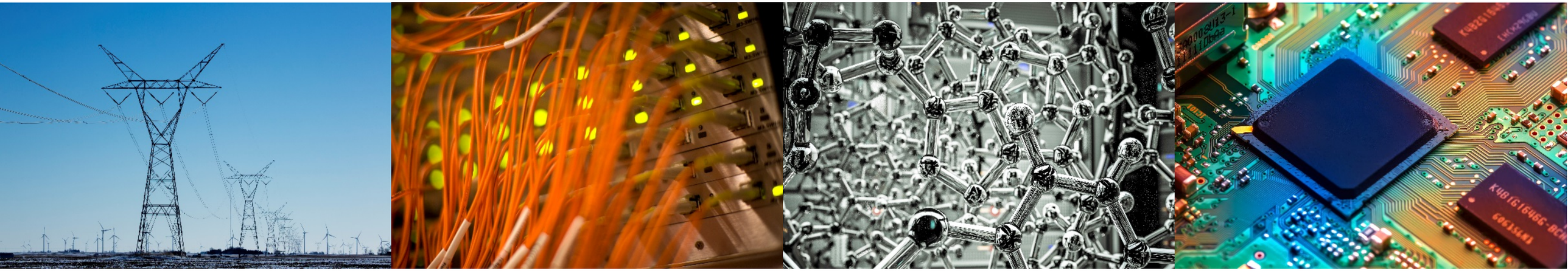
CUDA Experiences: Over-Optimization and Future HPC

Carl Pearson[1], Simon Garcia De Gonzalo[2]
Ph.D. candidates, Electrical and Computer Engineering[1] / Computer Science[2], University of Illinois Urbana-Champaign
Advised by Professor Wen-Mei Hwu



ECE ILLINOIS

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

CUDA Over-Optimization

In the trenches with ChaNGa

Simon Garcia de Gonzalo (grcdgnz2@illinois.edu)

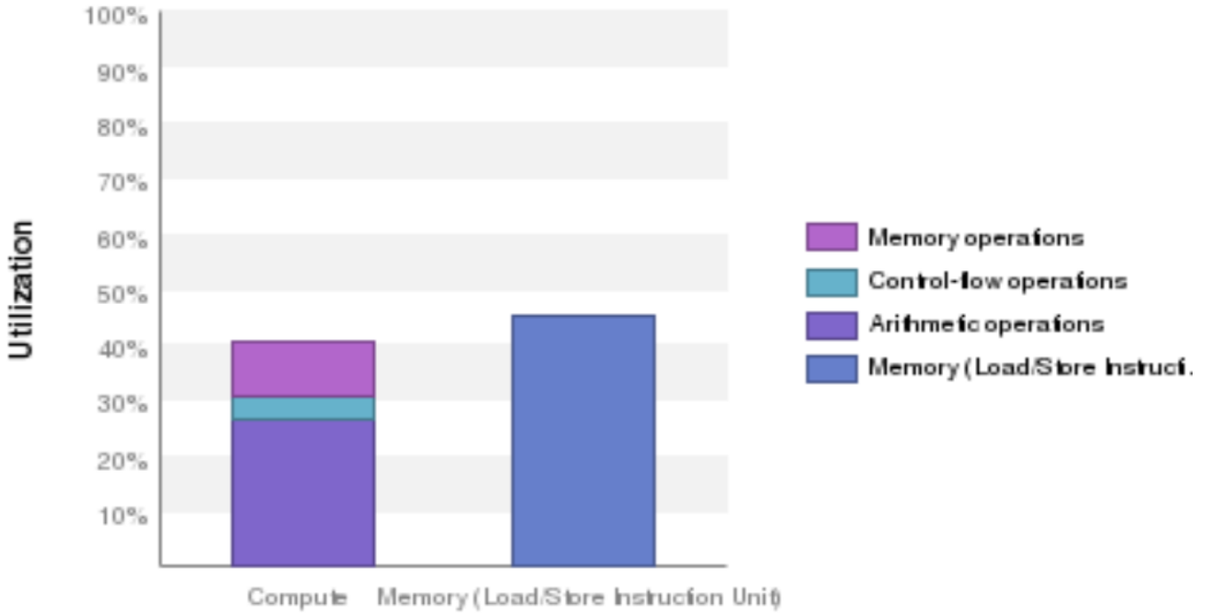
Types of bottle necks found in CUDA kernels

