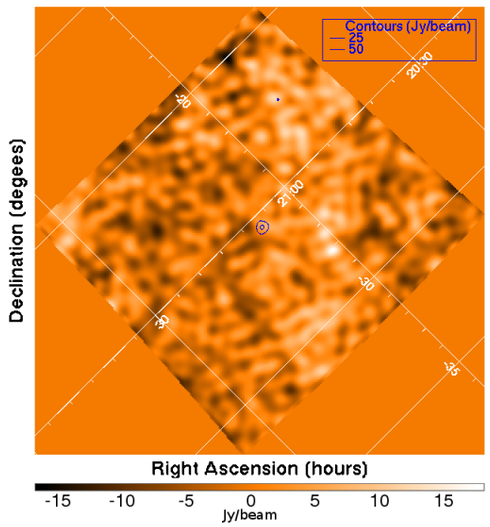
Using GPUs for Signal Correlation



Michael Clark

with

Paul La Plante and Lincoln Greenhill

Outline

- Motivation
- Mapping the X-engine onto a GPU
  - Or how to get a sustained TFLOP from a Fermi
- Comparison
- Summary and Conclusions

Dipole Array Signal Processing

Heterogeneous HPC solution

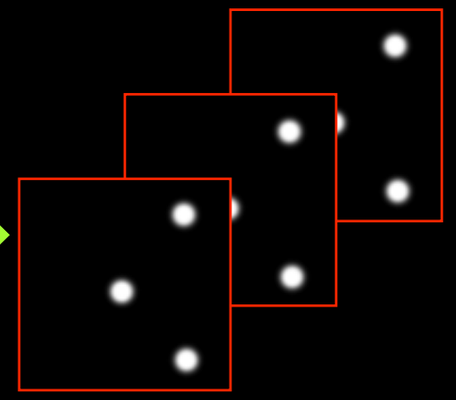
RF Samplers

GPU/FPGA Real-Time

Correlator

GPU/CPU R-T, postR-T

Calibration & Imaging



N

O(N)

O(N)

O(N²)

O(N)

O(NlogN)

O(N²)

digital

This talk

Last talk

Dipole arrays break the assumptions relied upon in operation of traditional arrays

Correlator

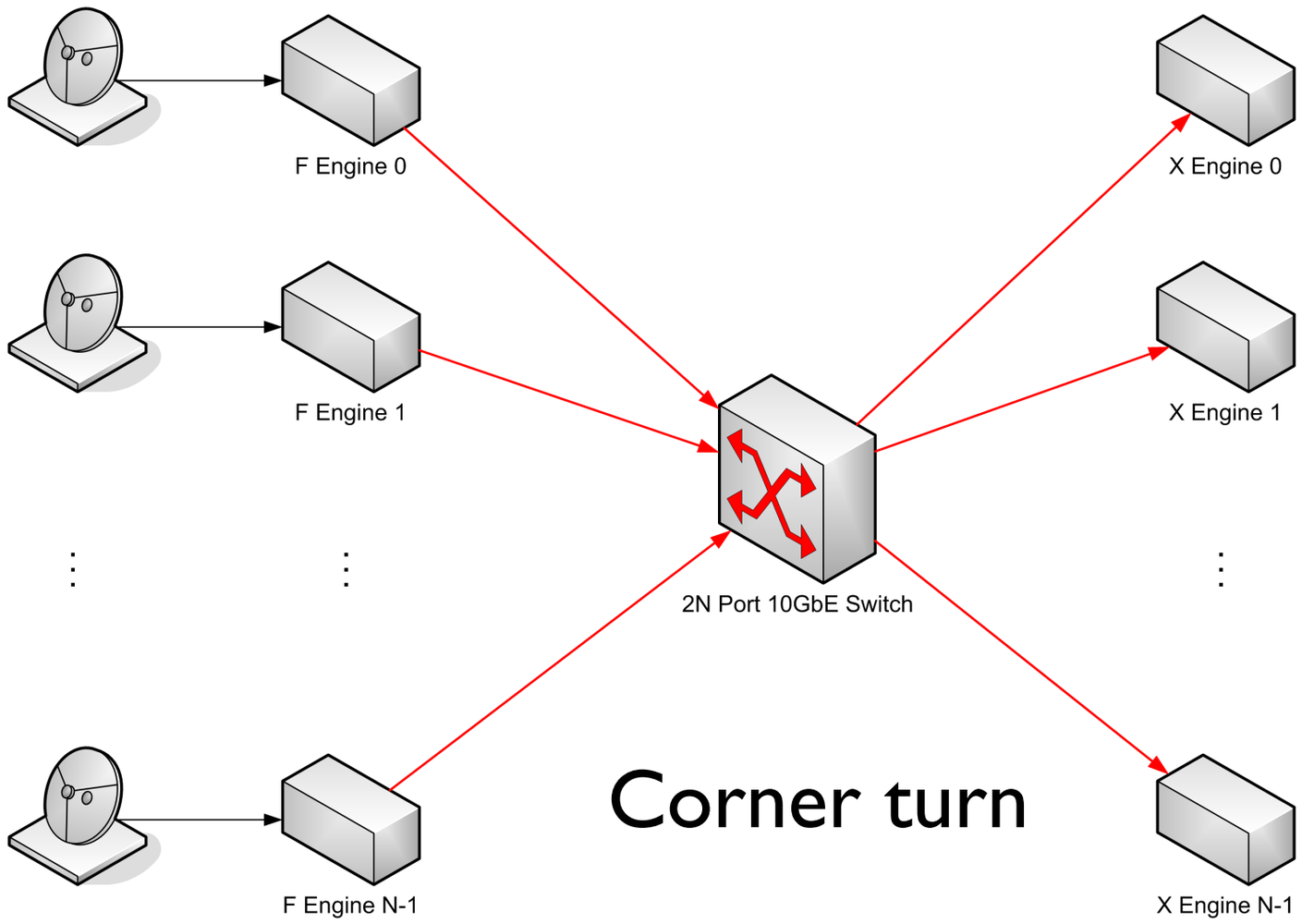
- Builds the cross-power spectrum of the sky

$$S_{ij}(\nu) = \int_{-\infty}^{\infty} (A_i \star A_j)(\tau)\, e^{-i2\pi\nu\tau} d\tau$$

- Cross-correlate the signal from every station pair
- XF Correlator
  - Cross correlate the signal then FFT
- FX Correlator
  - From convolution theorem $\mathcal{F}(A \star B) = (\mathcal{F}A) \times (\mathcal{F}B)$
  - FFT the signal then cross-multiply $O(B\ N_s\,(N_s + k))$

FX Correlator

(Figure taken from Casper collaboration)

