcpy(d_u, u, buffer_size, cudaMemcpyHostToDevice);
cudaMemcpy(d_u_new, u_new, buffer_size, cudaMemcpyHostToDevice);    Copy data from CPU to GPU

dim3 blockSize(16, 16);
dim3 numBlocks((nx + blockSize.x - 1) / blockSize.x, (ny + blockSize.y - 1) / blockSize.y);

for (int i = 0; i < n_it; ++i) {
  jacobi<<<numBlocks, blockSize>>>(nx, ny, d_u, d_u_new);     GPU kernel call
  std::swap(d_u, d_u_new);                                    Swap device pointers
}

cudaMemcpy(u, d_u, buffer_size, cudaMemcpyDeviceToHost);
cudaMemcpy(u_new, d_u_new, buffer_size, cudaMemcpyDeviceToHost);    Copy data back from GPU to CPU

                                                              CPU code
```

What is a good block size?

- 1 thread per grid point, GLUPS for varying block size, V100

problem size 64 x 64

	1	2	4	8	16	32	64	128	256	512	1024
1	0,4	0,7	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,1
2	0,7	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	
4	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0		
8	1,0	1,0	1,0	1,0	1,0	1,1	1,0	1,0			
16	1,0	1,0	1,0	1,0	1,0	1,0	1,0				
32	1,0	1,0	1,0	1,0	1,0	1,0					
64	1,0	1,0	1,0	1,0	1,0						
128	1,0	1,0	1,0	1,0							
256	1,0	1,0	1,0								
512	1,0	1,0									
1024	1,0										

problem size 256 x 256

	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,1	2,2	4,0	7,1	11,2	15,9	16,6	16,7	16,6	16,7
2	1,2	2,1	4,0	7,0	11,3	16,1	16,7	16,7	16,5	16,7	
4	2,4	4,0	7,0	11,3	16,1	16,5	16,6	16,4	16,6		
8	4,4	7,0	11,2	16,0	16,6	16,3	16,5	16,8			
16	7,7	11,2	15,6	16,5	16,6	16,7	16,5				
32	9,5	13,7	15,0	16,7	16,5	16,4					
64	8,8	13,0	16,6	16,6	16,6						
128	8,3	12,3	16,7	16,5							
256	7,6	11,3	16,4								
512	7,5	11,4									
1024	7,4										

problem size 1024 x 1024

	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,3	2,5	5,1	10,0	19,5	36,8	44,3	43,9	43,8	43,7
2	1,3	2,5	5,1	10,0	19,4	36,7	44,4	44,0	43,9	43,8	
4	2,5	5,1	10,0	19,5	36,7	44,5	44,1	44,0	43,9		
8	5,1	10,0	19,5	36,9	44,5	44,2	44,1	43,9			
16	10,0	19,4	36,7	45,1	44,0	44,2	44,0				
32	14,4	28,0	44,3	45,1	44,1	44,0					
64	14,7	27,4	44,0	44,6	43,9						
128	14,3	27,2	43,9	44,1							
256	14,0	23,6	43,4								
512	13,2	22,2									
1024	12,2										

problem size 4096 x 4096

	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,3	2,6	5,1	10,2	20,5	40,7	49,5	49,4	49,3	49,0
2	1,3	2,6	5,1	10,2	20,5	40,7	49,6	49,5	49,4	49,2	
4	2,6	5,1	10,2	20,5	40,7	49,6	49,6	49,5	49,3		
8	5,1	10,2	20,4	40,7	49,5	49,8	49,7	49,5			
16	10,2	20,4	40,6	49,7	49,8	50,0	49,7				
32	14,8	30,3	50,3	50,5	50,4	50,5					
64	15,7	29,5	50,0	50,5	50,2						
128	14,9	24,5	45,0	48,4							
256	14,7	27,0	46,9								
512	14,0	24,9									
1024	12,8										

Block size in y-direction

Block size in x-direction

What is a good block size?