- Compute bound:
  - Most of the kernel time is spend in arithmetic operations
  - Compute cores are kept well supplied by the memory subsystem
  - Latency is hidden by computation

- Bandwidth bound:
  - Most of the time spend in memory operations and the kernel approaches peak bandwidth limit.
  - Compute unites are under supplied and generally waiting for data
  - Can be improve by explicitly taking advantage of the CUDA memory hierarchy

Latency limited kernels

- Characterized by having both low compute utilization and low memory utilization
- Low GPU occupancy is the main factor in this type of limitation.
- Unlike latency oriented CPUs, GPUs need a large degree of ILP to hide instruction latency.
- Common issue for highly optimized kernels that overuse limited resources that lowers possible achievable occupancy.



Resources that limit occupancy

- The following table contain the resources that are most likely to cause low occupancy

Accelerator	Maximum Threads per SM	Maximum Blocks per SM	Shared Memory per SM	Maximum Registers per Threads
C2070 (Fermi)	1536	8	48KB	63
K20X (Kepler)	2048	16	48KB	255
M40 (Maxwell)	2048	32	96KB	255
P100 (Pascal)	2048	32	64KB	255

Reducing share memory in ChaNGa kernels

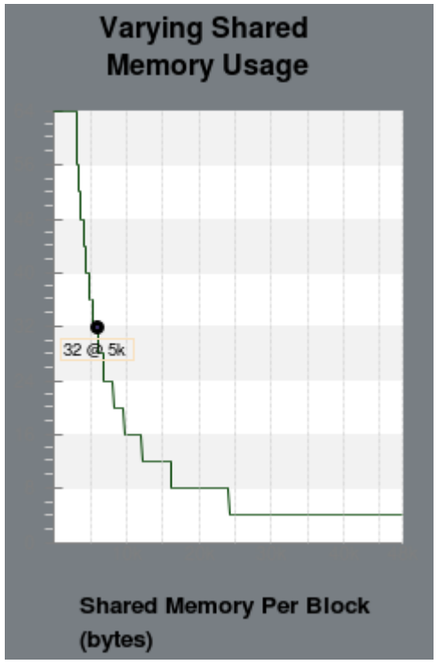
- 5.97KB of shared memory per block was being used
- Tesla K20X is configured to have 48KB of shared memory per SMX
- Each SMX was limited to simultaneously execute only 8 blocks (32 warps) out of the possible 16 block (64 warps)
- What to do:

```
//__shared__ CudaVector3D acc[THREADS_PER_BLOCK_PART];
// __shared__ cudatype pot[THREADS_PER_BLOCK_PART];
// __shared__ cudatype idt2[THREADS_PER_BLOCK_PART];
CudaVector3D acc;
cudatype pot;
cudatype idt2;
```