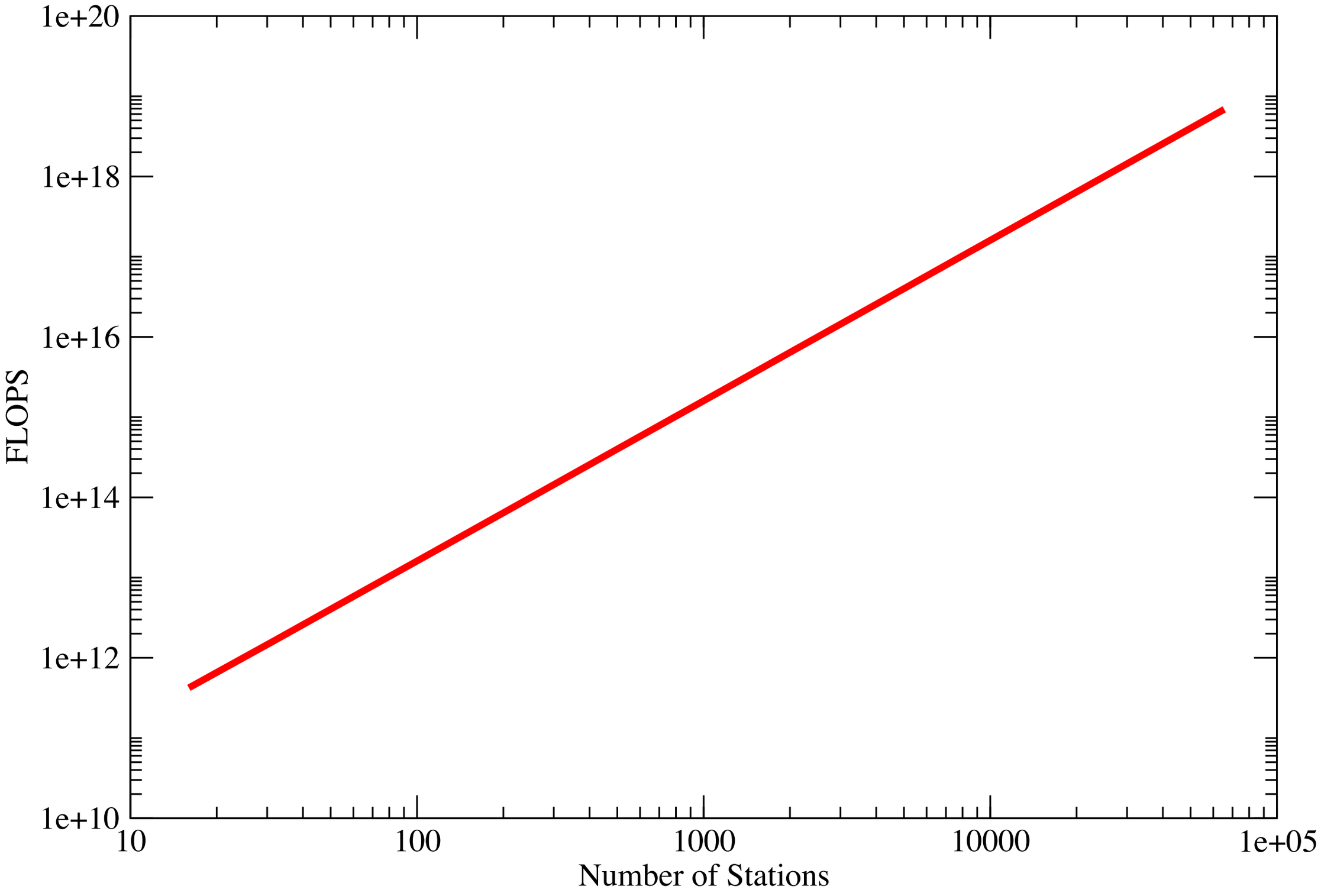


F-engine compute scales linearly with N_s

X-engine compute scale quadratically with N_s

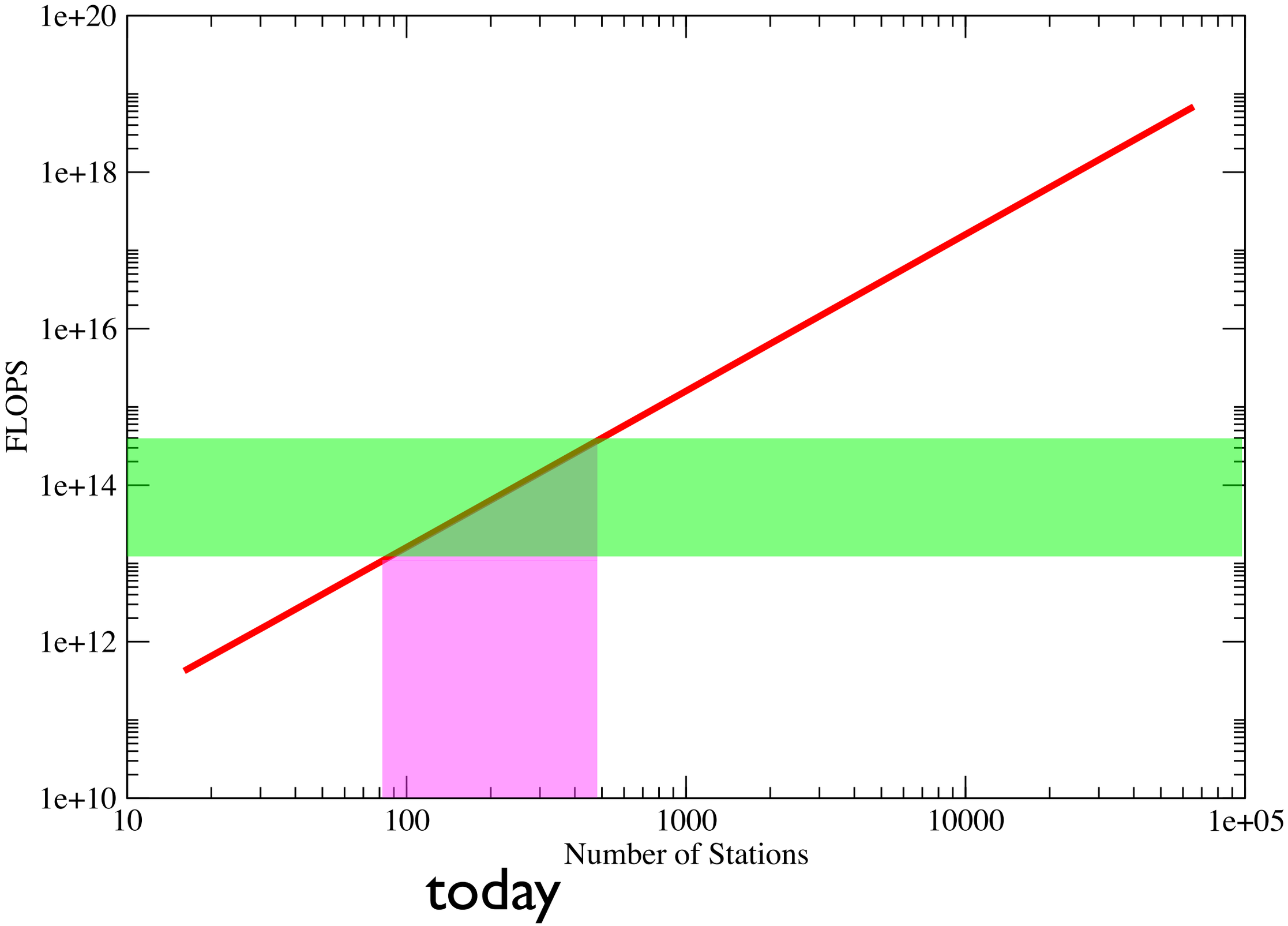
X-Engine Computing Requirements

$N_c = 1000, B = 100$ MHz



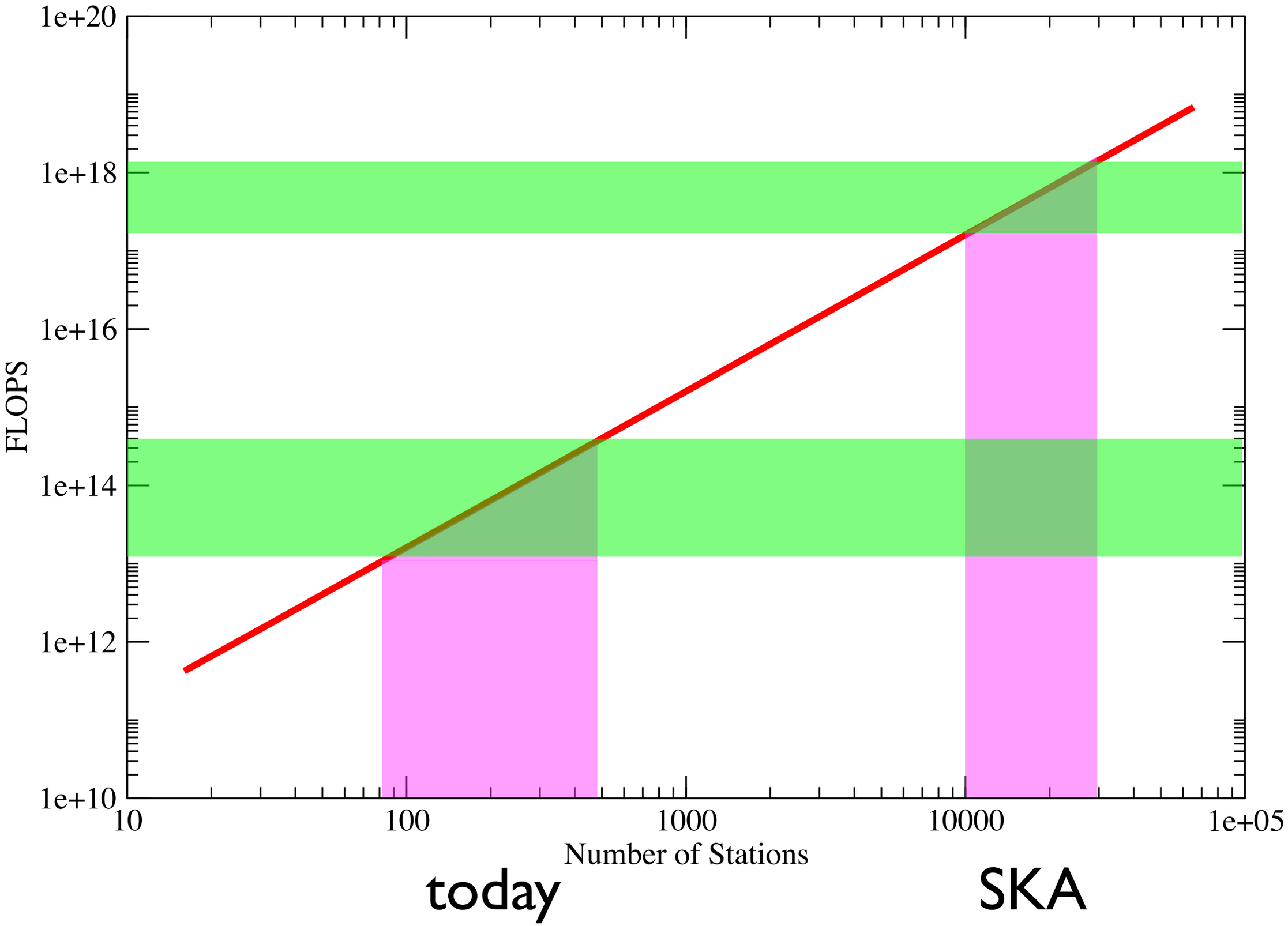
X-Engine Computing Requirements

$N_c = 1000, B = 100$ MHz



X-Engine Computing Requirements

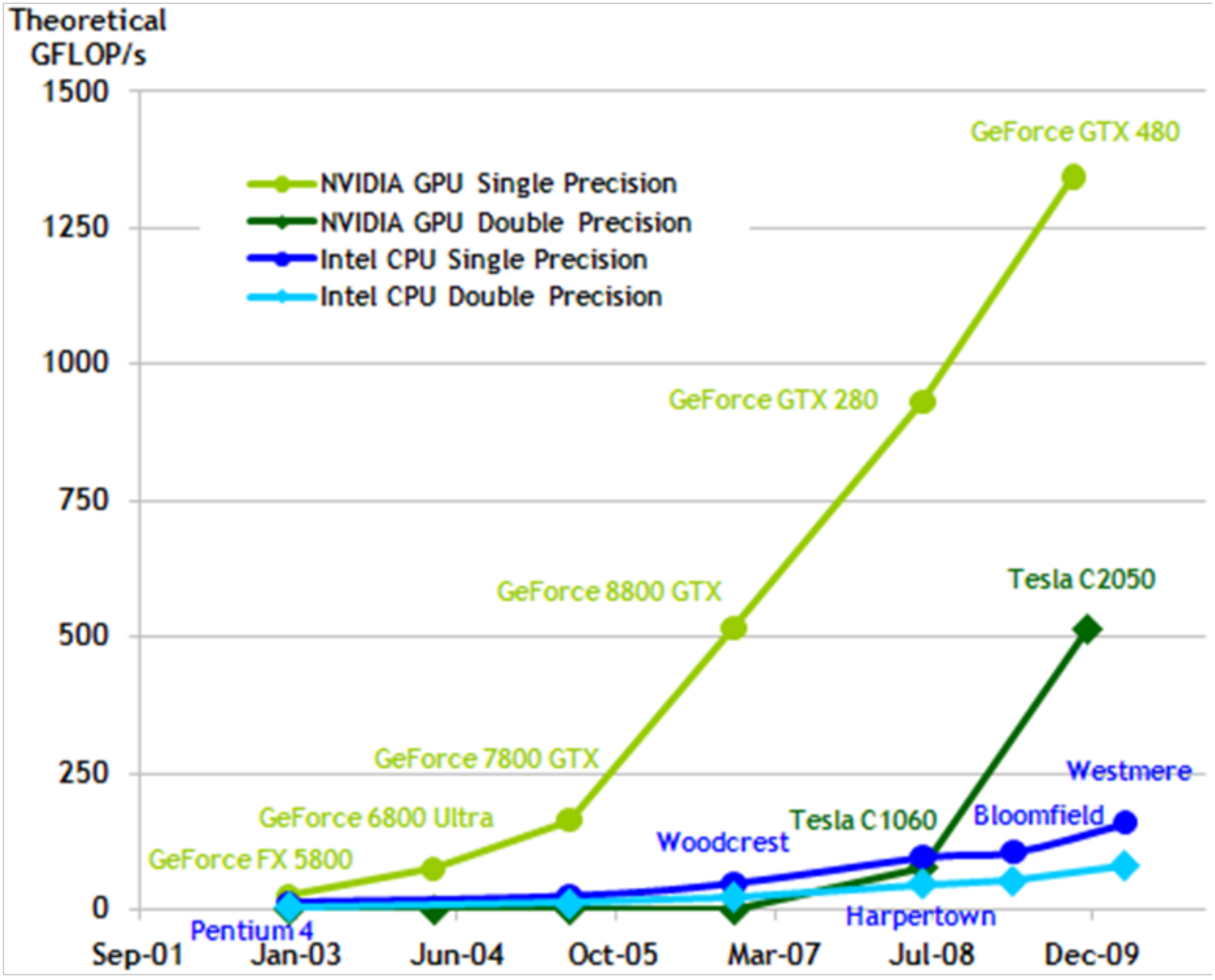
$N_c = 1000, B = 100$ MHz



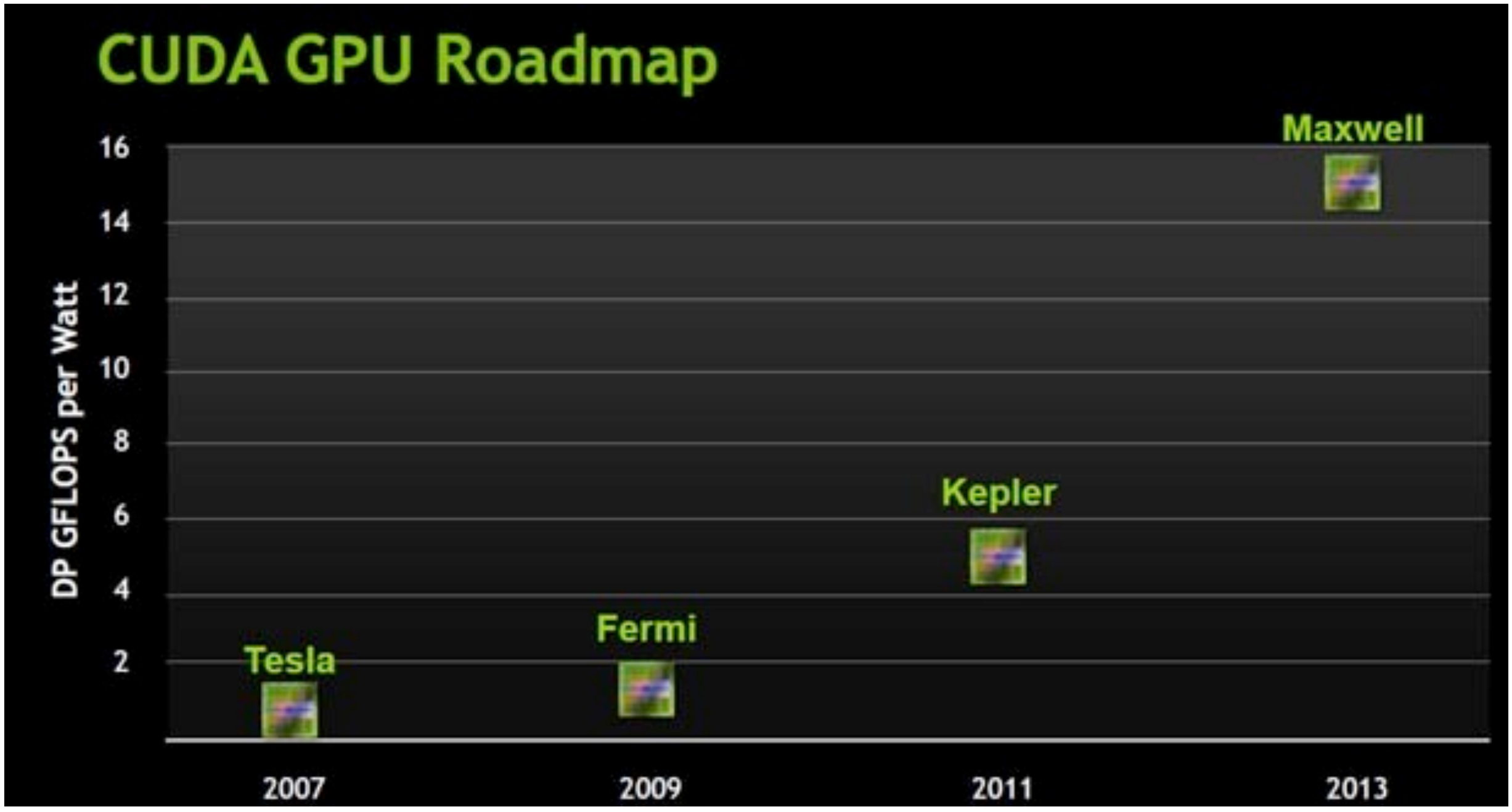
FPGAs

- FPGAs (or ASICs) generally used for correlation
- Ideal since only require limited fixed-point ops
- Very power efficient since all die area devoted to problem
- Extremely expensive
  - Astronomers survive on donations and old h/w
- Development time can be very long
- Local experts at Berkeley

http://casper.berkeley.edu/wiki/CASPER_Correlator



Cuda Programming Guide



Huang's GTC Keynote

Are GPUs (part of) the solution?

Motivation

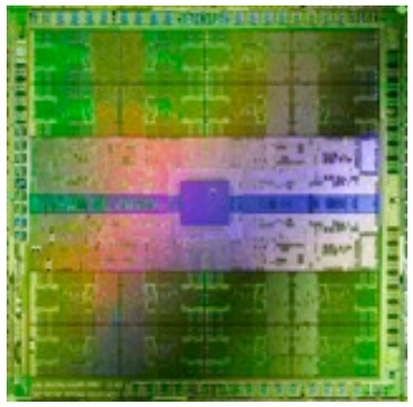
- GPUs are an interesting platform
  - Harness commodity hardware
  - Low cost
  - Easy to develop
  - Forwards compatible
- How competitive are GPUs?
  - Vs. other commodity hardware
  - Vs. FPGAs
  - Testcase: MWA Correlator
- Recall Nbody with GRAPE

Previous Work

- Harris *et al* (2008)
  - GPU for X-engine only - no consideration for integrated system
- Wayth *et al* (2009)
  - Consider GPU for both F-engine and X-engine
  - Developed for the 32 station MWA prototype
  - Not a scalable solution
- van Nieuwpoort *et al* (2009)
  - Compared X-engine performance across range of platforms
  - Claim: GPU implementations suffer from memory bottlenecks
  - Conclusion: BG/P and Cell optimal
- All of the above had low percentage of peak performance for GPUs

MWA Correlator

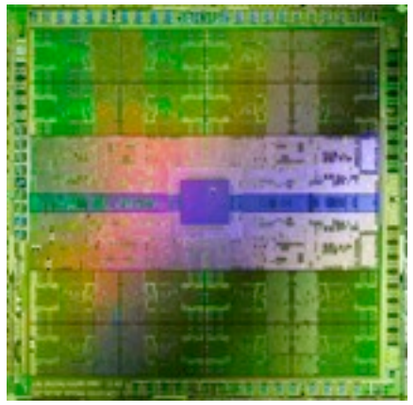
- Array located in Western Australian Outback
  - Total power budget ~ 50 kW
- Tasked with detecting EOR signal
- 512 stations x 2 polarizations = 1024 inputs
- total bandwidth = 31 MHz
- X-engine requires
  - 1.6×10^{13} CMACs = 128 TFLOPS
- How many GPUs required?



Fermi Architecture

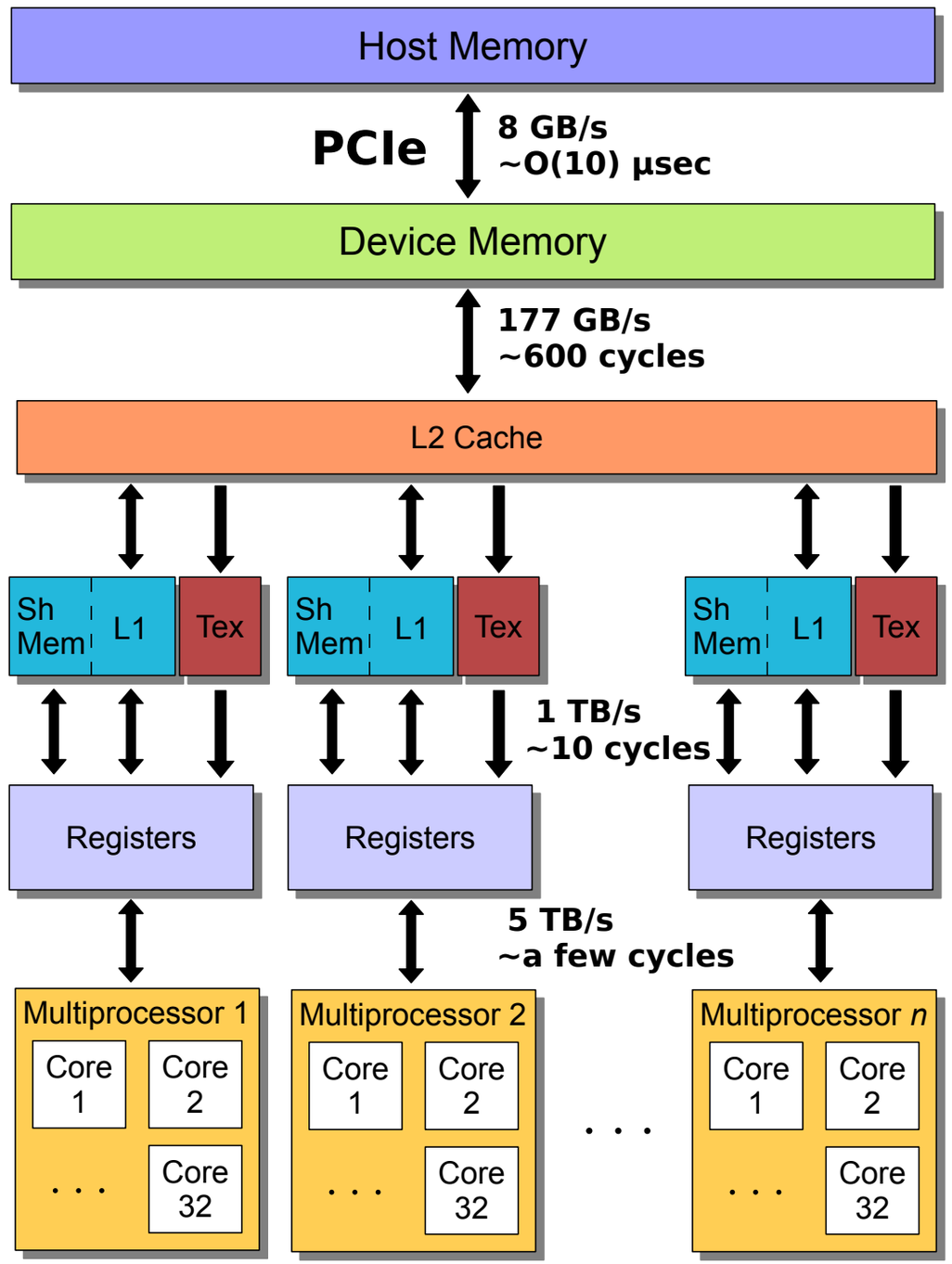
GeForce GTX 480

- 480 processing cores
- 1.345 TFLOPS SP
- 177 GB/s memory bandwidth
- Power consumption 250 Watts
- Easy to program
- $300 (as of yesterday)



Fermi Architecture

- High memory bandwidth
- BUT high flop:byte ratio 7.6:1
- Need high arithmetic intensity to keep GPU busy
- Use memory hierarchy
  - Registers
  - Shared memory
  - Device memory
  - PCIe bus

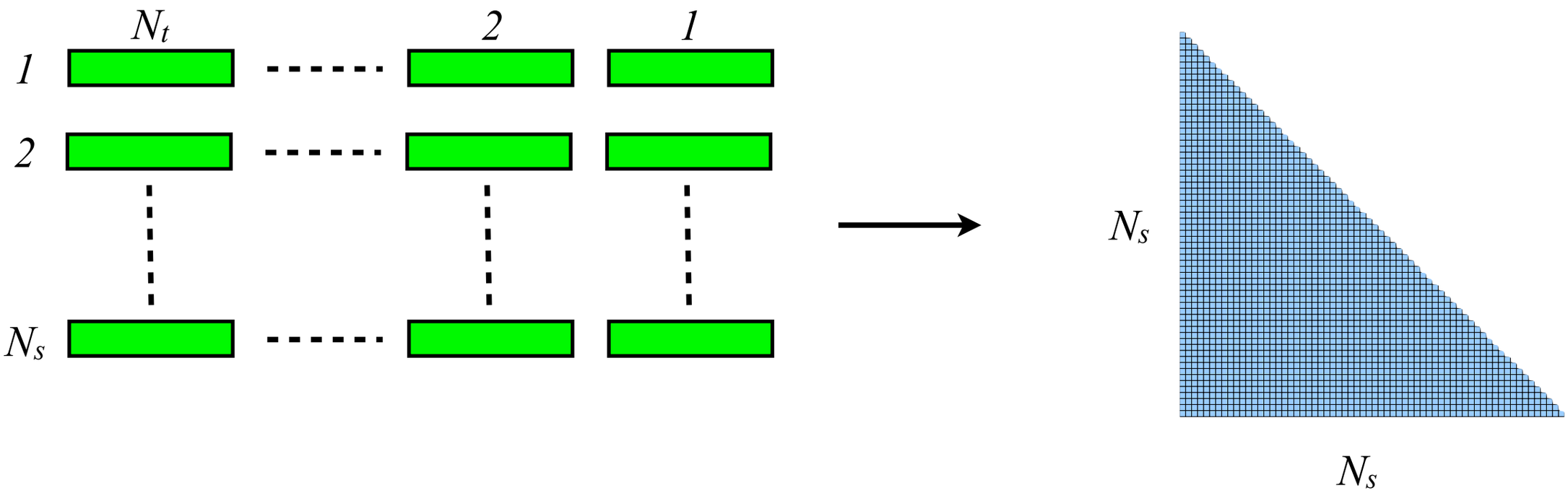


The X-engine

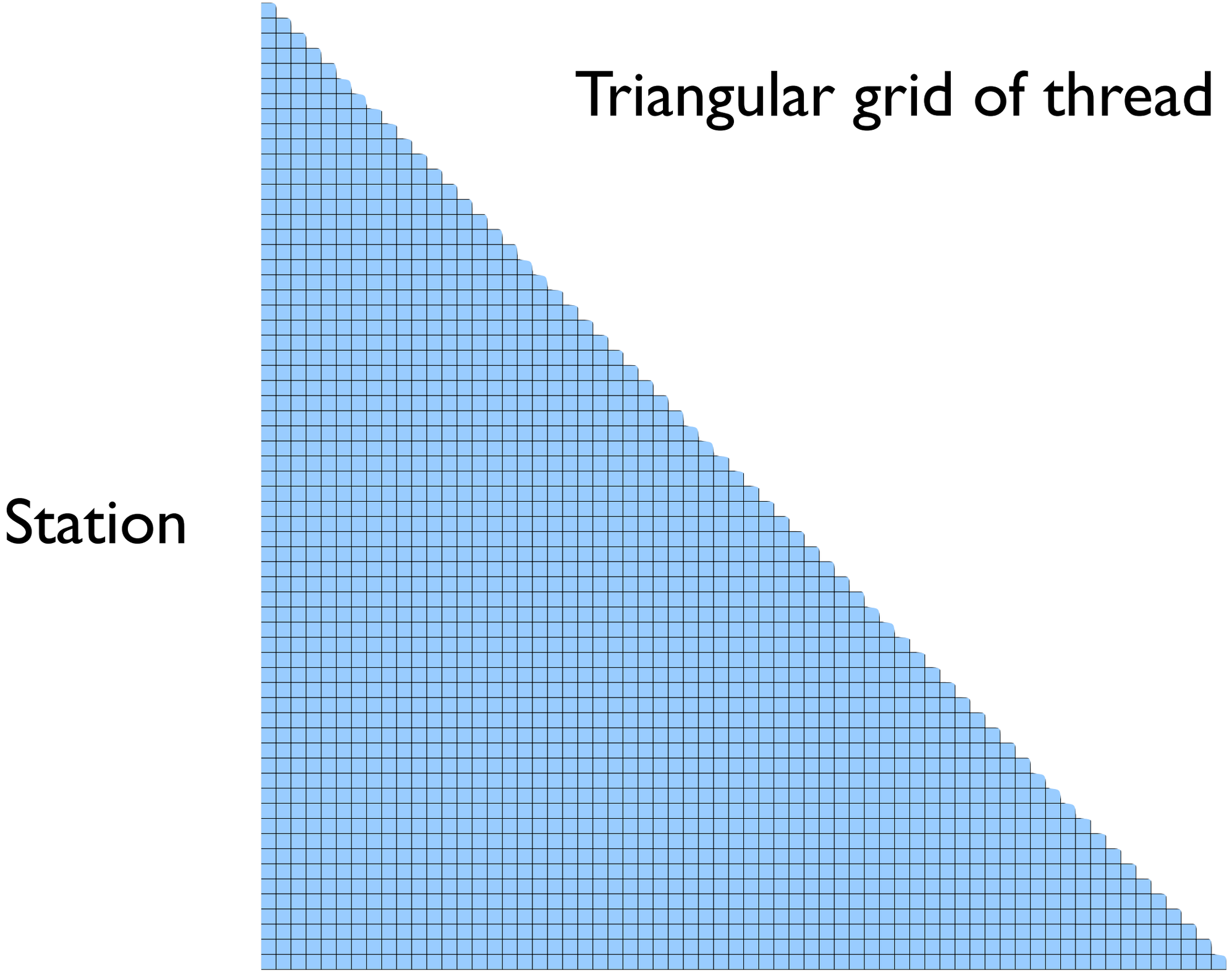
- Mathematically, just the sum of a series of vector outer products

$$S_{ij}(\nu) = \sum_{t=1}^{N_t} X_i(\nu) X_j(\nu)^\dagger \qquad \text{FLOPS} = \frac{1}{2} 8B(2N_s)(2N_s+1)$$