- 1 thread per grid point, GLUPS for varying block size, V100

problem size 256 x 256

	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,1	2,2	4,0	7,1	11,2	15,9	16,6	16,7	16,6	16,7
2	1,2	2,1	4,0	7,0	11,3	16,1	16,7	16,7	16,5	16,7	
4	2,4	4,0	7,0	11,3	16,1	16,5	16,6	16,4	16,6		
8	4,4	7,0	11,2	16,0	16,6	16,3	16,5	16,8			
16	7,7	11,2	15,6	16,5	16,6	16,7	16,5				
32	9,5	13,7	15,0	16,7	16,5	16,4					
64	8,8	13,0	16,6	16,6	16,6						
128	8,3	12,3	16,7	16,5							
256	7,6	11,3	16,4								
512	7,5	11,4									
1024	7,4										

problem size 1024 x 1024

	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,3	2,5	5,1	10,0	19,5	36,8	44,3	43,9	43,8	43,7
2	1,3	2,5	5,1	10,0	19,4	36,7	44,4	44,0	43,9	43,8	
4	2,5	5,1	10,0	19,5	36,7	44,5	44,1	44,0	43,9		
8	5,1	10,0	19,5	36,9	44,5	44,2	44,1	43,9			
16	10,0	19,4	36,7	45,1	44,0	44,2	44,0				
32	14,4	28,0	44,3	45,1	44,1	44,0					
64	14,7	27,4	44,0	44,6	43,9						
128	14,3	27,2	43,9	44,1							
256	14,0	23,6	43,4								
512	13,2	22,2									
1024	12,2										

problem size 4096 x 4096

	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,3	2,6	5,1	10,2	20,5	40,7	49,5	49,4	49,3	49,0
2	1,3	2,6	5,1	10,2	20,5	40,7	49,6	49,5	49,4	49,2	