  - Shuffle instruction for reduction

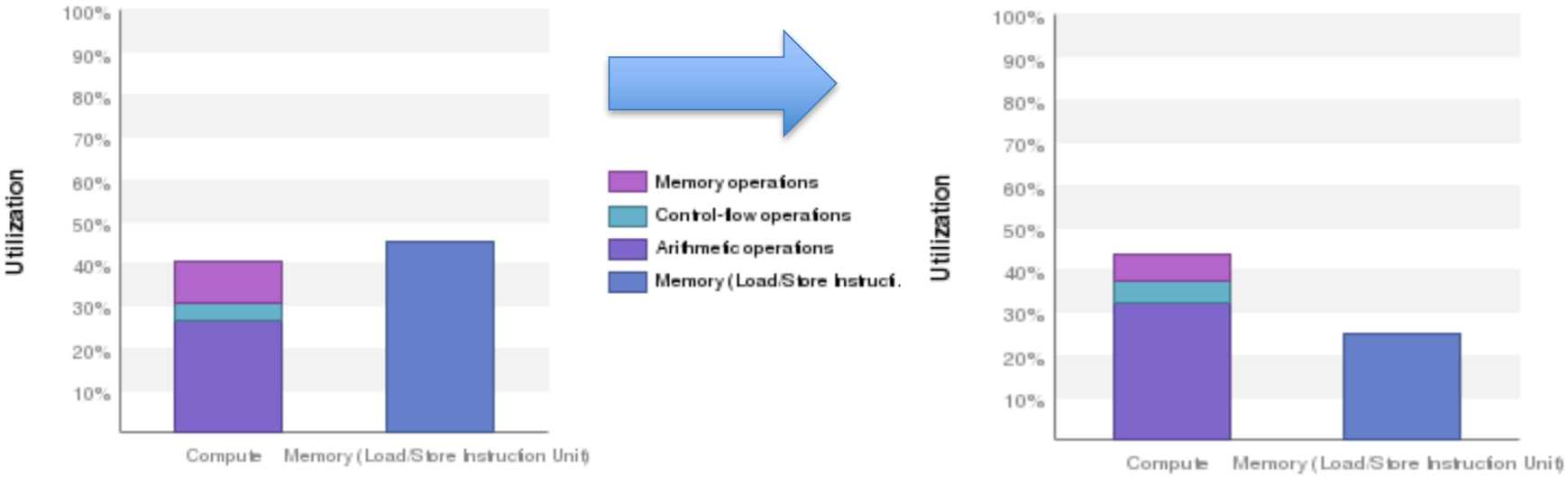
```
sumx += __shfl_down(sumx, offset, NODES_PER_BLOCK_PART);
sumy += __shfl_down(sumy, offset, NODES_PER_BLOCK_PART);
sumz += __shfl_down(sumz, offset, NODES_PER_BLOCK_PART);
poten +=  shfl down(poten, offset, NODES PER BLOCK PART);
```

  - Some __syncthreads() can be removed due to threads not having to wait for all threads to read or write to shared memory



Reducing share memory

- By using less shared memory we lowered the memory utilization as expected but did not improve the compute utilization…. We are still Latency limited!



- Register usage could be the limiting resources.

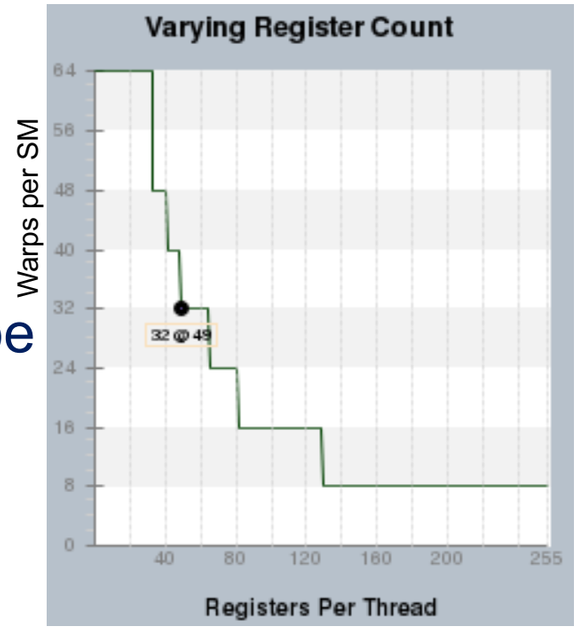
Reducing registers usage

- 56 registers per thread was being used or 14336 registers per block
- Tesla K20X is configured to have up to 65536 registers per SMX
- Each SMX was limited to simultaneously execute only 4 blocks (32 warps) out of the possible 16 block (64 warps)
- No direct way of controlling register usage, but we can help the compiler to do a better job.
- What to do:

  __launch_bounds__(**maxThreadsPerBlock**, minBlockPerMultiProc)

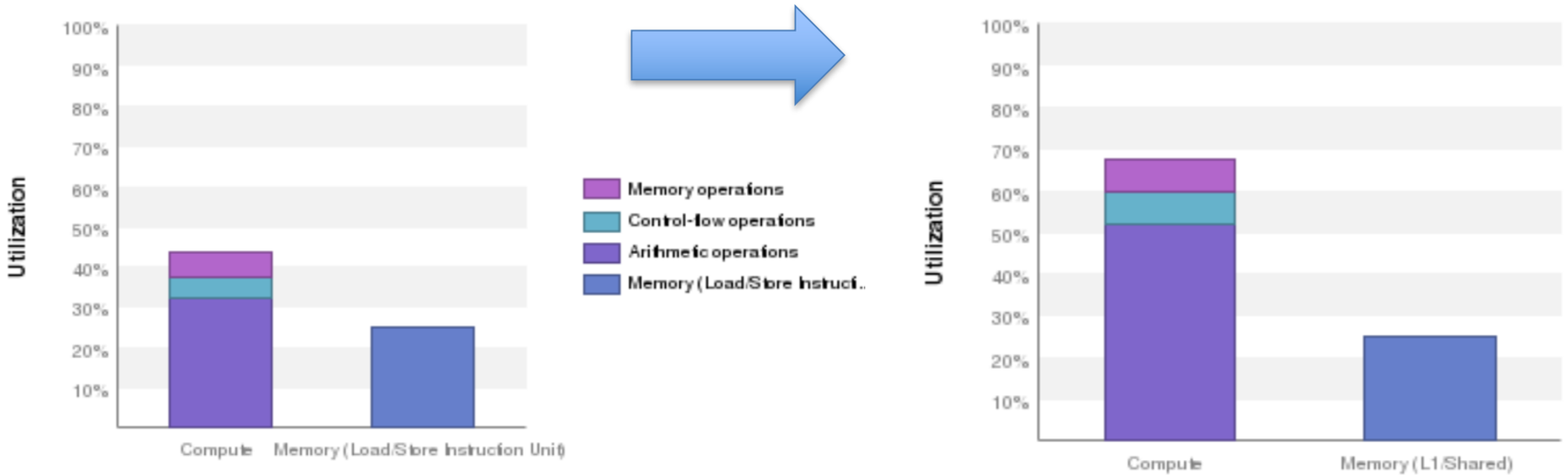
- The compiler will derive the number of register it needs per threads to be minBlockPerMultiProc*__maxThreadsPerBlock__ per SMX.

NUM_REG ↓ LOCAL_MEM ↑ NUM_INSTRUCTIONS ↑



Reducing registers usage

- Register usage decreased from 56 to 24 thus utility rose to approximately 70%



- Further reducing register usage causes spilling onto global memory adversely affecting execution time!

What does it all mean in terms of speedup

- Two kernels from ChaNGa, N-Body Cosmological application,
  - particleGravityComputation
  - nodeGravityComputation

- Both kernels are non-trivial and highly optimized making use of shared memory.

- After described latency optimizations:
  - particleGravityComputation
    - Utilization improved from about 40% to 70%
    - 1.66x speedup
  - nodeGravityComputation
    - Utilization improved from about 30% to 60%
    - 2.11x speedup