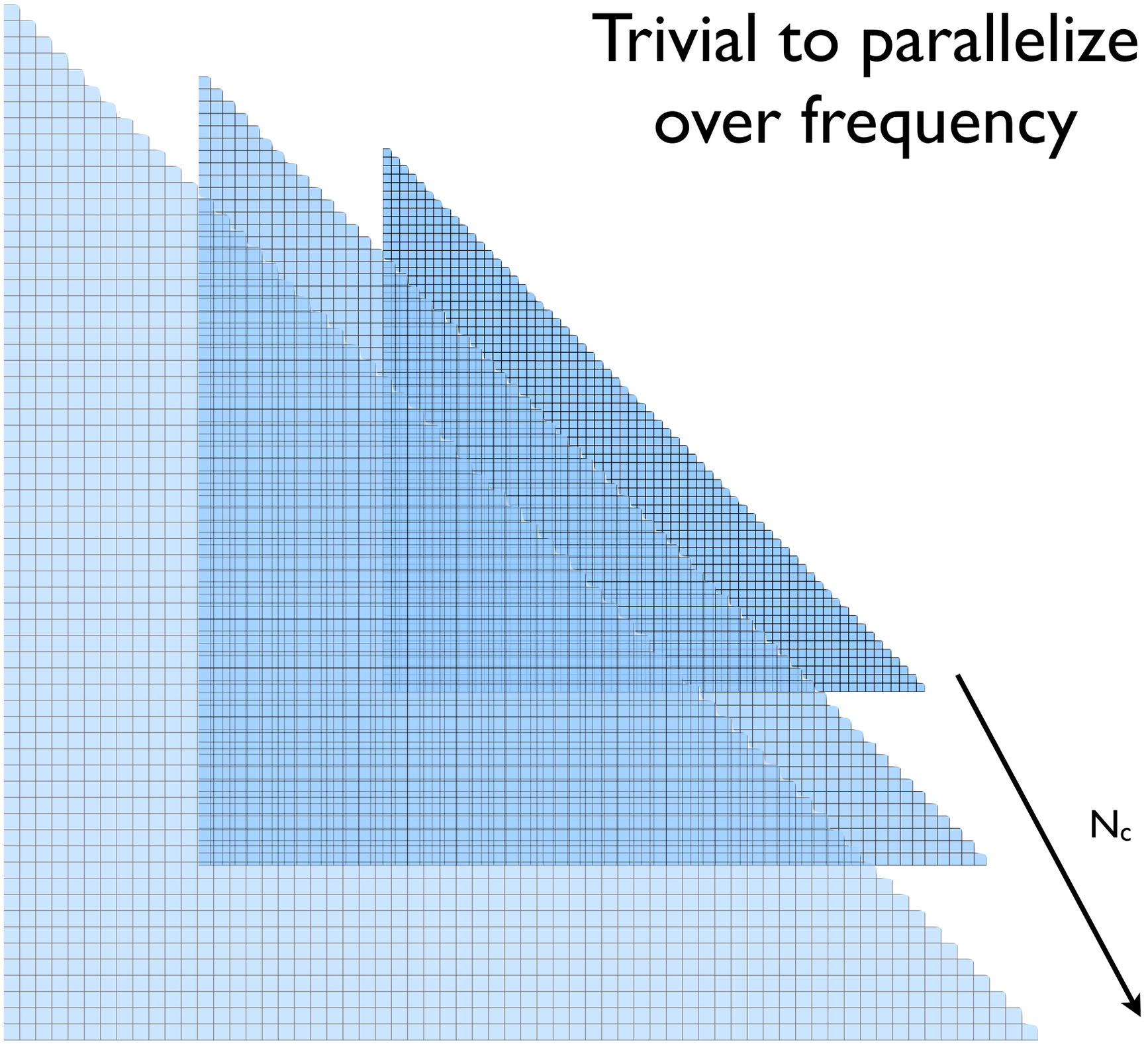
- Nt=B/T is the integration length, e.g. ~100000
- Matrix is Hermitian - just calculate lower triangular elements



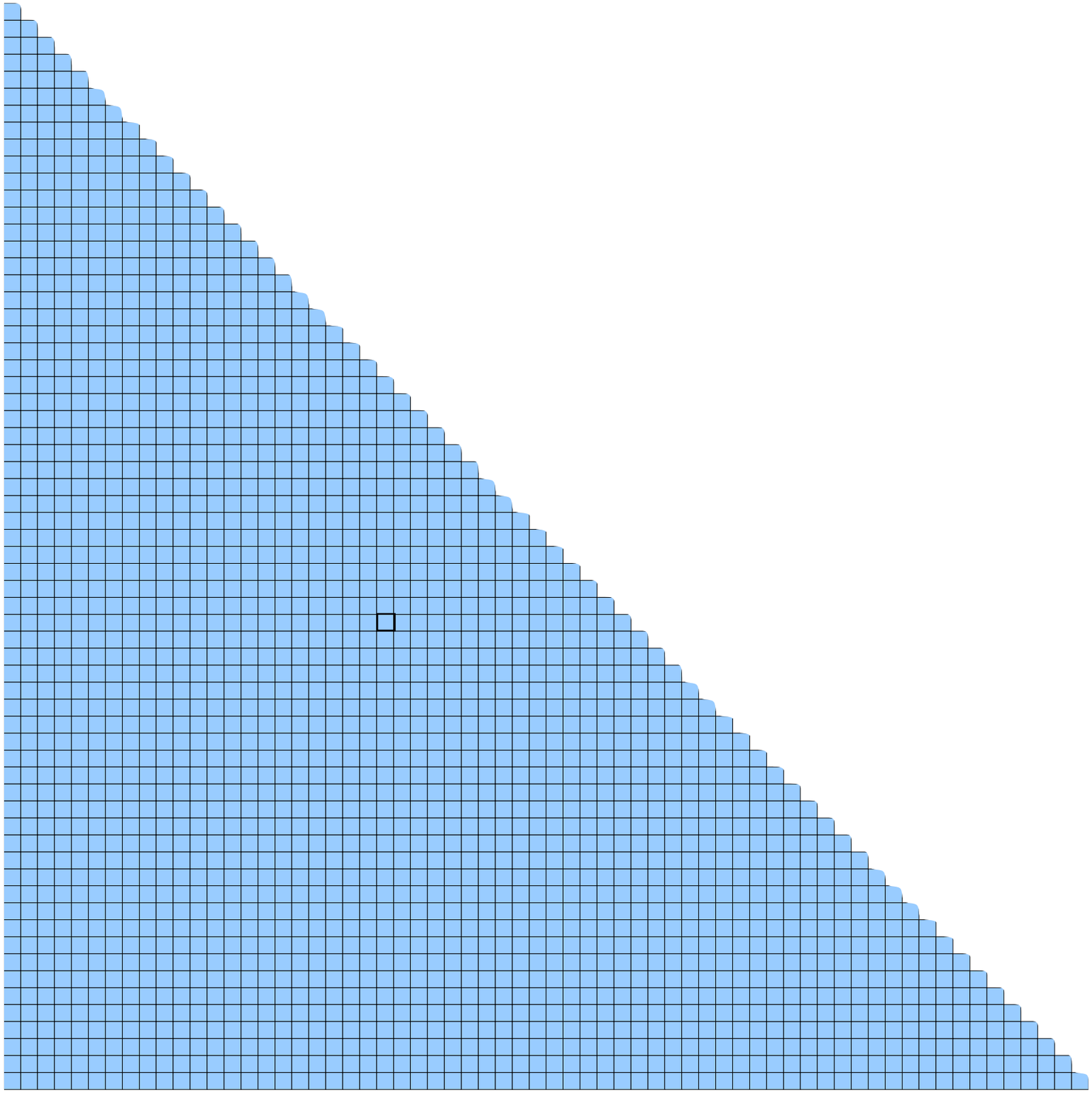
Triangular grid of thread blocks



Trivial to parallelize over frequency



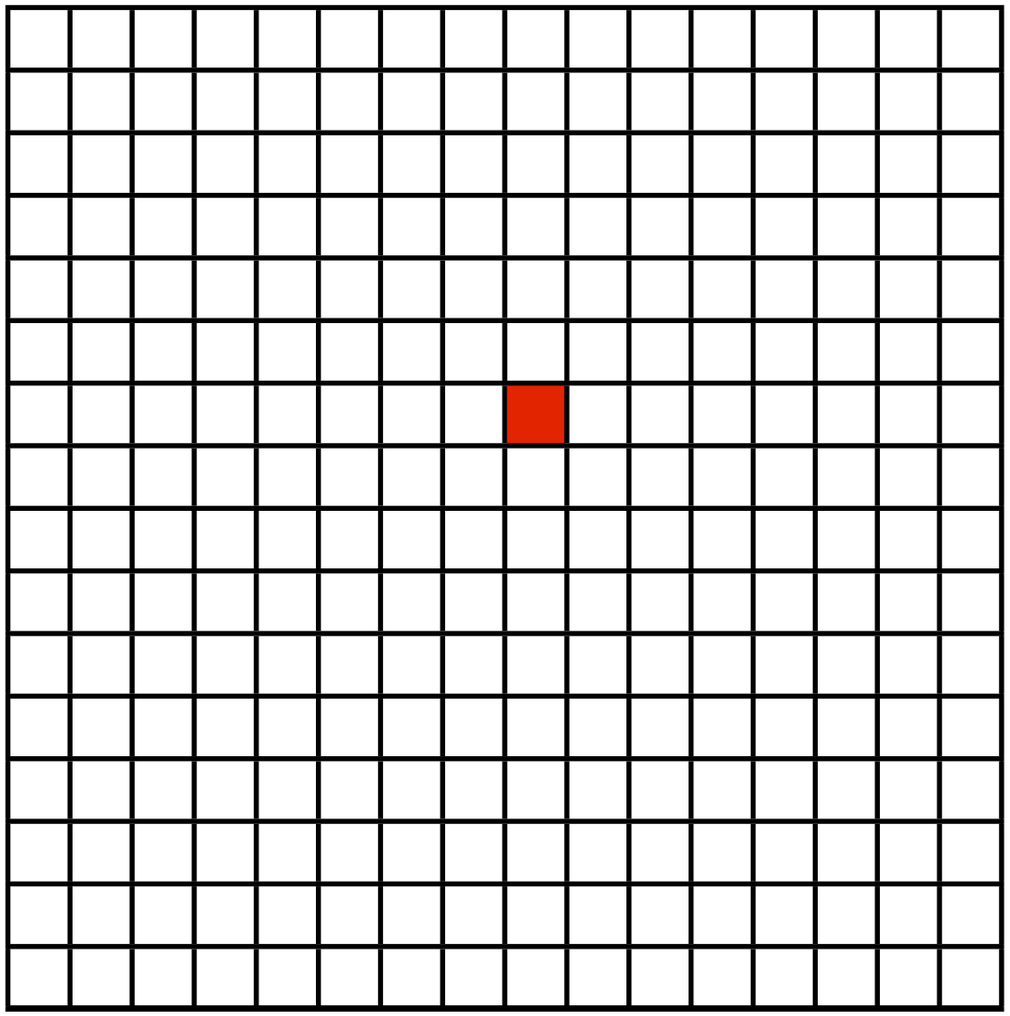
Station



Station

16x16 threads = 16x16 stations

Matrix elements stored in registers



Output

Each thread outputs its 1x1 tile to memory

flop/byte
Algorithm: $1/N_t$
Hardware: 7.6

Registers

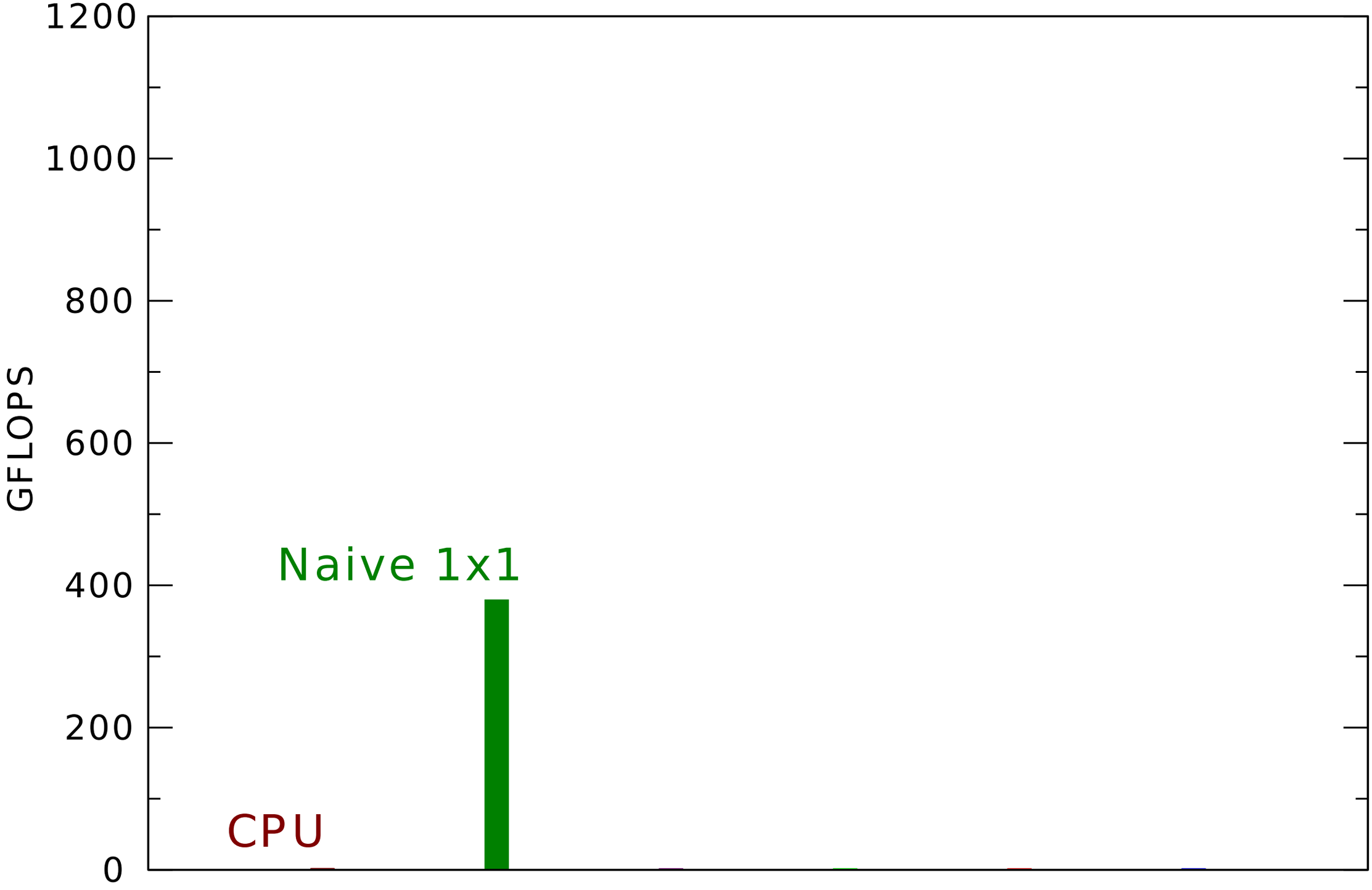
Each thread computes a 1x1 tile

flop/byte
Algorithm: 1
Hardware: 7.6

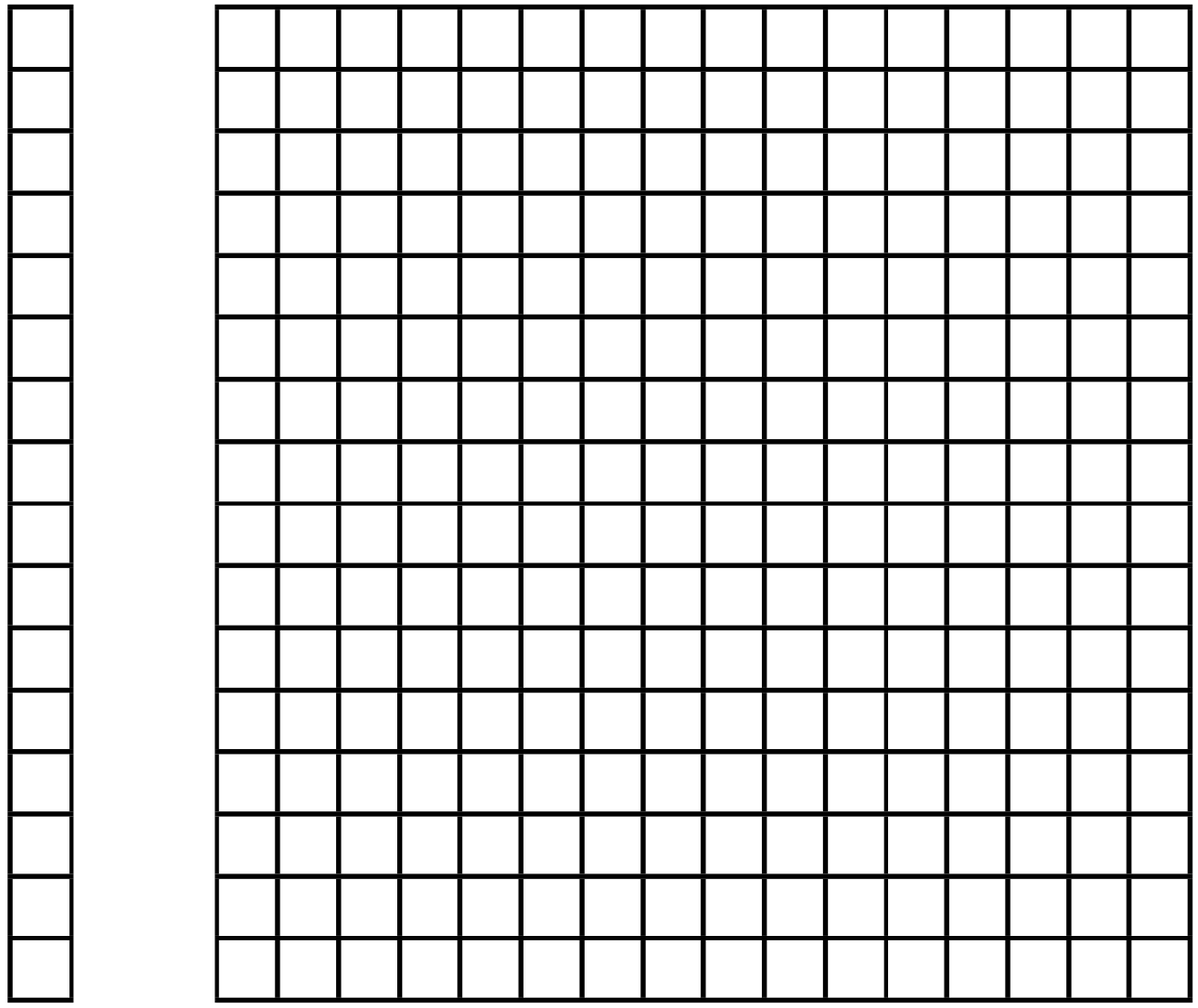
Cache

Data shared between threads using L1 cache

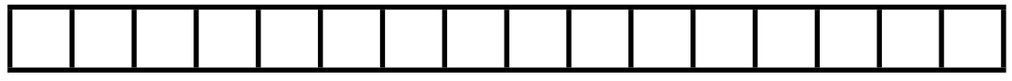
Performance for N_s=512, N_c=12, N_t=1000