4	2,6	5,1	10,2	20,5	40,7	49,6	49,6	49,5	49,3		
8	5,1	10,2	20,4	40,7	49,5	49,8	49,7	49,5			
16	10,2	20,4	40,6	49,7	49,8	50,0	49,7				
32	14,8	30,3	50,3	50,5	50,4	50,5					
64	15,7	29,5	50,0	50,5	50,2						
128	14,9	24,5	45,0	48,4							
256	14,7	27,0	46,9								
512	14,0	24,9									
1024	12,8										

problem size 16384 x 16384

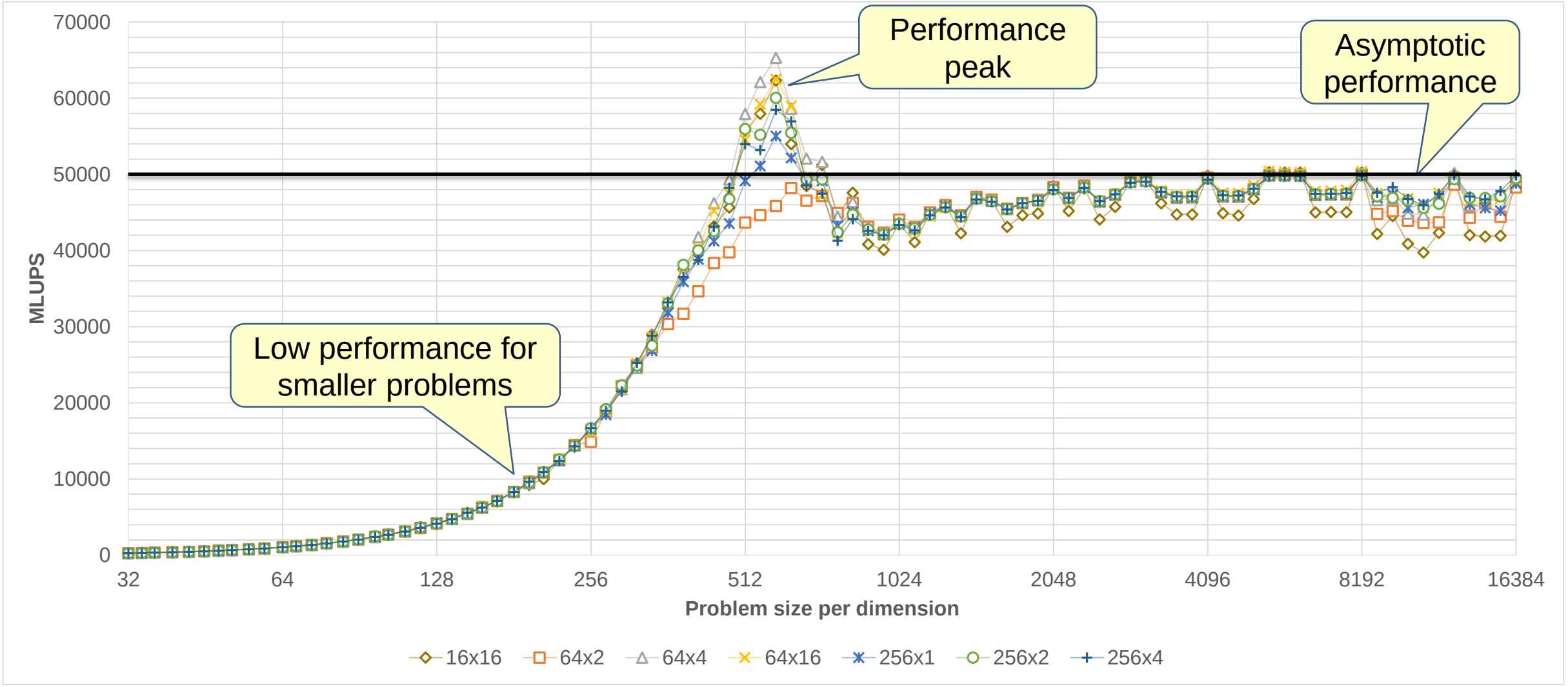
	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,3	2,6	5,1	10,3	20,5	40,9	48,6	48,2	48,4	48,8
2	1,3	2,6	5,1	10,2	20,5	41,0	48,1	48,3	48,5	49,0	
4	2,6	5,1	10,2	20,5	41,0	48,6	48,8	49,3	49,9		
8	5,1	10,2	20,5	40,9	49,3	48,4	48,7	48,9			
16	10,2	20,5	40,9	50,1	49,8	48,8	49,2				
32	14,8	29,5	48,1	49,0	48,3	48,0					
64	13,9	26,3	48,3	48,9	47,7						
128	12,8	23,8	38,6	48,0							
256	11,9	22,5	29,9								
512	11,1	19,8									
1024	9,2										

Block size in y-direction

Block size in x-direction

What is a good block size?

- 1 thread per grid point, performance for varying block size



Performance Modeling

- Roofline (recap)
  - $P = \min(P_{peak}, I * b_s)$
- Each update requires
  - 1 load and 1 store (no write allocate for consecutive stores to DRAM)

  $\Rightarrow$ 16 bytes / LUP
  - 1 multiplication and 3 additions