Future HPC Transition

Blue Waters vs. Future Accelerator-Dense Nodes

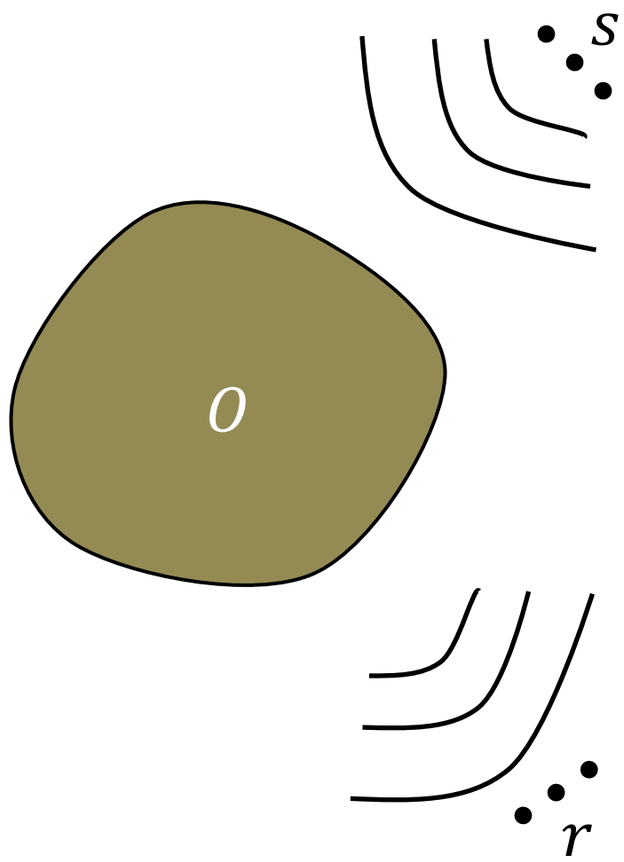
Carl Pearson (pearson@illinois.edu)

Application Background

Object hit with known field, scattered field recorded.

Solve large-scale electromagnetic wave equations numerically to reconstruct object.

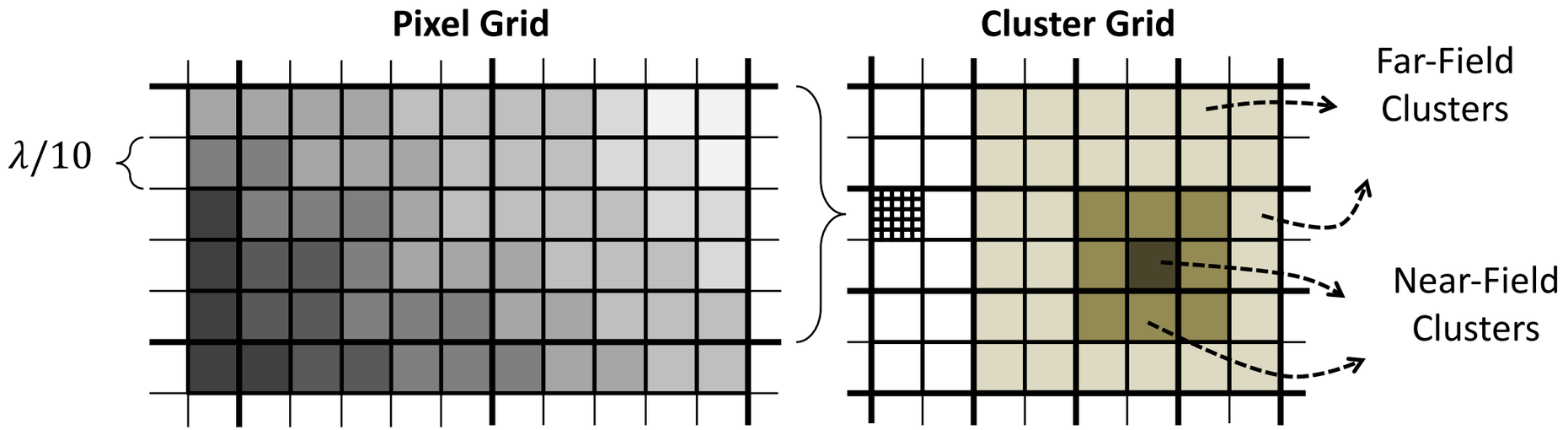
Useful for electromagnetics, acoustics, geophysics, radar, medical imaging, antenna design.