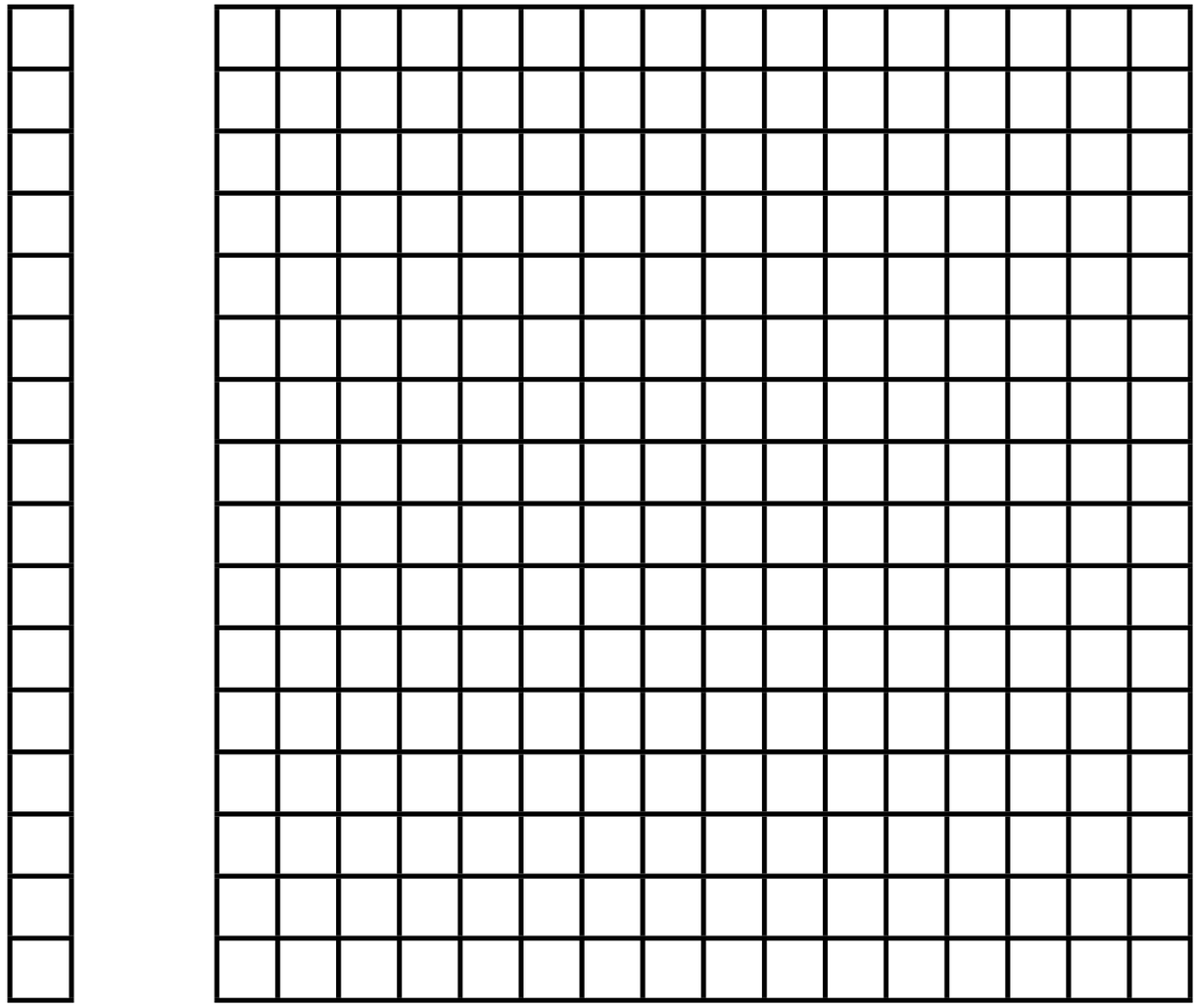
Matrix elements stored in registers



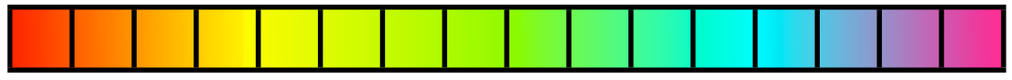
shared memory



Matrix elements stored in registers

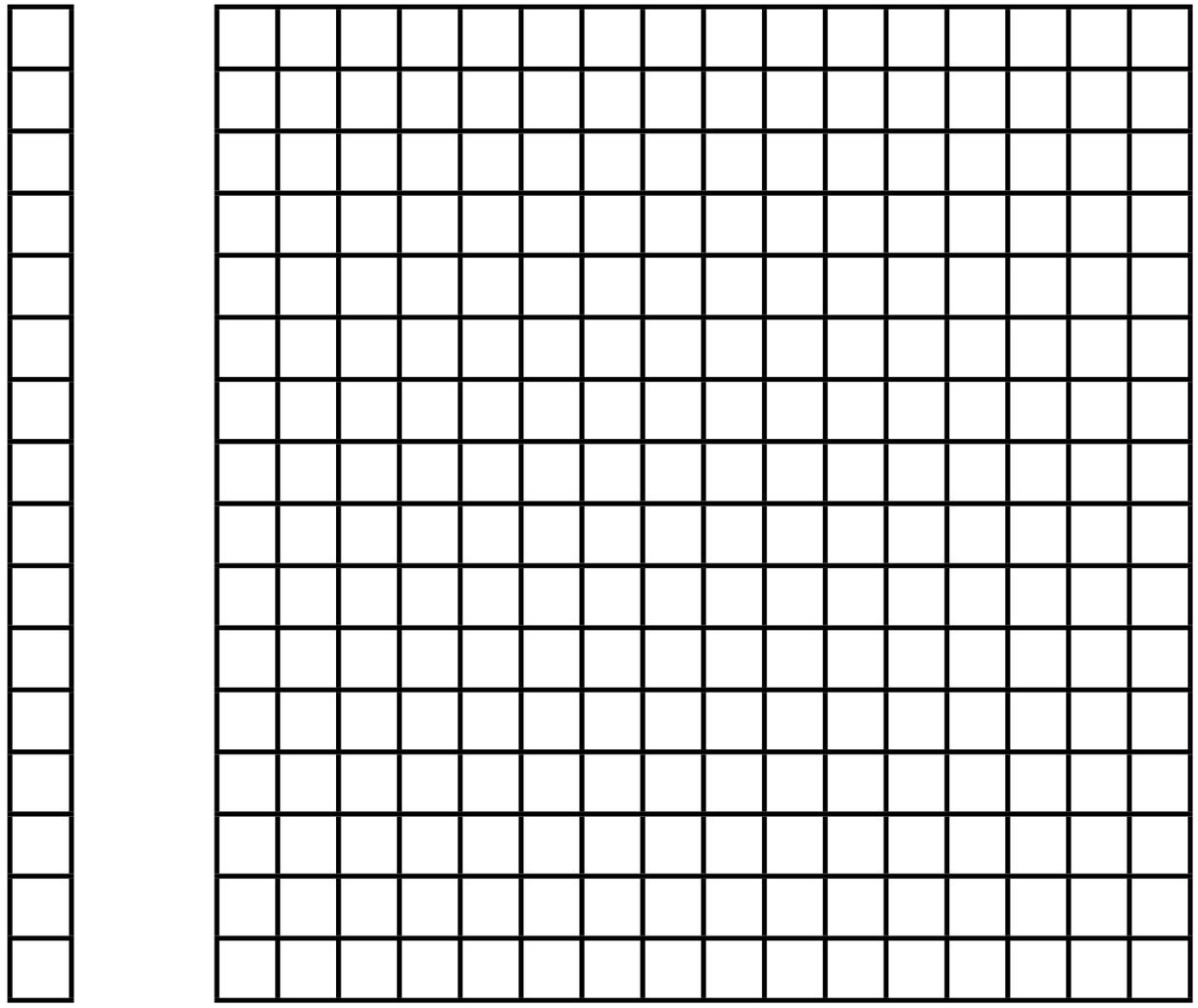


shared memory

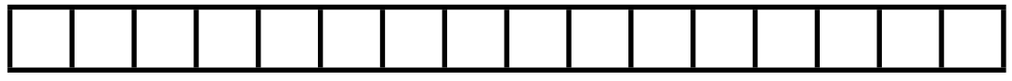


1st warp reads row

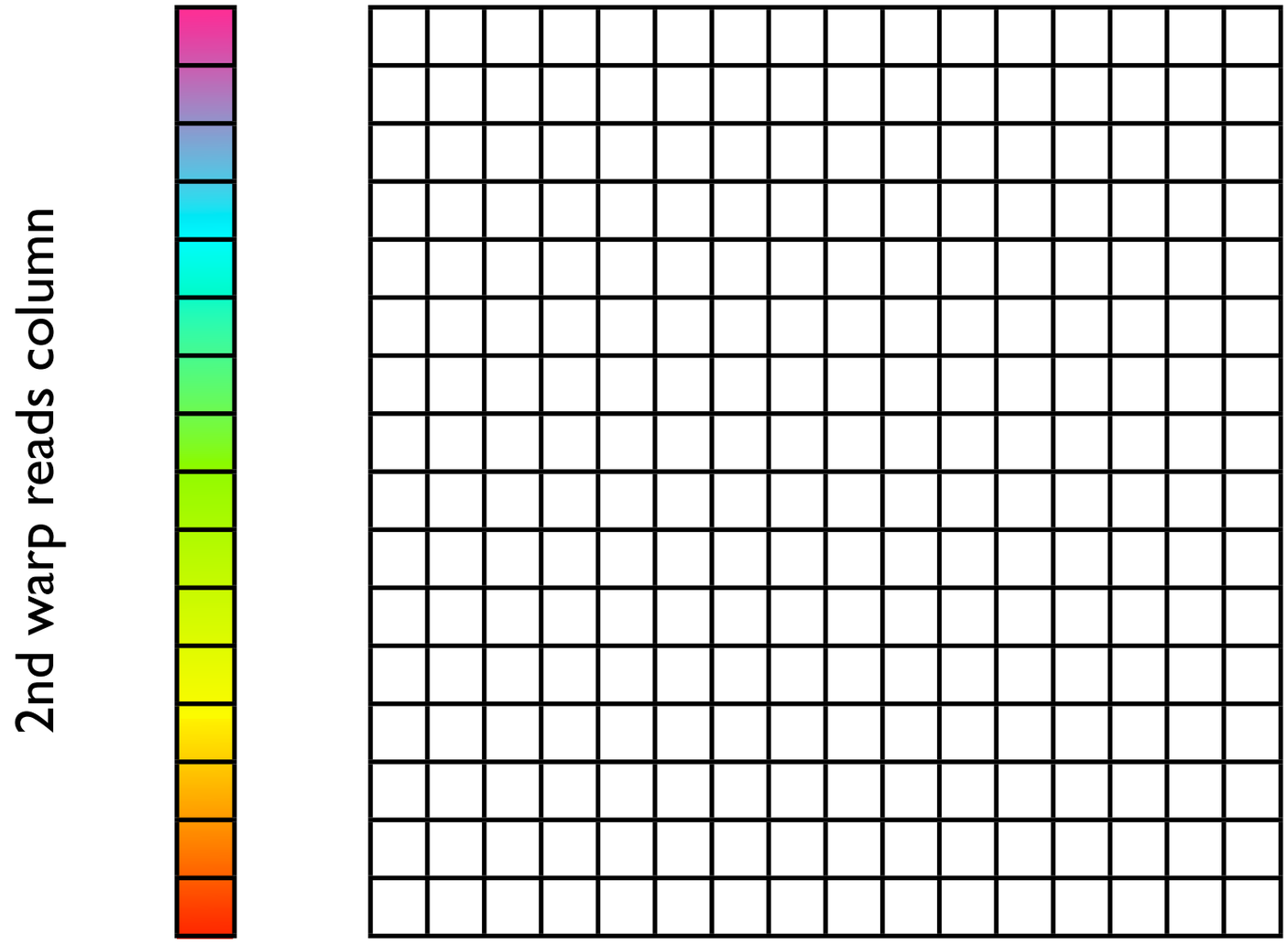
Matrix elements stored in registers



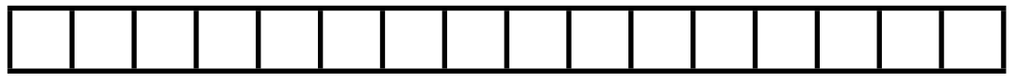
shared memory



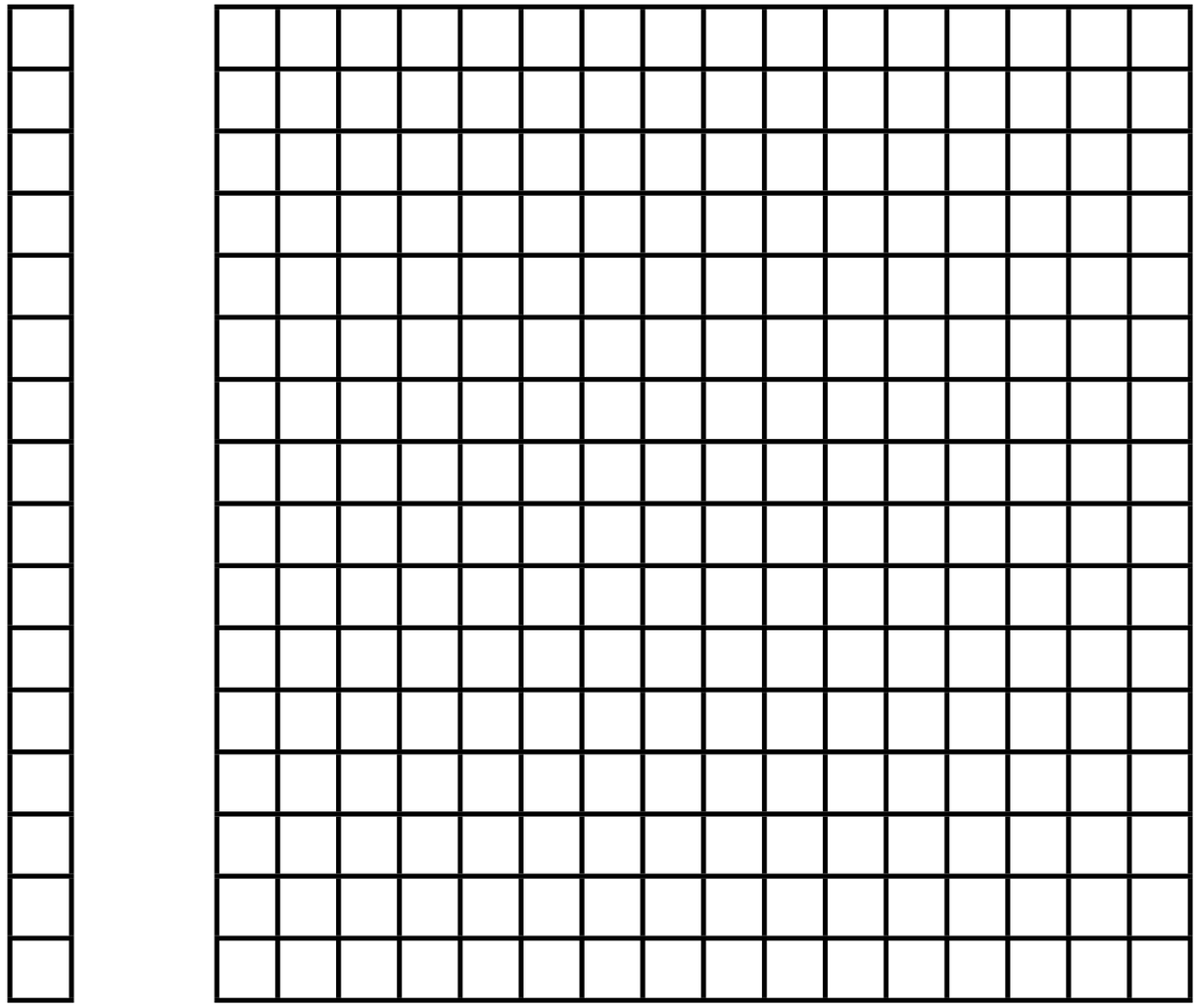
Matrix elements stored in registers



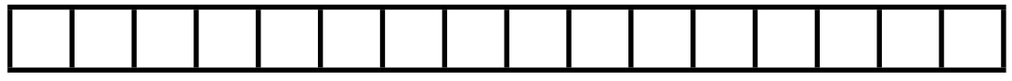
shared memory

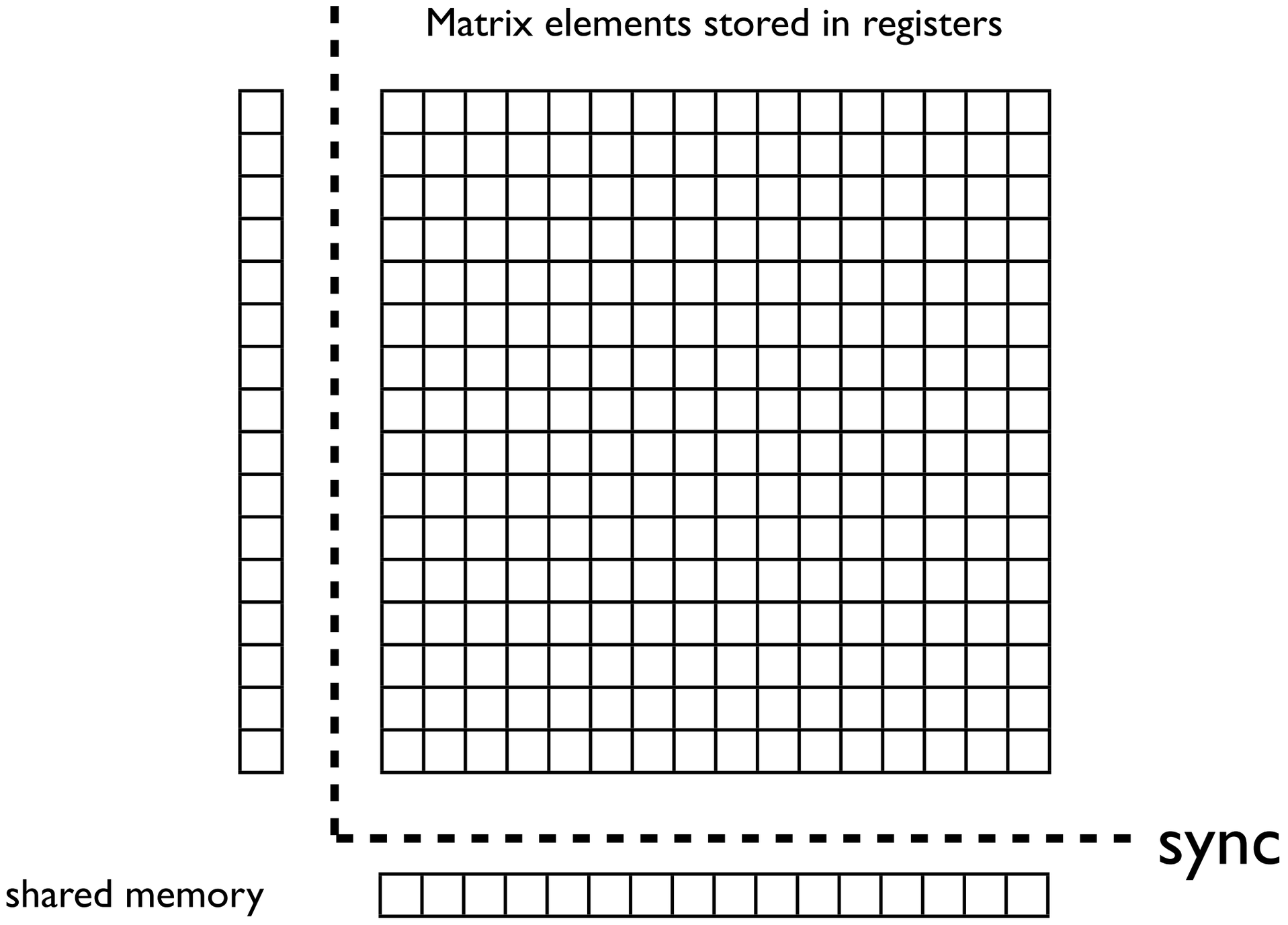