  $\Rightarrow$ 4 flops / LUP

  $\Rightarrow$ Computational intensity $I = \frac{4F}{16B} = 0.25\frac{F}{B}$
- Estimated roofline performance
  - $P = \min(P_{peak}, I * b_s) = \min\left(7 \cdot 10^{12}\frac{F}{s}, 0.25\frac{F}{B} * 900 \cdot 10^{9}\frac{B}{s}\right) = 225 \cdot 10^{9}\frac{F}{s}$

```
u_new[j*nx + i] = 0.25 * (
    u[ j   *nx + i-1] + u[ j   *nx + i+1]
  + u[(j-1)*nx + I  ] + u[(j+1)*nx + I  ]
);
```

Performance Modeling

- Estimated roofline performance
  - $P = \min\left(P_{peak}, I * b_s\right) = \min\left(7\frac{TF}{s}, 0.25\frac{F}{B} * 900\frac{GB}{s}\right) = 225\frac{GF}{s}$

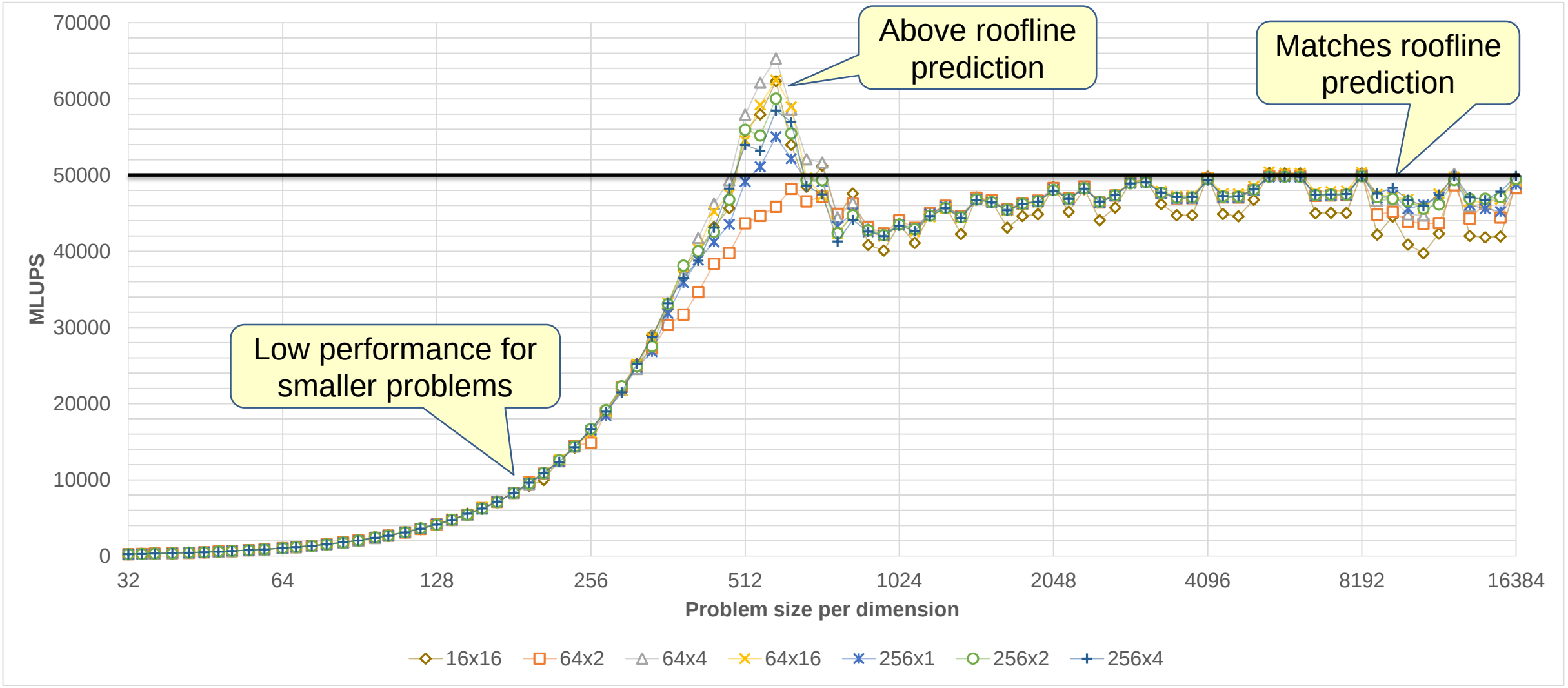
  $\Rightarrow$ The kernel is massively main memory bound
- How many lattice updates per second are the limit?
  - With $4\frac{F}{LUP}$, P translates to $225\frac{GF}{s} / 4\frac{F}{LUP} = 56.25\ GLUPS$
- A more realistic estimation can be obtained by repeating the calculation with the measured peak memory bandwidth (c.f. stream benchmark) of 800 GB/s

  $\Rightarrow \frac{\min\left(7\frac{TF}{s}, 0.25\frac{F}{B} * 800\frac{GB}{s}\right)}{4\frac{F}{LUP}} = 50\ GLUPS$

In good accordance with our measurements for medium and large problem sizes

What is a good block size?