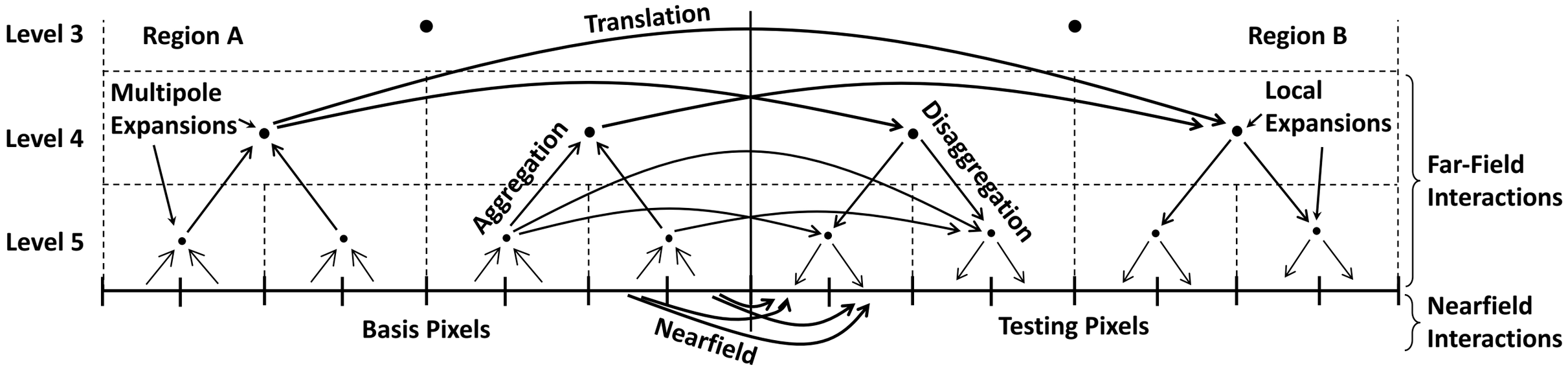
Multilevel Fast Multipole Method

Compute pairwise interactions between discretized object pixels.

Local interactions are spatially binned in a hierarchical manner to compute N^2 interactions in O(N) work.



MLFMM Schematic

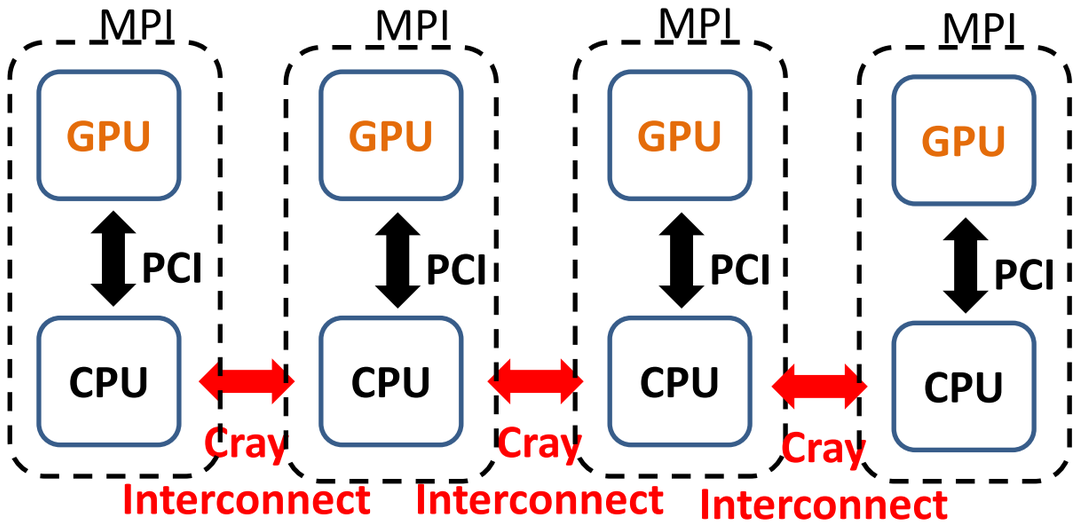


Expansions	Interpolations	Shiftings	Translations	Near-field
Dense	Band-diagonal	Diagonal	Diagonal	Sparse