Matrix elements stored in registers

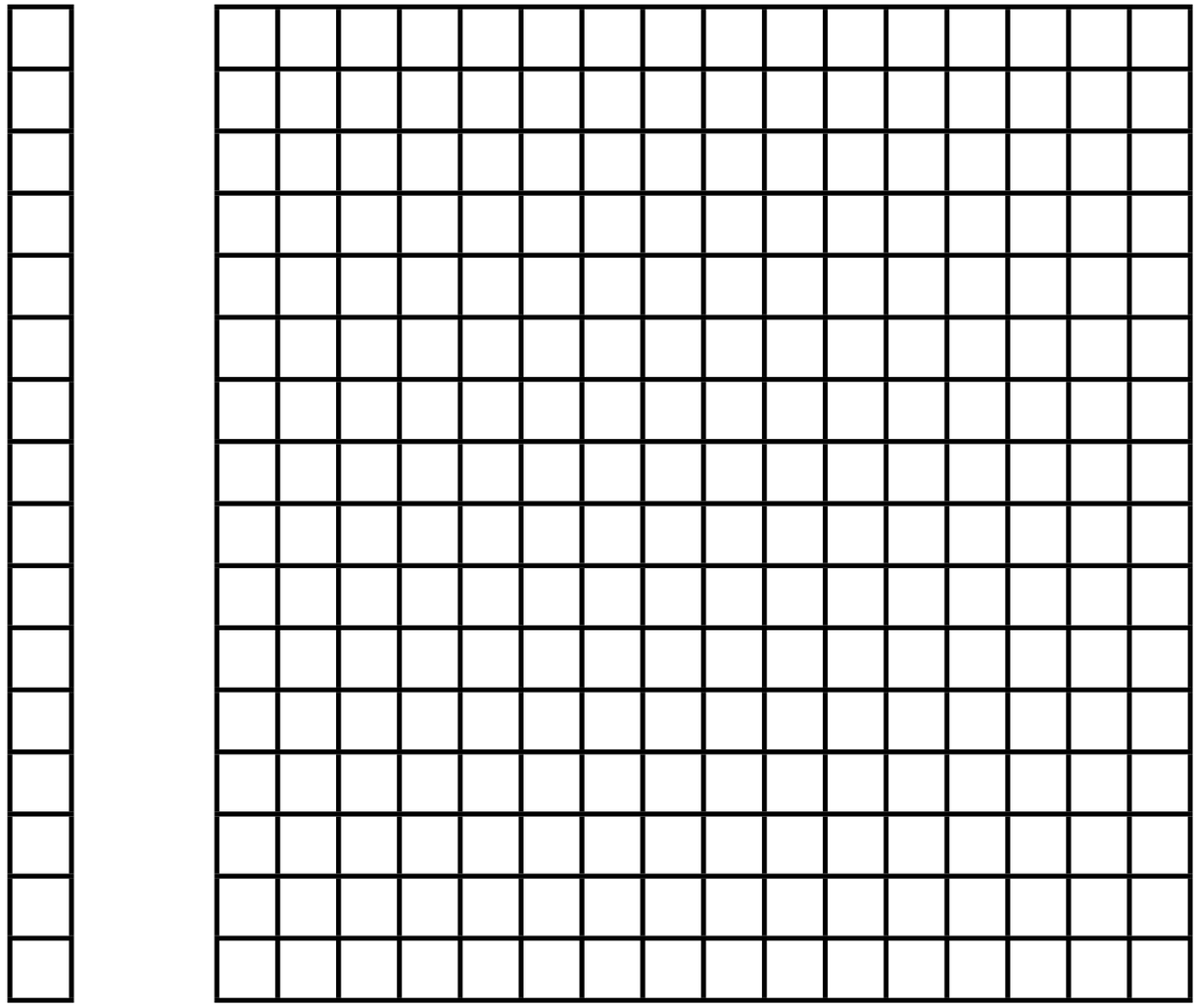


shared memory

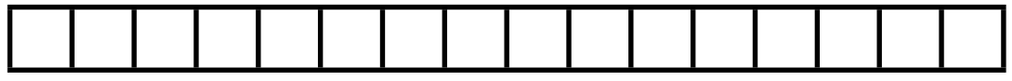



Matrix elements stored in registers
sync
shared memory

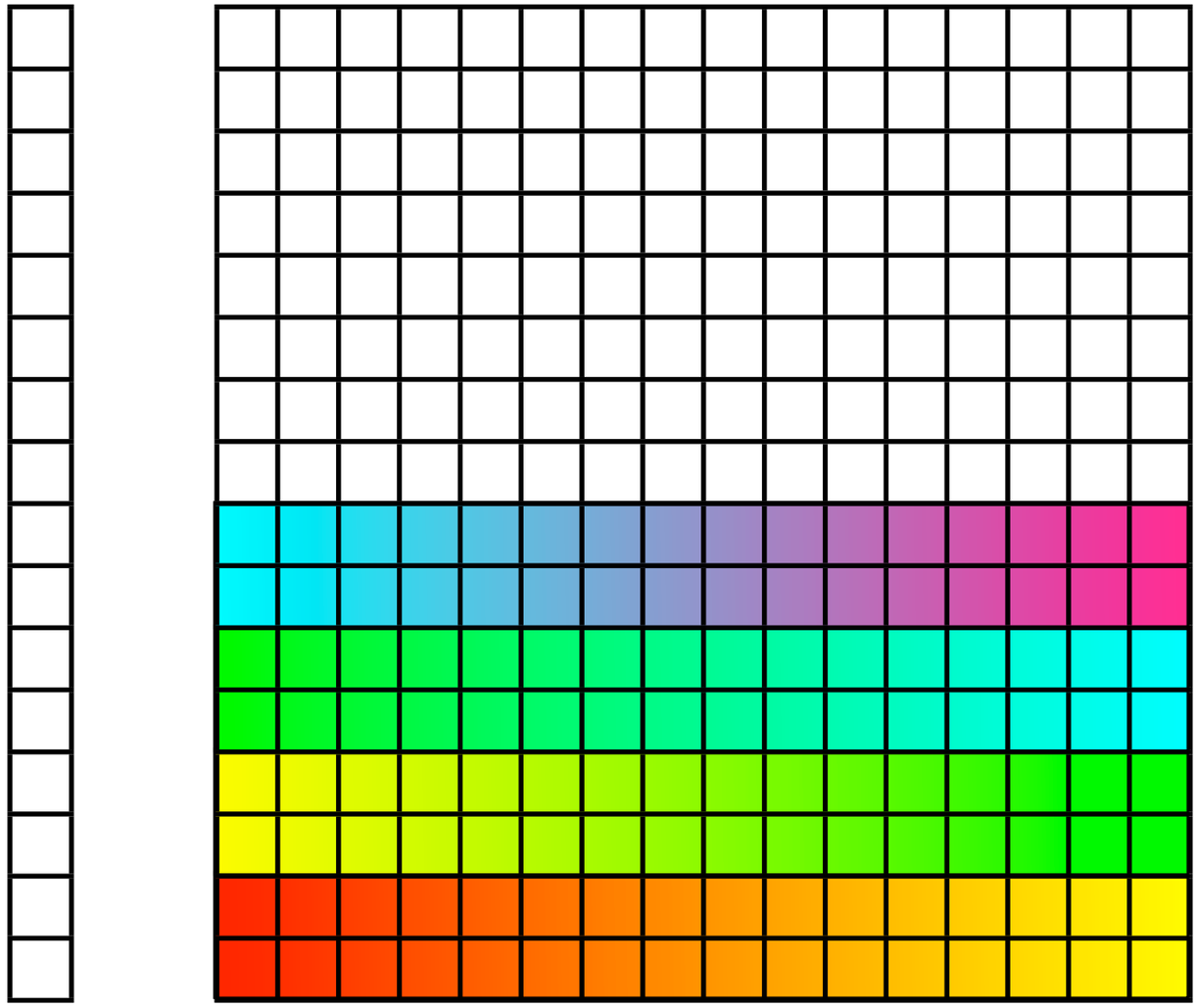
Matrix elements stored in registers



shared memory

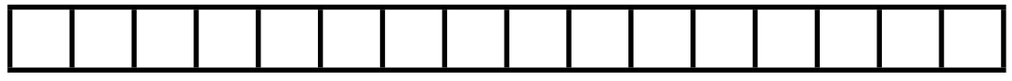


Matrix elements stored in registers

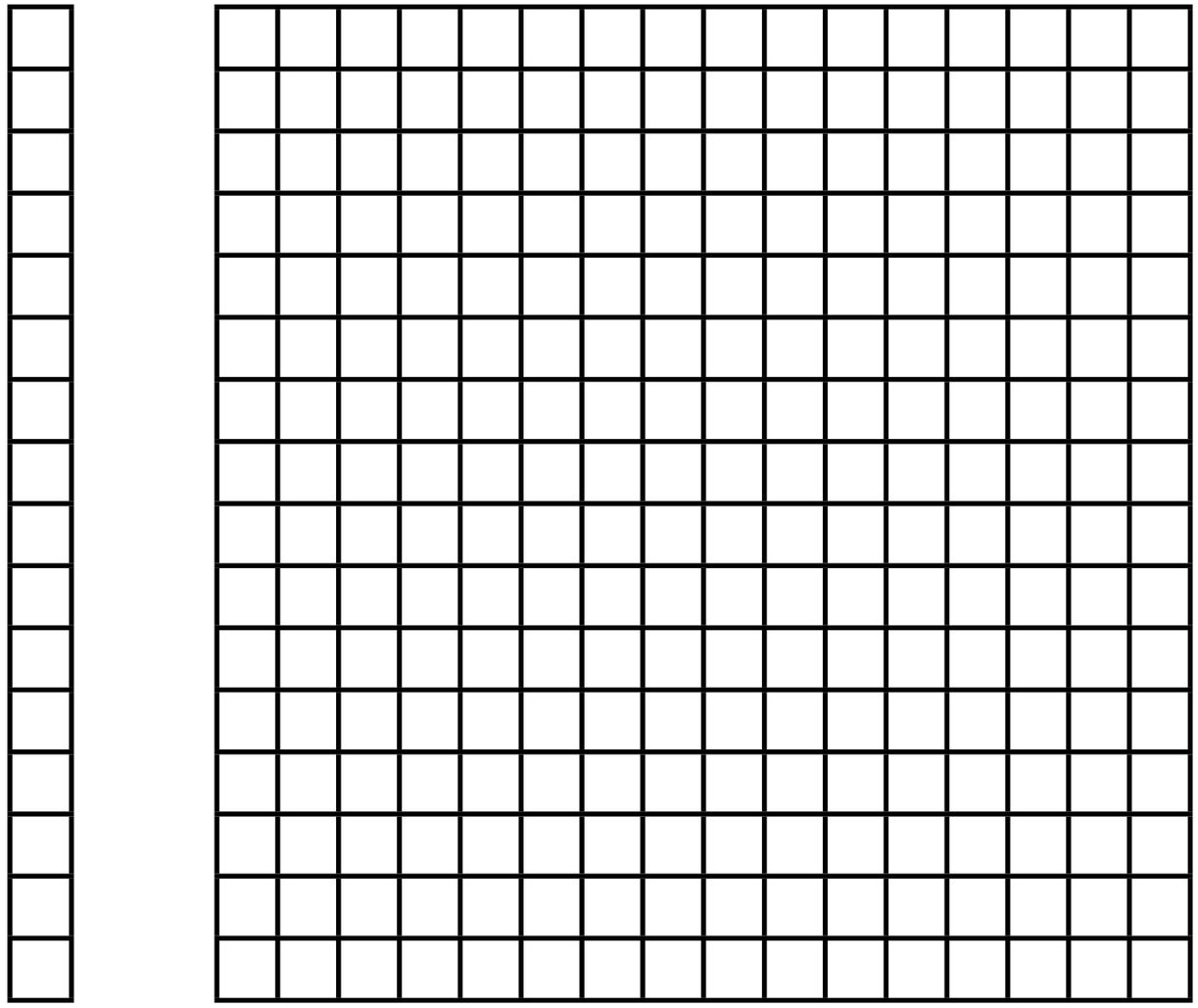


1st warp compute

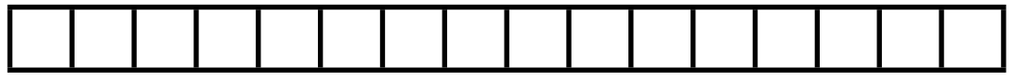
shared memory



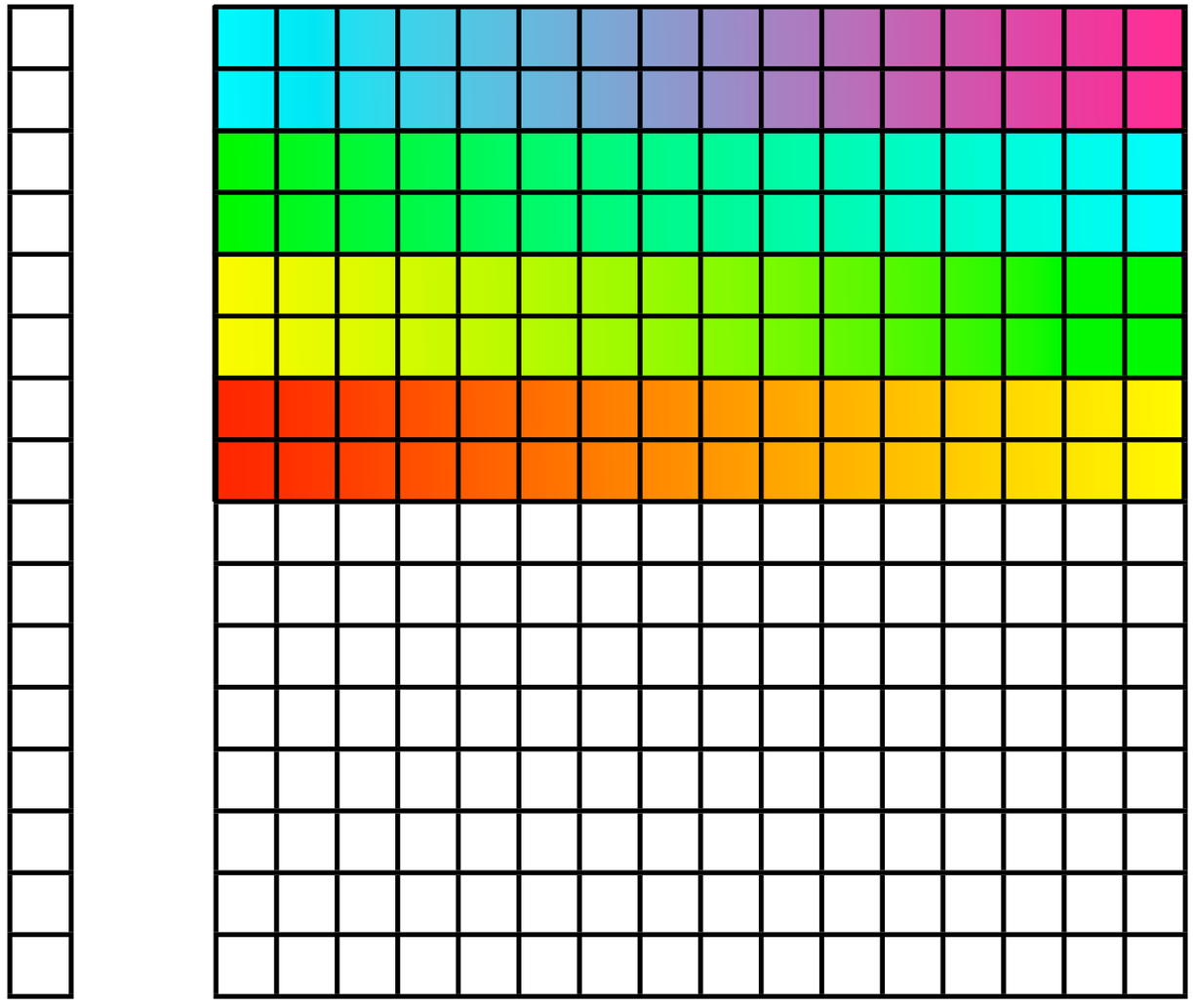
Matrix elements stored in registers



shared memory

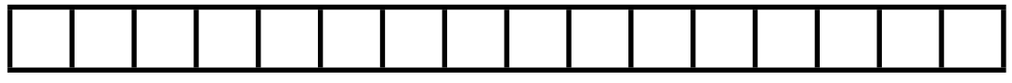