- 1 thread per grid point, performance for varying block size



Performance Modeling

- Performance above the roofline for sizes around 512^2
  - Best performance for problem size 588^2
  - Memory requirement: $588^2 \cdot 16B \approx 5\,400\,KB$
  - L2 Cache size for the V100 is $6\,144\,KB$

  $\Rightarrow$ Problem fits into L2 cache
- Comparison with measured transferred data volume and bandwidth
  - Block size 64×4, problem size 588^2
  - Measured DRAM read | write volume: $0\,B$ | $24\,KB$
  - Measured L2 read | write bandwidth: $729\,\frac{GB}{s}$ | $451\,\frac{GB}{s}$
  - Measured L2 aggregated bandwidth: $1180\,\frac{GB}{s}$

Plugging this as bandwidth into our model yields $73.75\ GLUPS$

Performance Modeling

- Performance above the roofline for sizes around 512^2
  - $\Rightarrow$ Problem fits into L2 cache
- Low performance for problem sizes between 32^2 and $\sim 400^2$
  - Are we using the full extend of the GPU?
  - Possible threads per device (V100)
    - $2\,048$ threads per SM, 80 SMs = $163\,840$ threads
  - At one thread per grid point this requires a problem size of at least 405^2
- Comparison with measuring occupancy:
  - Block size 64×4, problem size $128^2 = 16\,384$
  - Measured occupancy: 0.12

NHR FAU