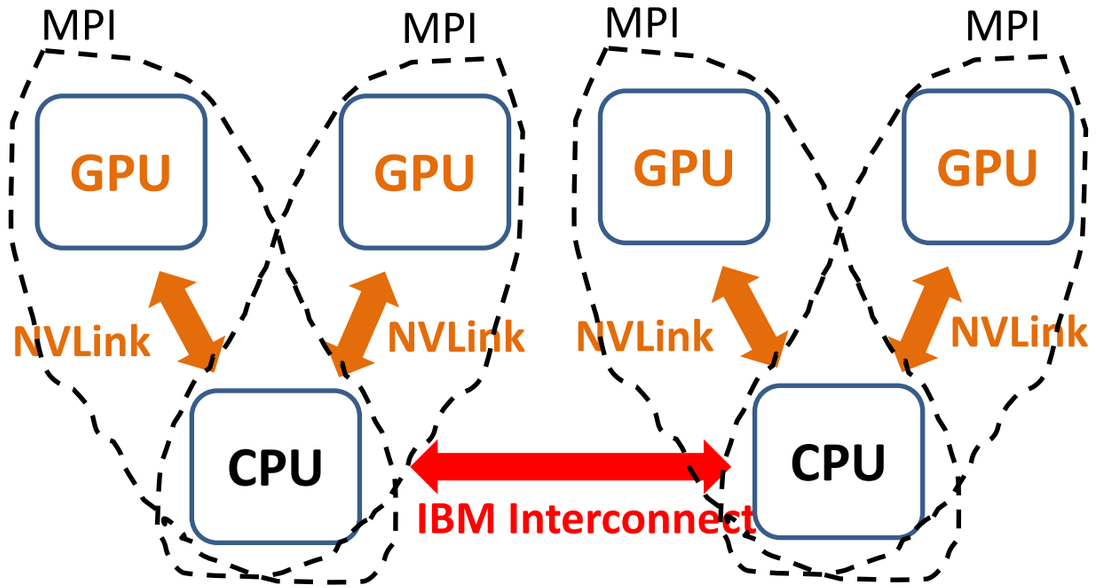
Blue Waters and S822LC System Comparison

Feature	XK6	XK7	S822LC
CPU 0	Opteron 6276	Opteron 6276	Power8
CPU 1	Opteron 6276	--	Power8
GPU 0	--	K20x	P100 SMX
GPU 1	--	--	P100 SMX
GPU 2	--	--	P100 SMX
GPU 3	--	--	P100 SMX
System RAM (CPU + GPU)	64 GB	32 GB + 6 GB	512 GB + 64 GB
System Floating-Point Units	16	8	20
System CPU Threads	32	16	160