Matrix elements stored in registers

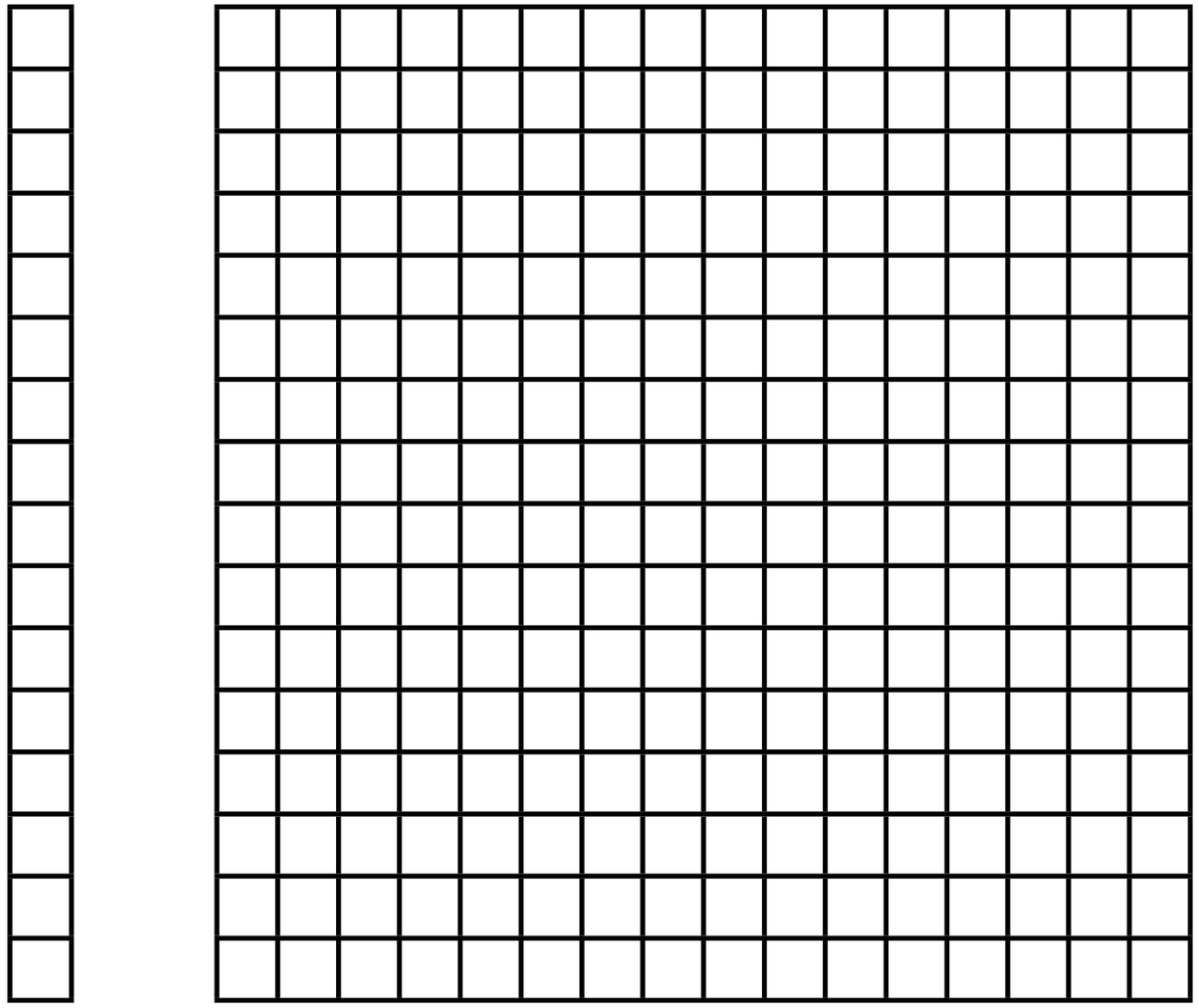


2nd warp compute

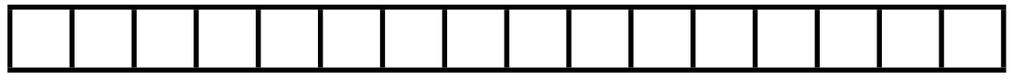
shared memory

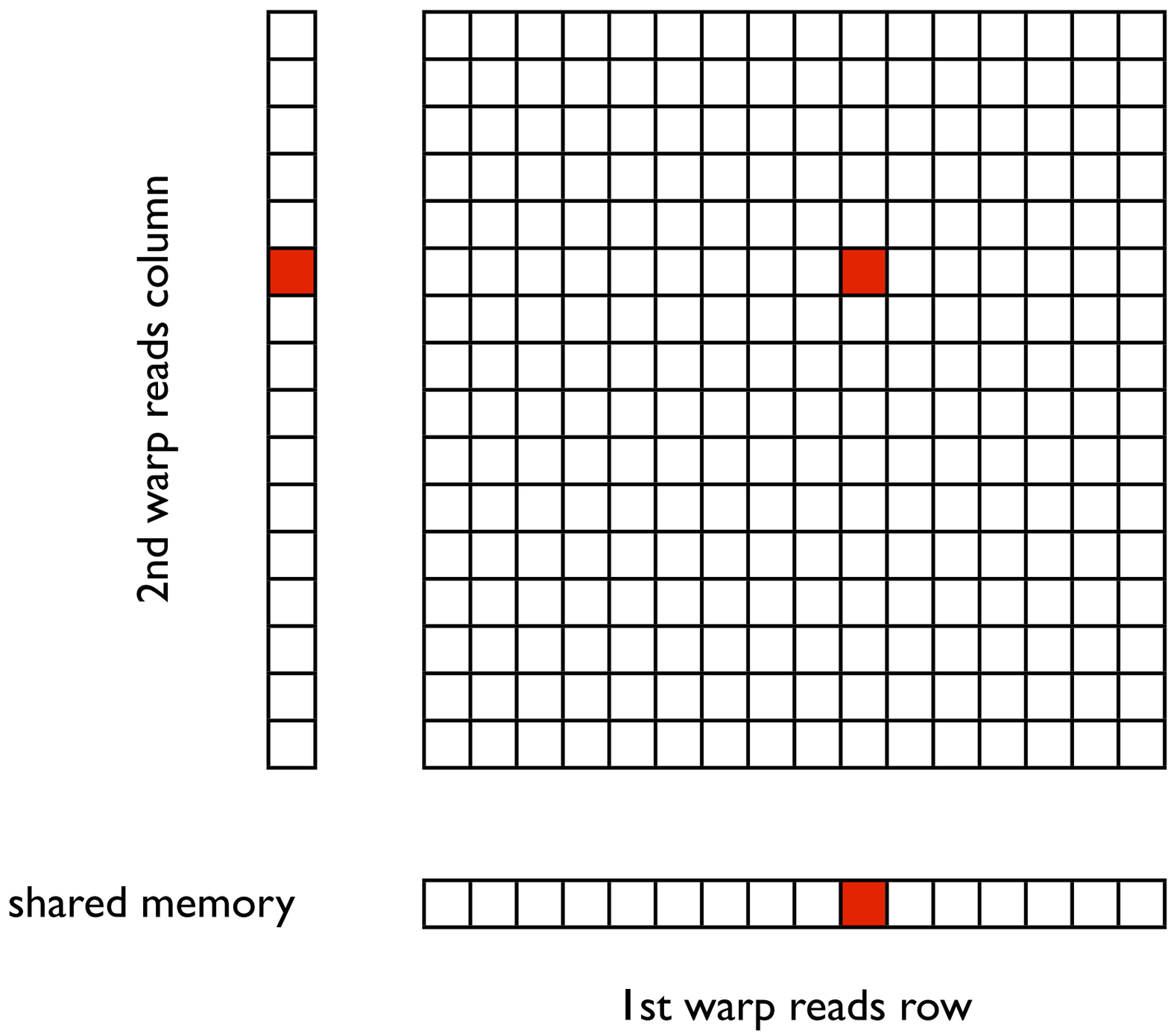


Matrix elements stored in registers



shared memory





Registers

Each thread
computes
a 1x1 tile

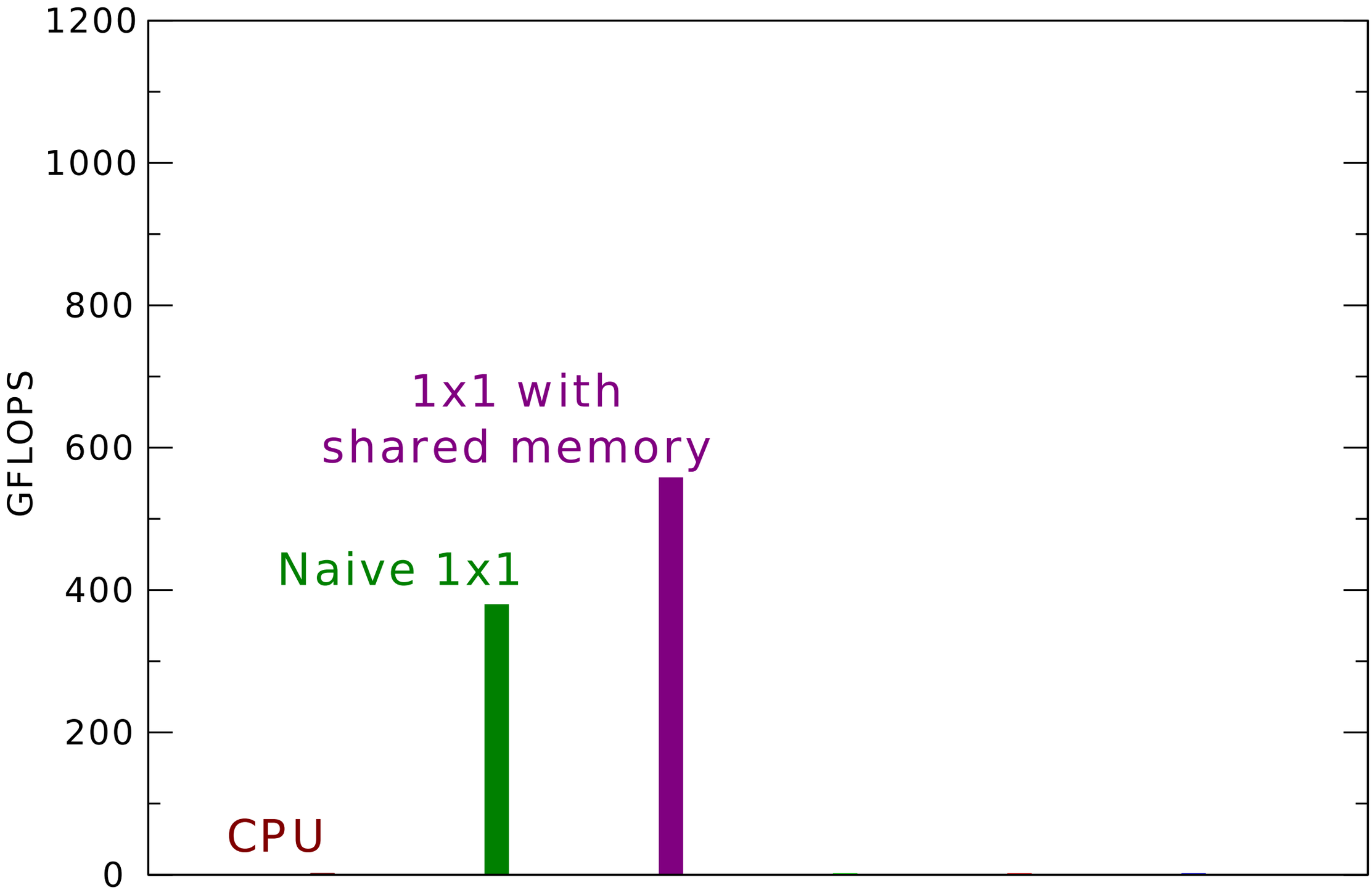
flop/byte
Algorithm: 1
Hardware: 1.5

Shared Memory

Each thread
block
loads a 16x16 tile

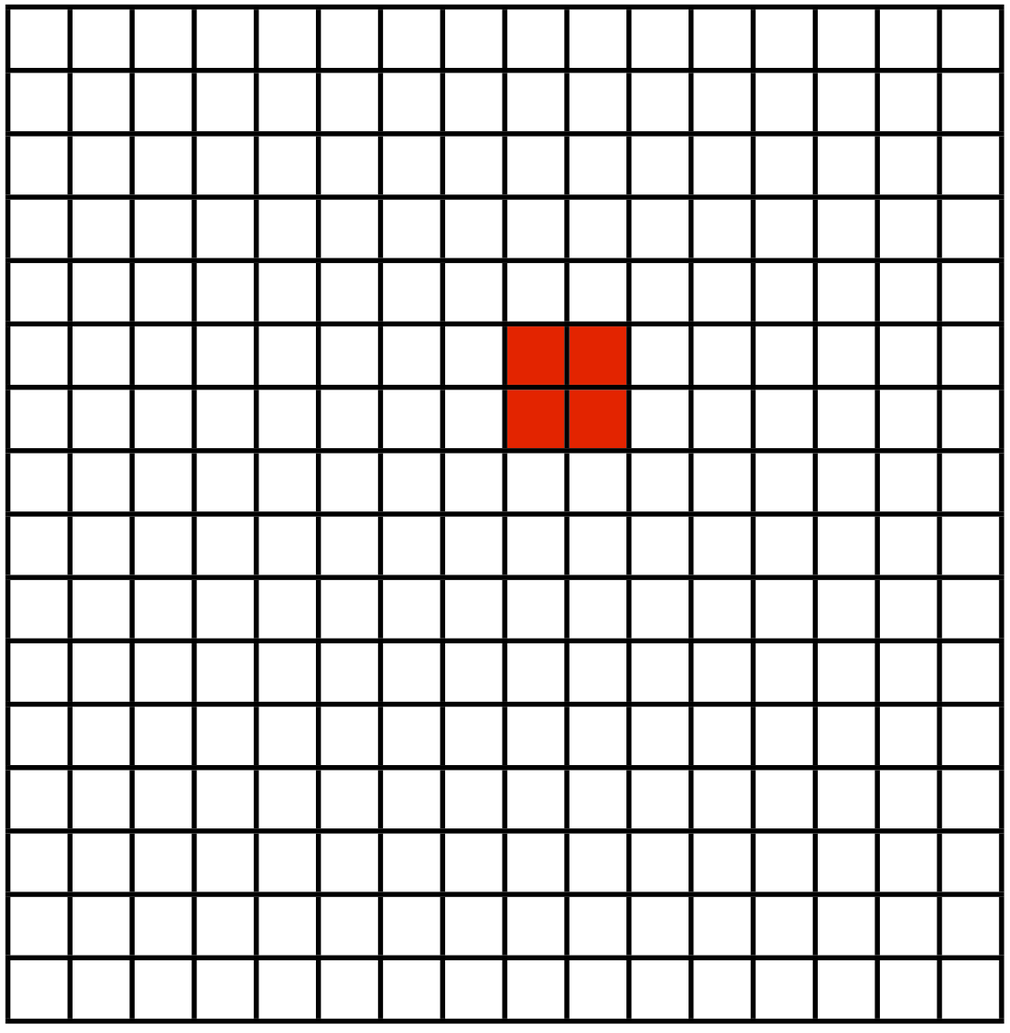
flop/byte
Algorithm: 16
Hardware: 7.6

Performance for N_s=512, N_c=12, N_t=1000



8x8 threads = 16x16 stations

Matrix elements stored in registers