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Case Study: 3D Stencil



Case Study: 3D Stencil

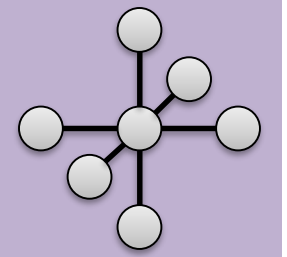
- Analogous to the 2D case

```
__global__ void jacobi(int nx, int ny, int nz, double * __restrict__ u, double * __restrict__ u_new)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;
  int k = blockIdx.z * blockDim.z + threadIdx.z;

  if (i < 1 || j < 1 || k < 1 || i >= nx - 1 || j >= ny - 1 || k >= nz - 1)
    return;

  u_new[ k   *nx*ny +  j   *nx + i  ] = 0.25 * (
      u[ k   *nx*ny +  j   *nx + i-1] + u[ k   *nx*ny +  j   *nx + i+1]
    + u[ k   *nx*ny + (j-1)*nx + i  ] + u[ k   *nx*ny + (j+1)*nx + i  ]
    + u[(k-1)*nx*ny +  j   *nx + i  ] + u[(k+1)*nx*ny +  j   *nx + i  ] );
}
```

Canonical extension to 3D with additional index



GPU kernel

What is a good block size?

- 1 thread per grid point, GLUPS for varying block size, V100, problem size 128^3

1	1	2	4	8	16	32	64	128	256	512	1024
1	0,6	1,3	2,6	5,1	10,1	20,0	38,9	44,3	44,4	43,7	22,2
2	1,3	2,6	5,1	10,1	20,0	38,8	44,4	44,1	44,1	34,8	
4	2,6	5,1	10,1	20,0	38,7	44,6	44,3	44,0	43,7		
8	5,1	10,1	20,0	38,6	44,5	44,4	44,2	43,8			
16	10,1	18,5	36,3	45,0	44,4	44,2	44,0				
32	10,3	20,5	36,0	45,2	44,5	44,1					
64	11,3	18,8	36,3	44,7	43,9						
128	10,2	18,8	35,7	43,8							