Blue Waters and S822LC MLFMM Execution



Blue Waters



S822LC

MLFMM Performance

Sequential CPU Speedup

1T S822LC vs 1T XE: **1.17**

Multithreaded CPU Speedup

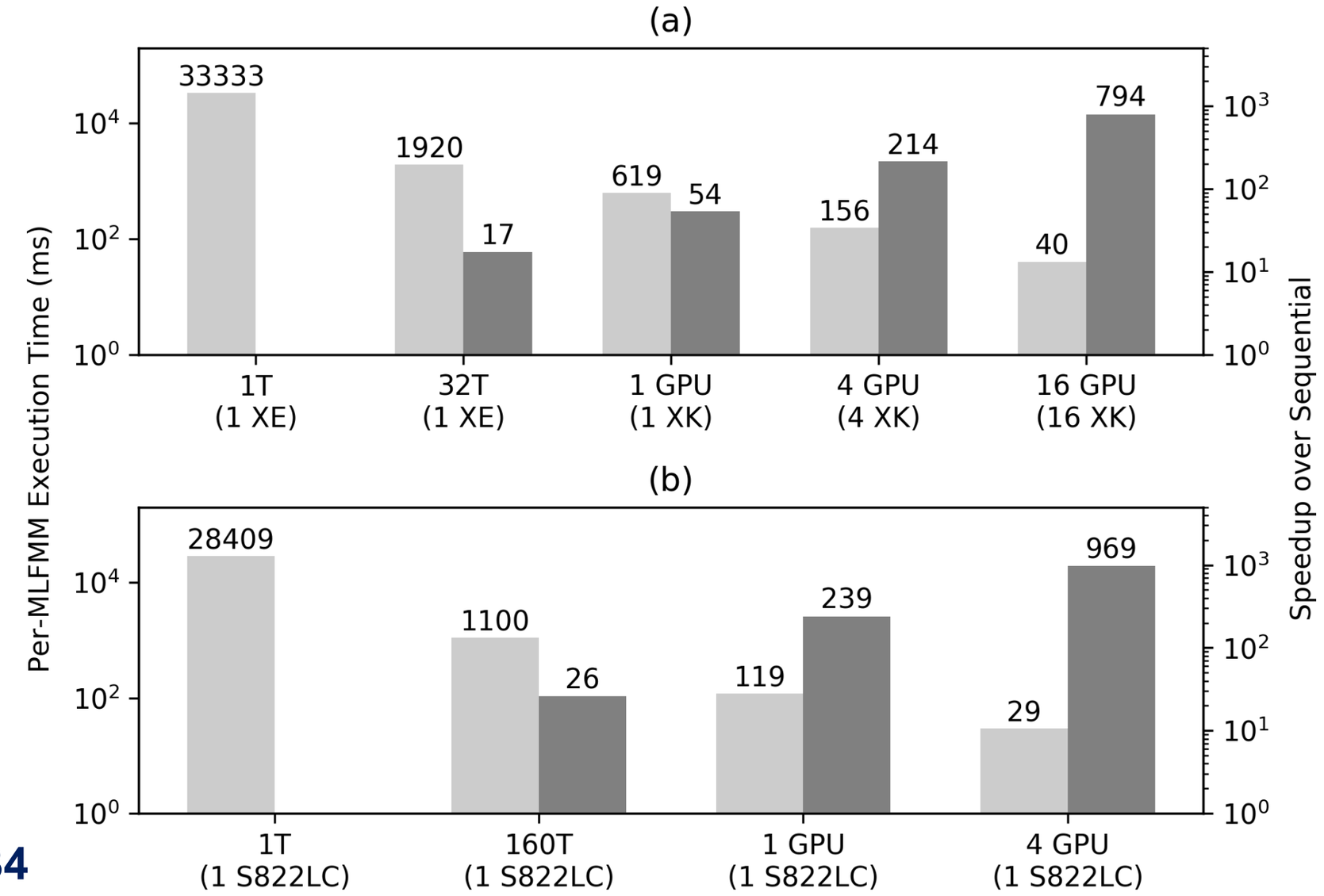
160T S822LC vs 32T XE: **1.75**

GPU Speedup

1GPU S822LC vs 1GPU XK: **5.2**

Per-node Speedup

4GPU S822LC vs 1GPU XK: **21.34**



Kepler and Pascal GPU Architectures

Feature	K20x Kepler (GK110)	P100 Pascal (GP100)
Core Clock	732 MHz	1328 MHz
Global Memory Bandwidth	250 GB/s	720 GB/s
Peak GFLOPs (single / double)	3935 / 1312	9519 / 4760
L2	1.5 MB	4 MB
# of SMs	14	56
Register File Size	256 KB	256 KB
L1	48 / 32 / 16 KB	0 KB
Shared Memory	16 / 32 / 48 KB	64 KB
"CUDA Cores"	192	64
Max Resident Blocks	16	32

Normalized Kernel Executions

	Blue Waters	S822LC	Speedup
L2L Kernel Time	78.5 ms	9.9 ms	8.0x
MLFMM Time	633 ms	119 ms	5.3x

L2L Occupancy	Blue Waters	S822LC
Theoretical	43.8	56.2
Achieved	30.7	42.1

Occupancy limited by shared memory

Relative performance improves due to increased shared memory size.

Observations

- Easier to fully utilize CPU FP without tuning performance of each thread (oversubscription)
- Lots of direct speedup on existing code
- Some kernels can be tuned for even more

Questions