Registers

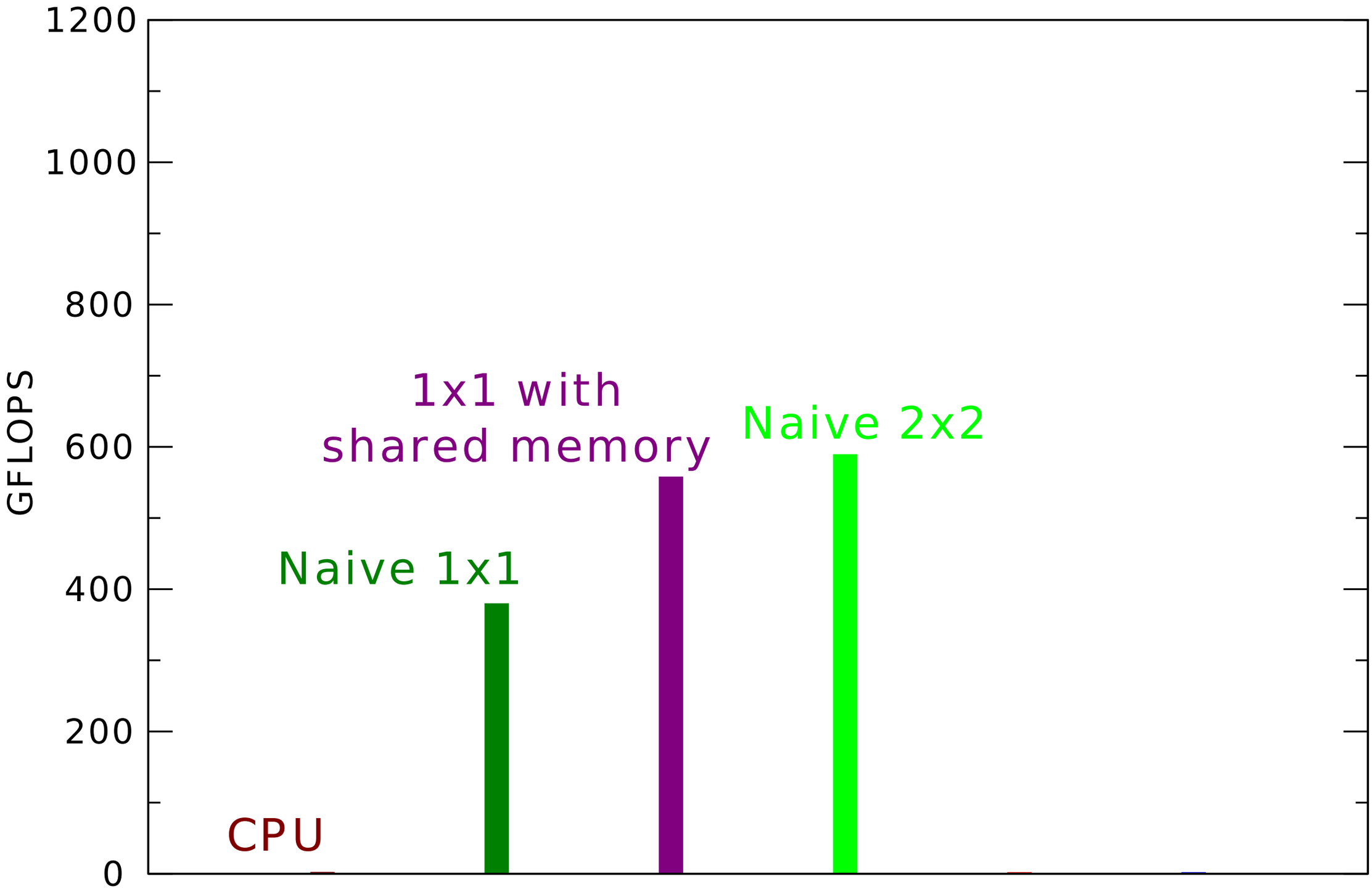
Each thread computes a 2x2 tile

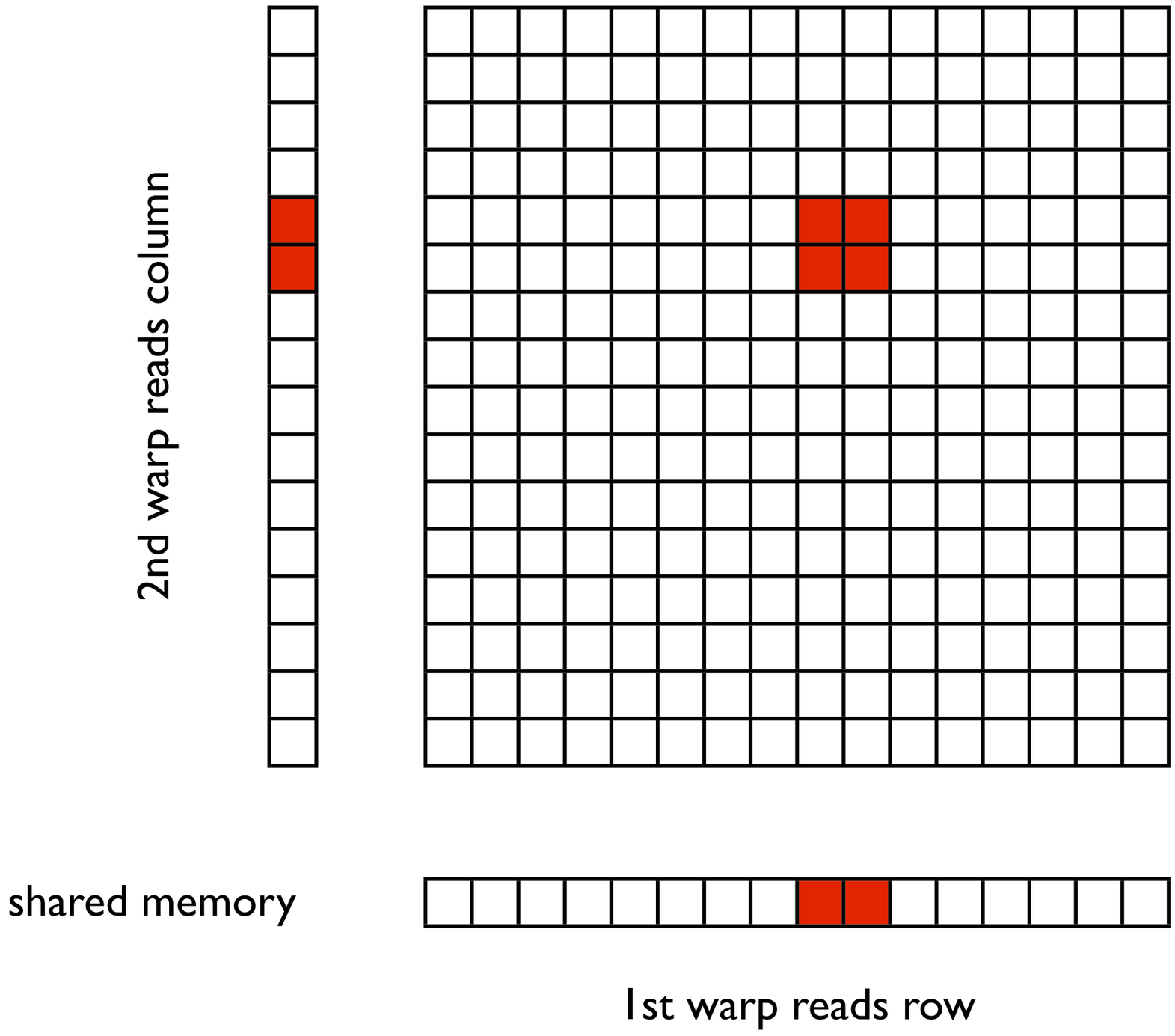
flop/byte
Algorithm: 2
Hardware: 7.6

Cache

Data shared between threads using L1 cache

Performance for N_s=512, N_c=12, N_t=1000





Registers

Each thread
computes
a 1x1 tile

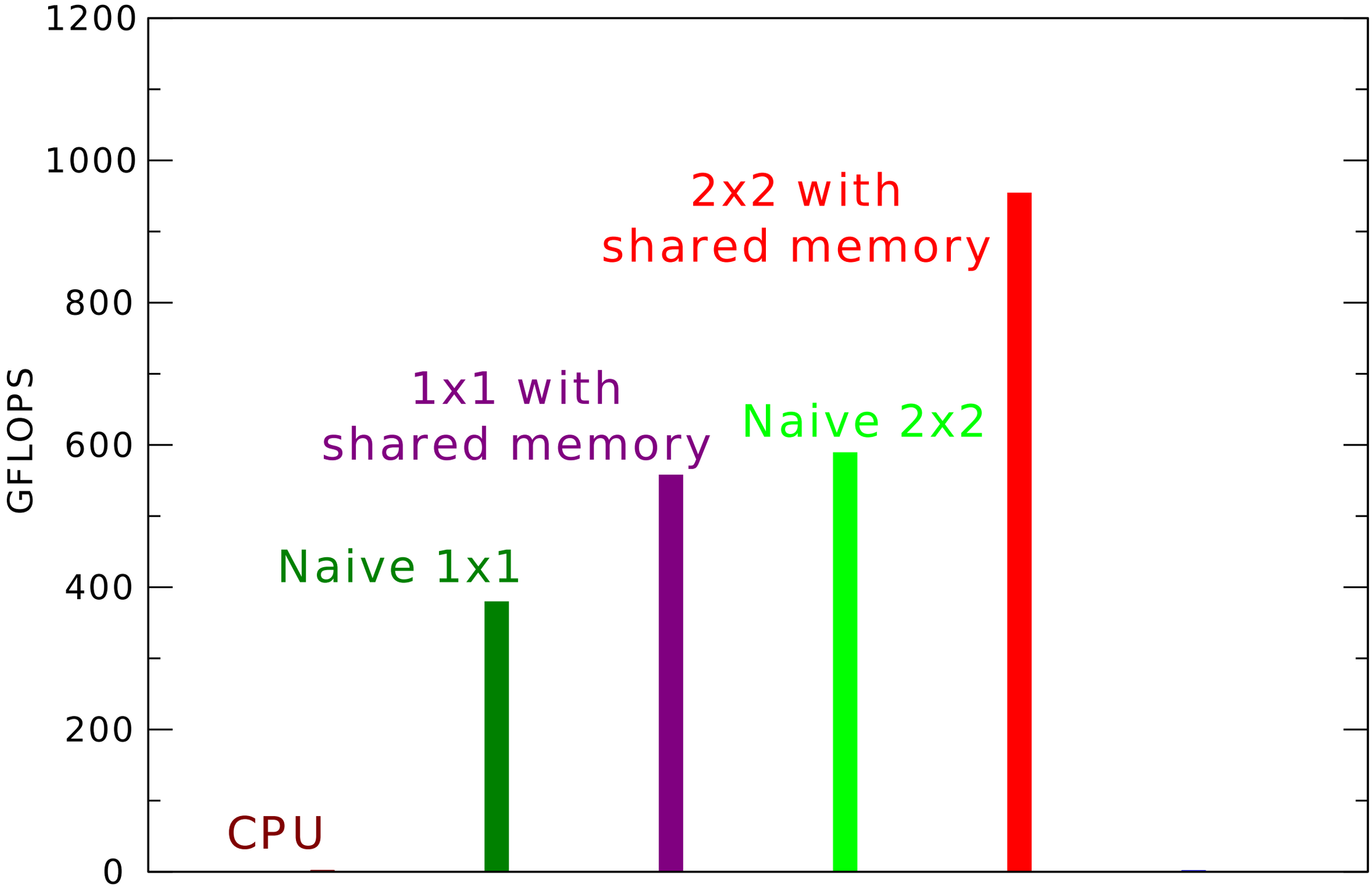
flop/byte
Algorithm: 2
Hardware: 1.5

Shared Memory

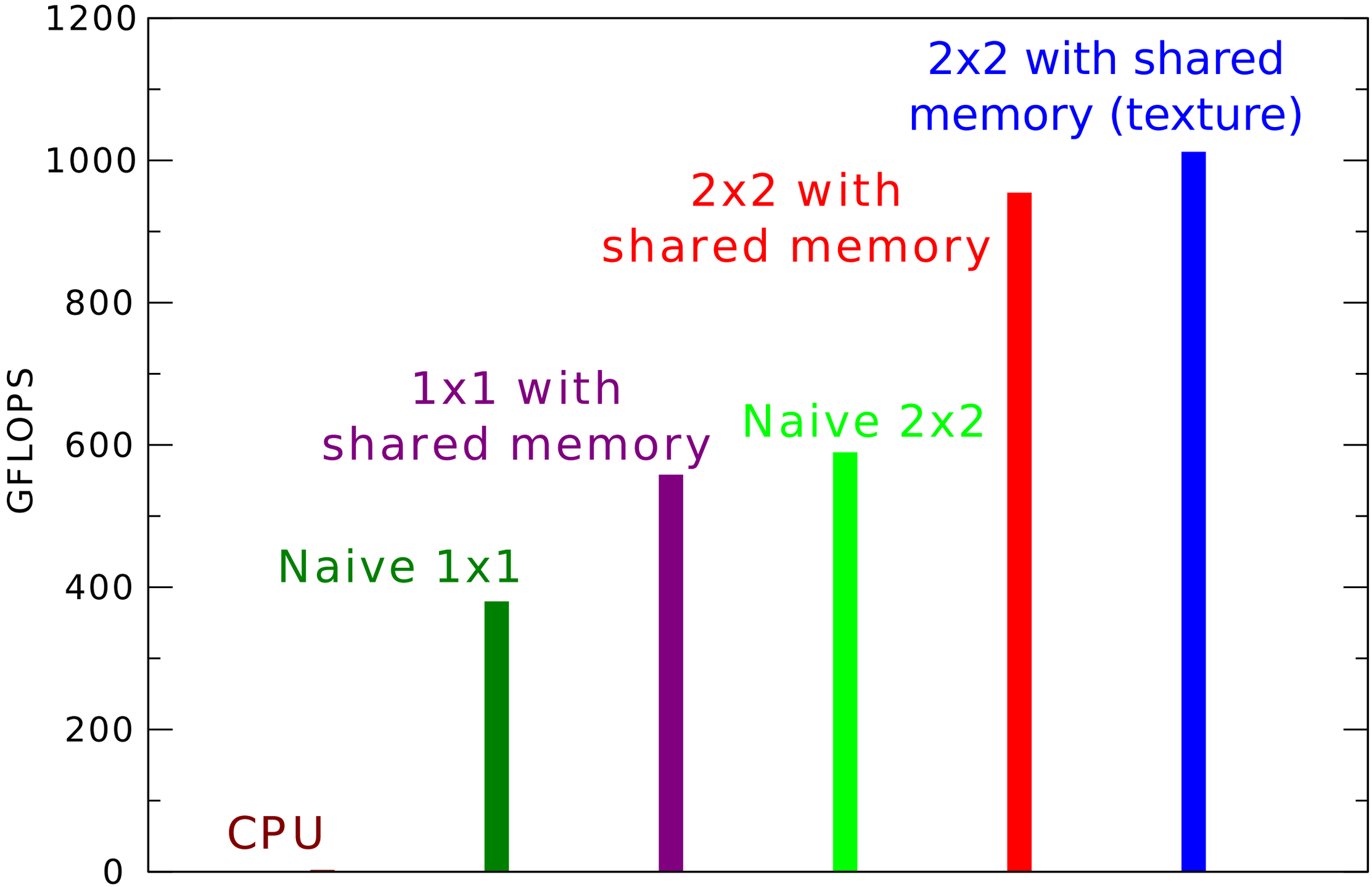
Each thread
block
loads a 16x16 tile

flop/byte
Algorithm: 16
Hardware: 7.6

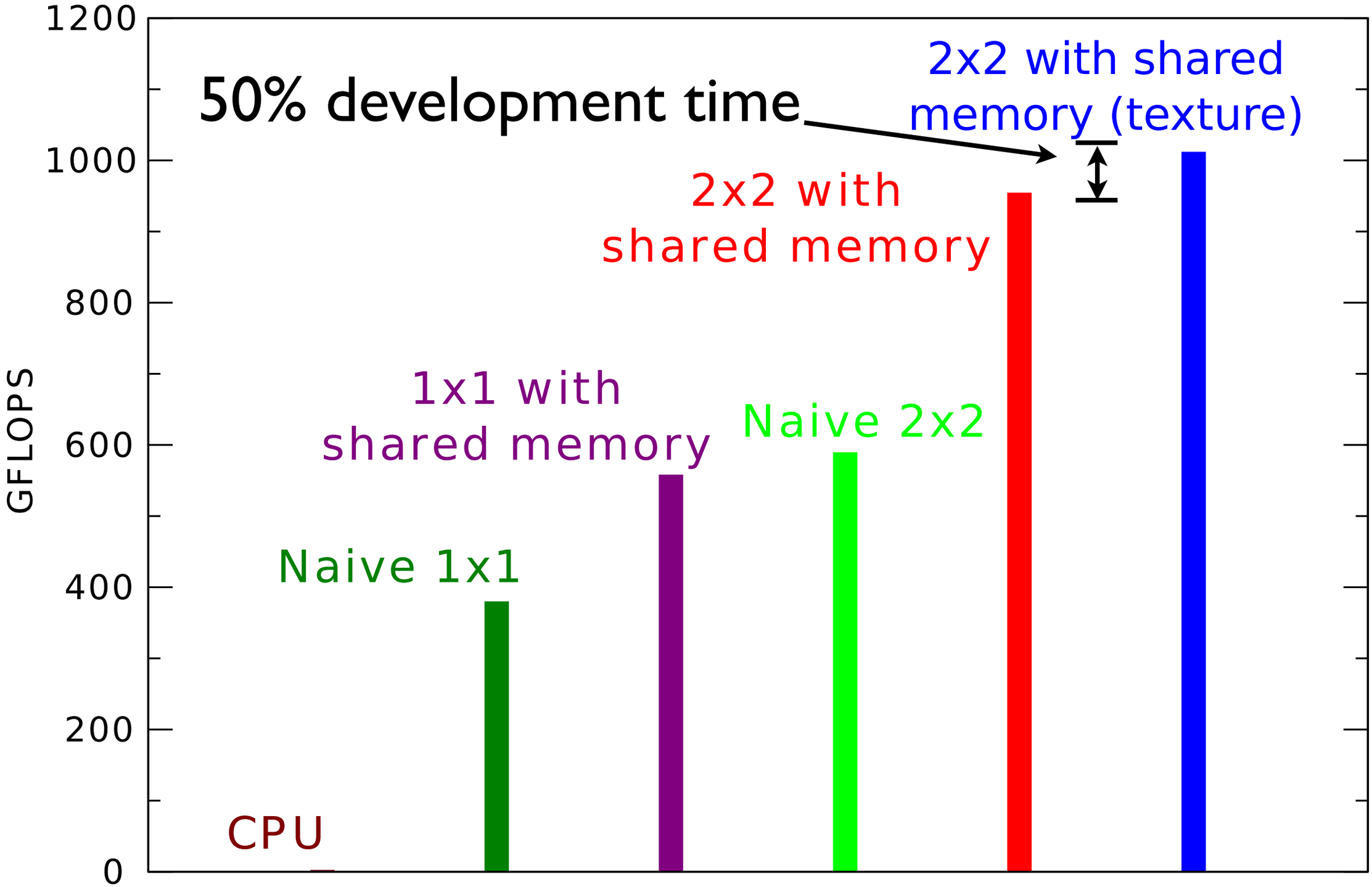
Performance for N_s=512, N_c=12, N_t=1000