256	10,3	18,8	34,0								
512	10,1	18,3									
1024	9,8										

Block size z = 1

2	1	2	4	8	16	32	64	128	256	512
1	1,3	2,6	5,1	10,1	20,0	38,7	44,4	44,1	44,0	33,2
2	2,6	5,1	10,1	20,0	38,6	44,5	44,3	44,0	43,3	
4	5,1	10,1	19,9	38,6	44,5	44,5	44,3	44,0		
8	10,1	19,9	38,4	44,9	44,4	44,3	44,1			
16	12,0	22,9	37,9	44,8	44,2	44,1				
32	12,5	21,9	41,8	44,8	44,1					
64	11,8	21,8	41,3	44,1						
128	11,6	21,2	40,0							
256	11,5	20,8								
512	11,1									

Block size z = 2

4	1	2	4	8	16	32	64	128	256
1	2,6	5,1	10,1	20,0	38,5	44,3	44,0	43,8	43,2
2	5,1	10,1	19,9	38,5	44,3	44,2	44,0	43,7	
4	10,1	19,9	38,5	44,5	44,2	44,1	43,8		
8	12,7	23,4	42,7	44,4	44,0	43,8			
16	13,1	23,4	40,7	44,2	43,7				
32	12,3	23,5	42,5	43,8					
64	12,3	21,8	40,8						
128	11,2	19,9							
256	10,9								

Block size z = 4

8	1	2	4	8	16	32	64	128
1	5,1	10,1	19,9	38,5	44,1	43,9	43,7	43,3
2	10,1	19,9	38,4	43,8	43,9	43,8	43,4	
4	12,7	23,8	42,3	43,9	43,8	43,4		
8	13,3	24,1	42,3	43,7	42,9			
16	12,8	24,1	39,5	42,5				
32	12,5	22,7	40,8					
64	11,8	19,9						
128	10,3							

Block size z = 8

16	1	2	4	8	16	32	64
1	10,1	19,4	38,0	43,1	43,1	42,9	42,3
2	11,9	22,7	40,5	43,2	42,9	42,3	
4	13,2	23,9	40,9	42,9	42,1		
8	12,9	24,2	40,8	42,1			
16	12,5	22,8	37,5				
32	11,8	20,3					
64	10,5						

Block size z = 16

32	1	2	4	8	16	32
1	10,2	21,4	37,5	41,5	41,5	41,1
2	12,7	22,6	39,3	41,4	40,9	
4	12,8	23,6	39,3	40,7		
8	12,5	22,4	38,1			
16	11,5	20,5				
32	10,5					

Block size z = 32

64	1	2	4	8	16
1	11,3	19,9	37,3	41,0	40,7
2	12,0	22,4	38,6	40,6	
4	12,4	22,2	38,0		
8	11,2	20,4			
16	10,6				

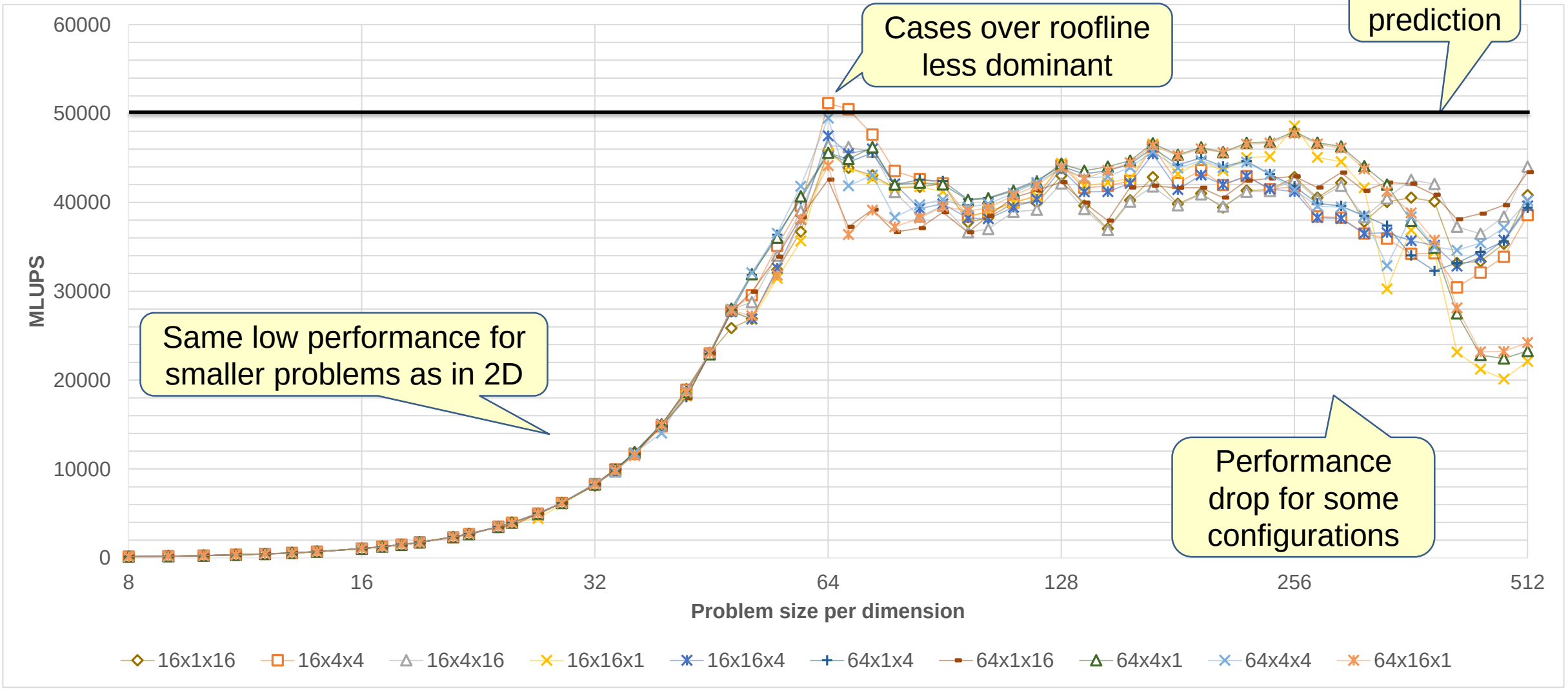
Block size z = 64

Block size in y-direction

Block size in x-direction

What is a good block size?

- 1 thread per grid point, MLUPS for varying block size

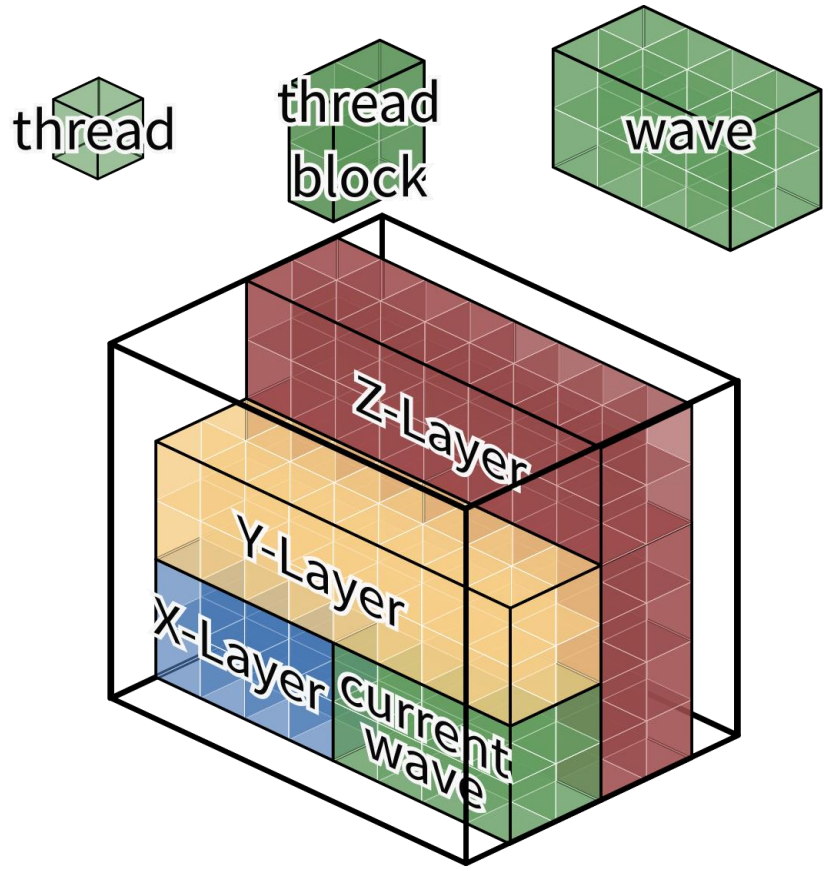


Performance Model

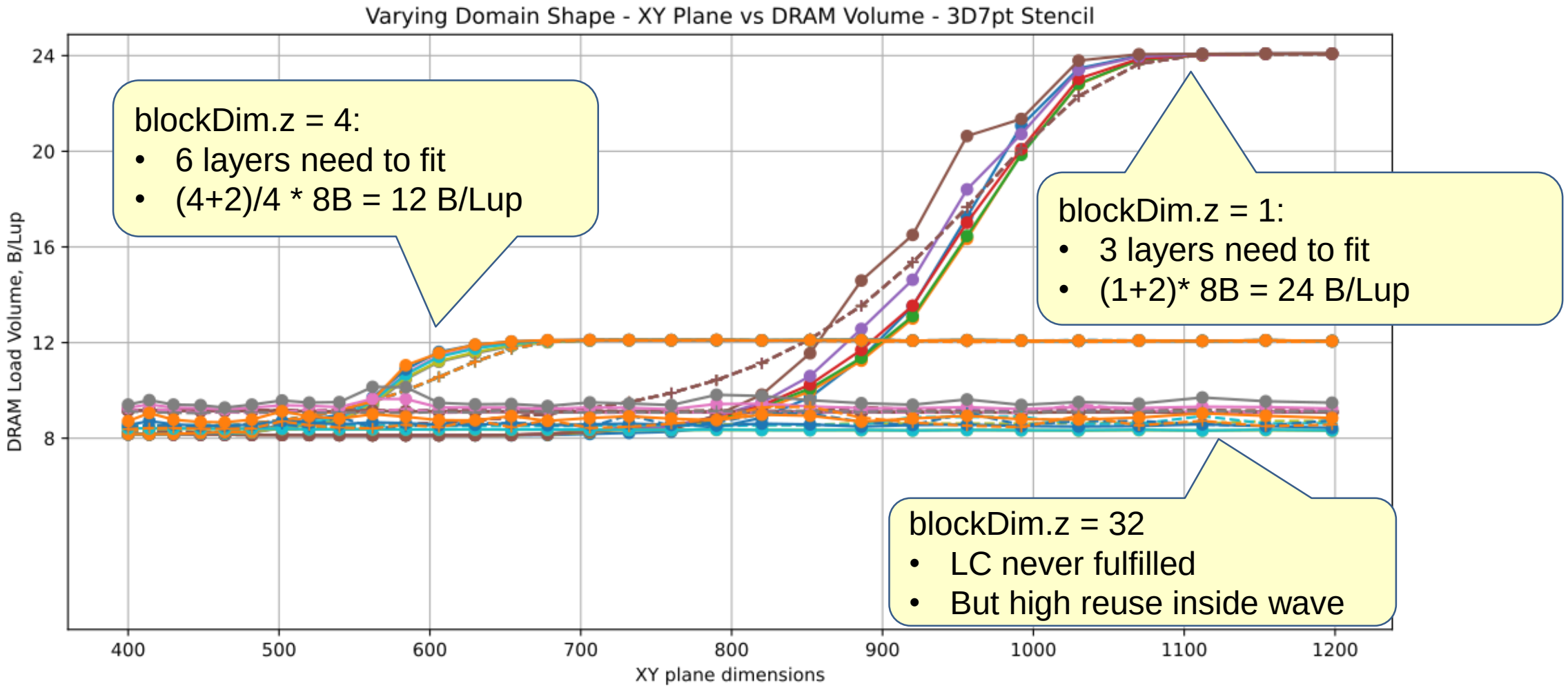
- Low performance for problem sizes between 8^3 and $\sim 60^3$
  - Similar problem as in 2D – GPU can handle $163\,840$ threads
  - At one thread per grid point this requires a problem size of at least 55^3

- Performance drop for some configurations starting at problem size $> 400^3$
  - Common characteristic: block size in z-direction = 1
  - Assuming a lexicographical scheduling of the blocks on the GPU, a layer condition for L2 can be set up
  - Need to fit a complete Z-Layer into L2 cache

  -> a 400^2 slice fits into V100's L2 cache



Performance Model - Layer Condition



3D 7pt stencil, A100 (40MB L2 cache), varying XY plane, constant total grid volume