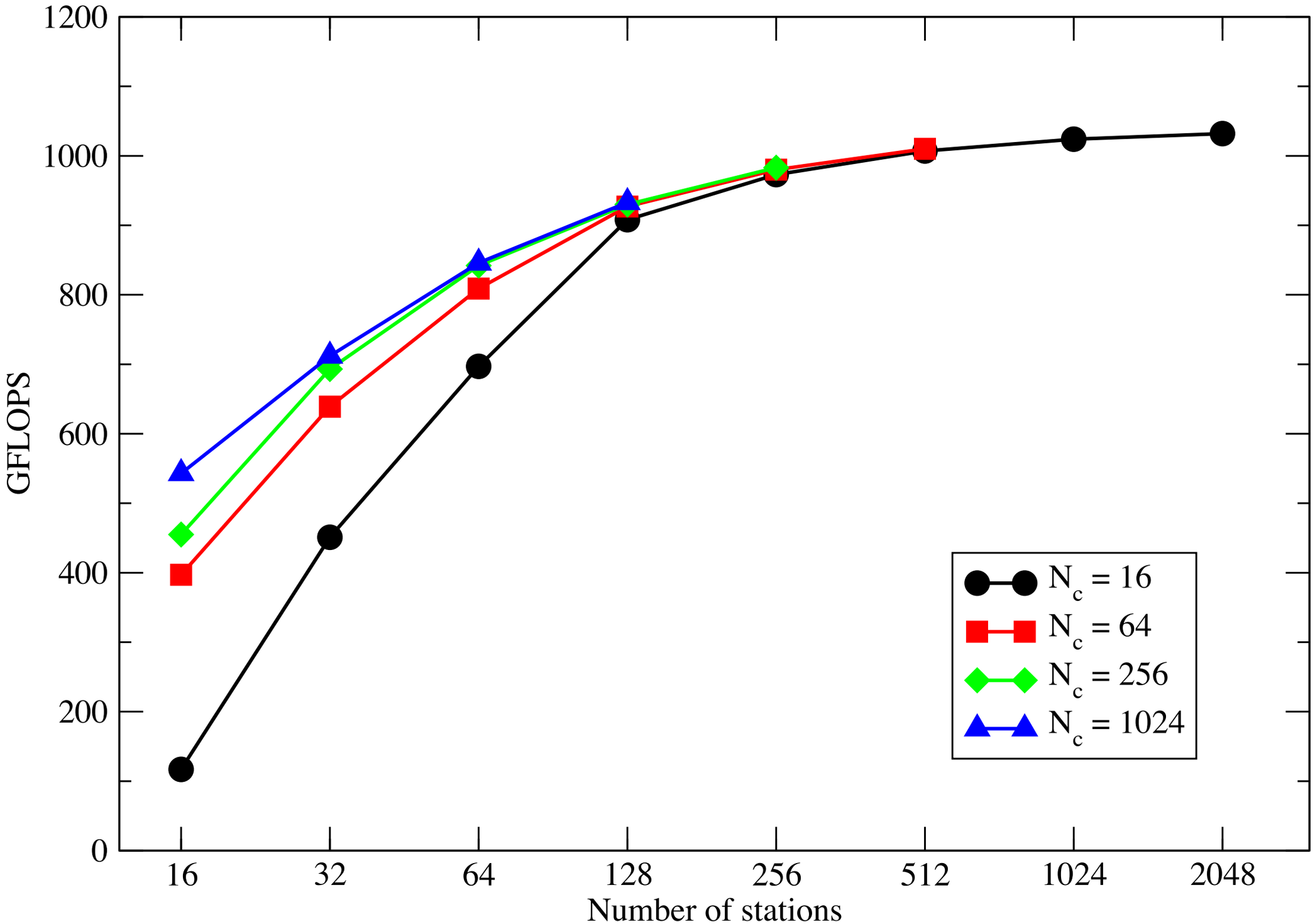


Performance for N_s=512, N_c=12, N_t=1000



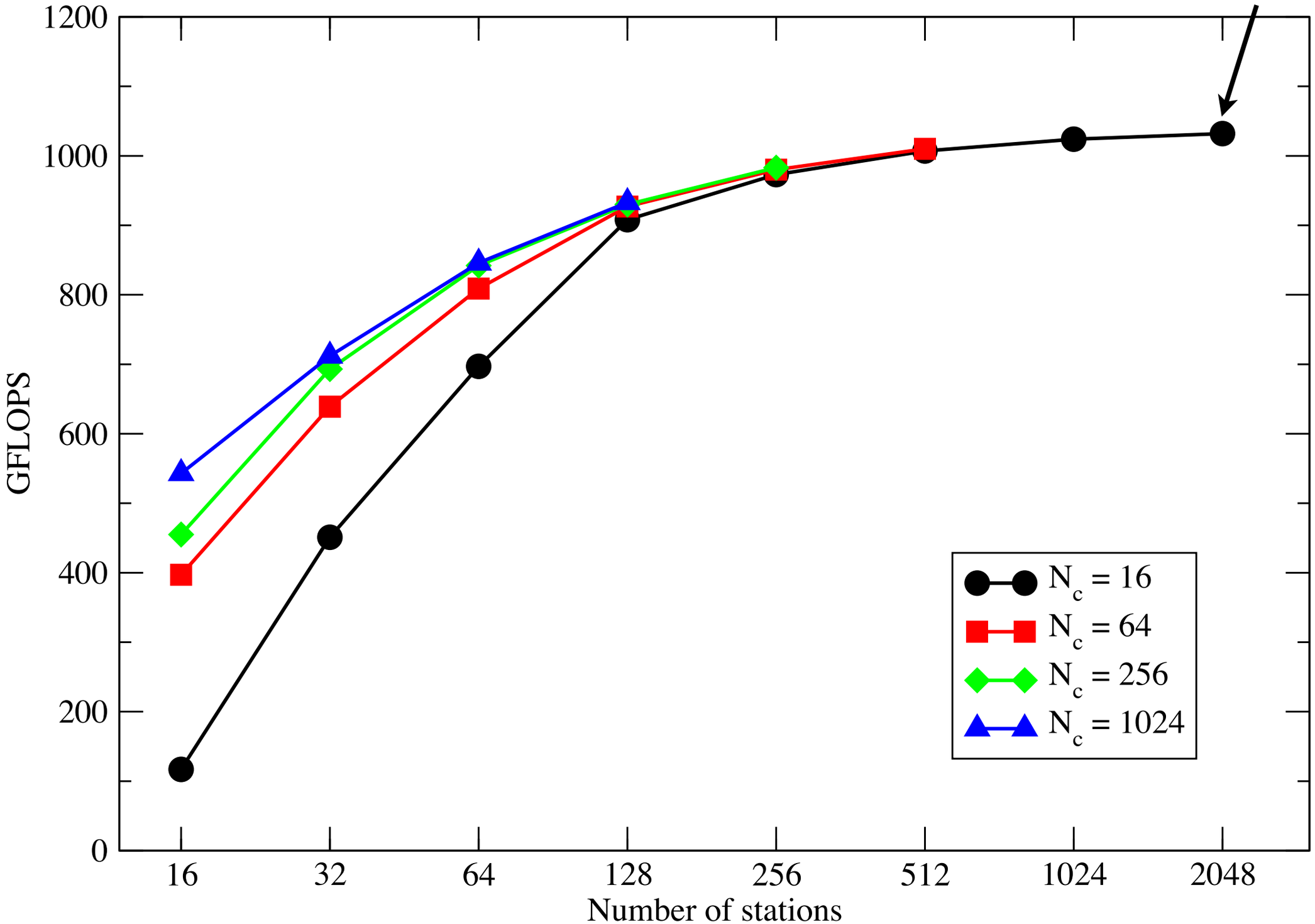
Performance for N_s=512, N_c=12, N_t=1000




1200
1000
800
600
400
200
0
GFLOPS
16
32
64
128
256
512
1024
2048
Number of stations
N_c = 16
N_c = 64
N_c = 256
N_c = 1024

1042 GFLOPS = 78% (91%) peak

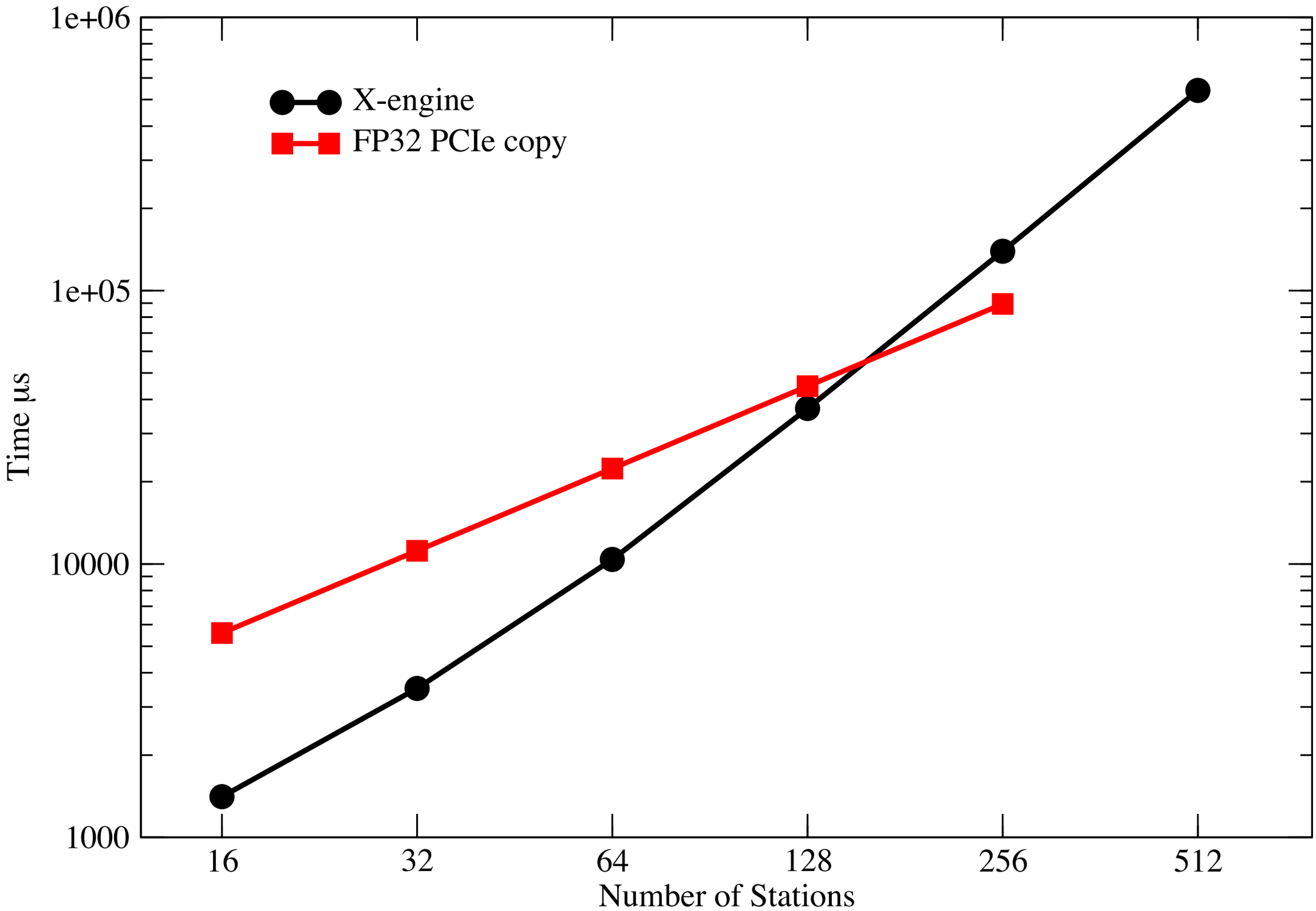
67 GB/s = 38% peak



Feeding the Beast

- 1 TFLOP sustained is all very well, but what if the bus can't keep up?
- Algorithm profile
  - Host → Device $O(N_c N_s B_c)$
  - Kernel $O(N_c N_s^2 B_c)$
  - Device → Host $O(N_c N_s^2)$
- Kernel will dominate at large N_s but what about small N_s?
- Previous work showed X-engine limited by bus
  - ~250 GFLOPS at $N_s = 64$ (van Nieuwpoort *et al*)

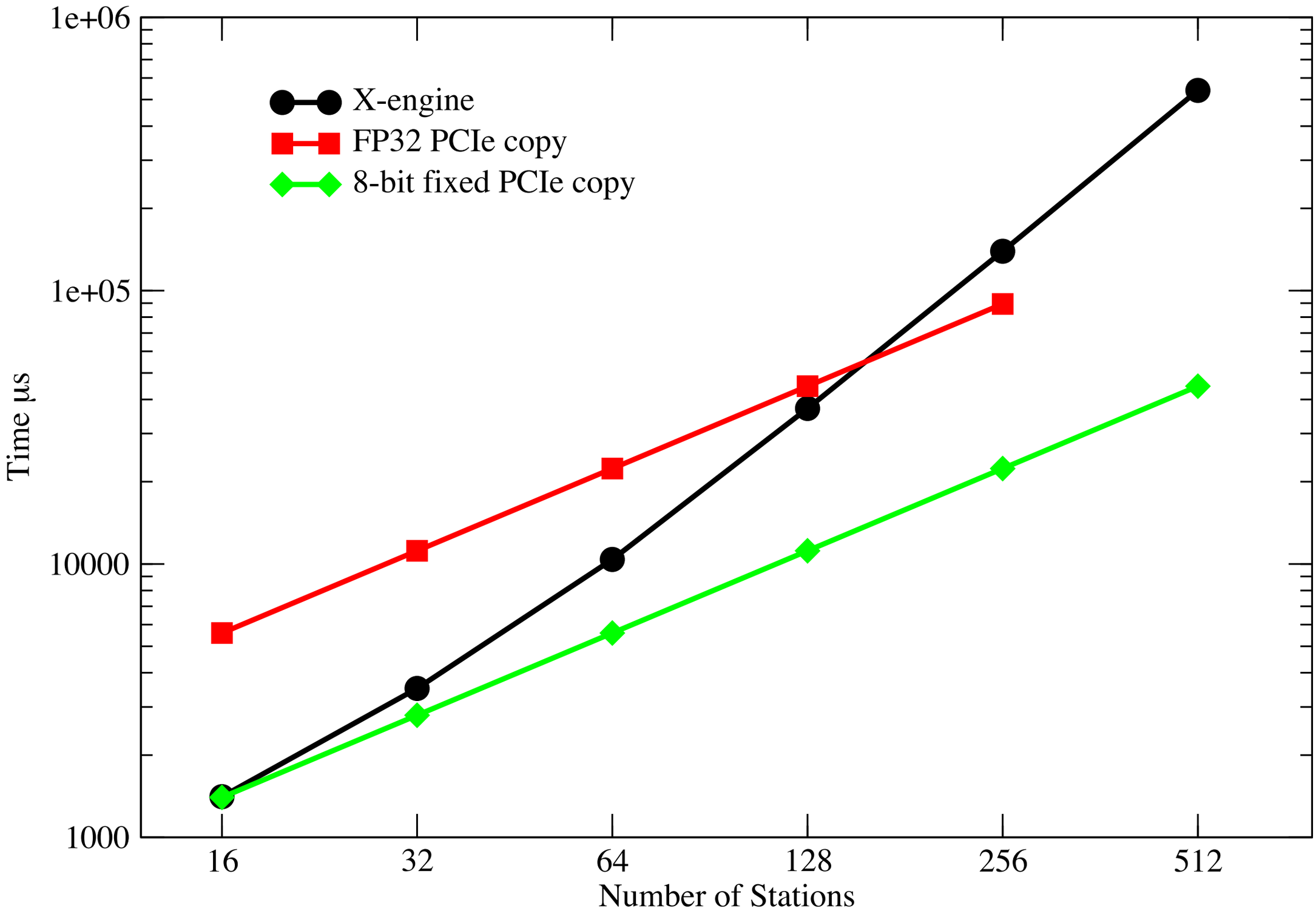
Download time versus Kernel execution time



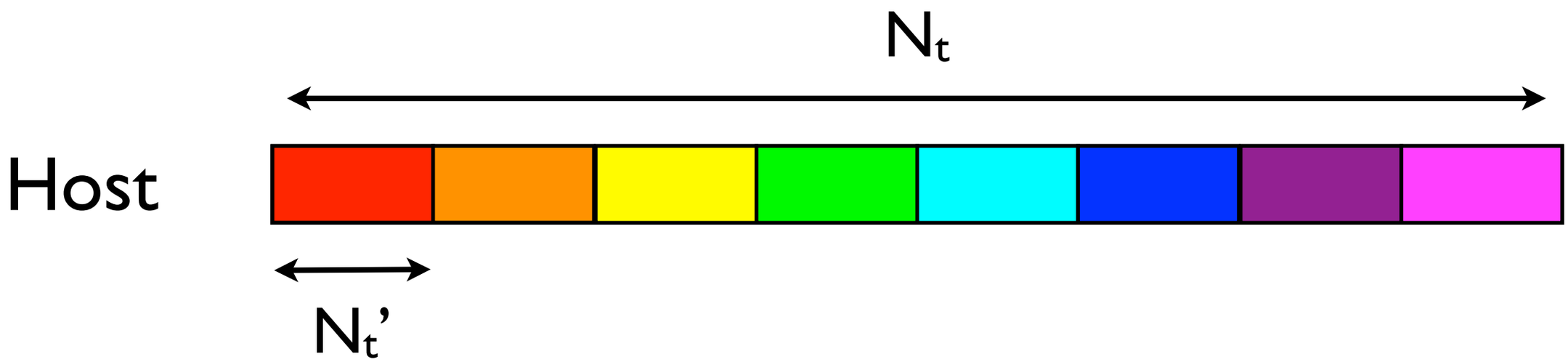
The beast eats a lot, but each mouthful is small...

- PCIe bus is a severe constraint to performance
  - $N_s \geq 256$ kernel dominates
- Input signal precision typically 4-5 bits
- FP32 is a complete waste of bandwidth
- Accumulated correlation matrix must be high precision (FPGAs typically use > 12-bit precision)
- Use 8-bit precision for input, keep matrix FP32
  - 8-bit natively supported through textures

Download time versus Kernel execution time

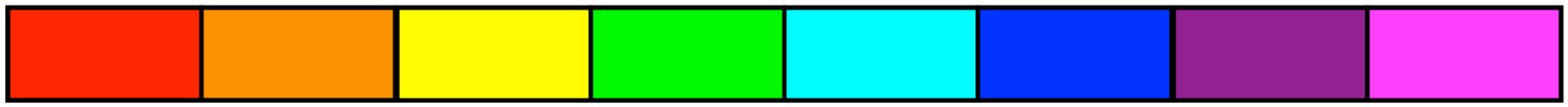


Pipelined Algorithm



Device Buffer Compute

Pipelined Algorithm

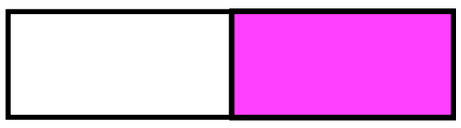
Host 

Device Buffer Compute

Pipelined Algorithm

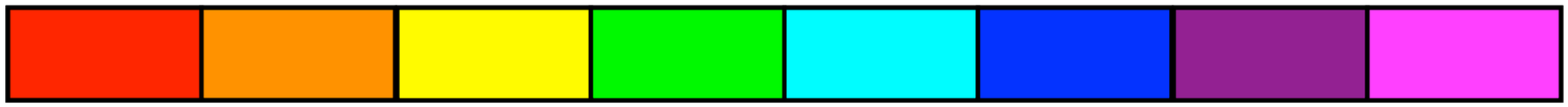
Host

Device

Buffer 

Compute

Pipelined Algorithm

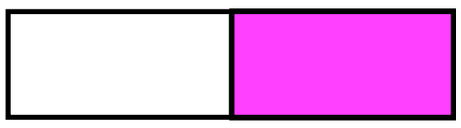
Host 

Device Buffer Compute

Pipelined Algorithm

Host

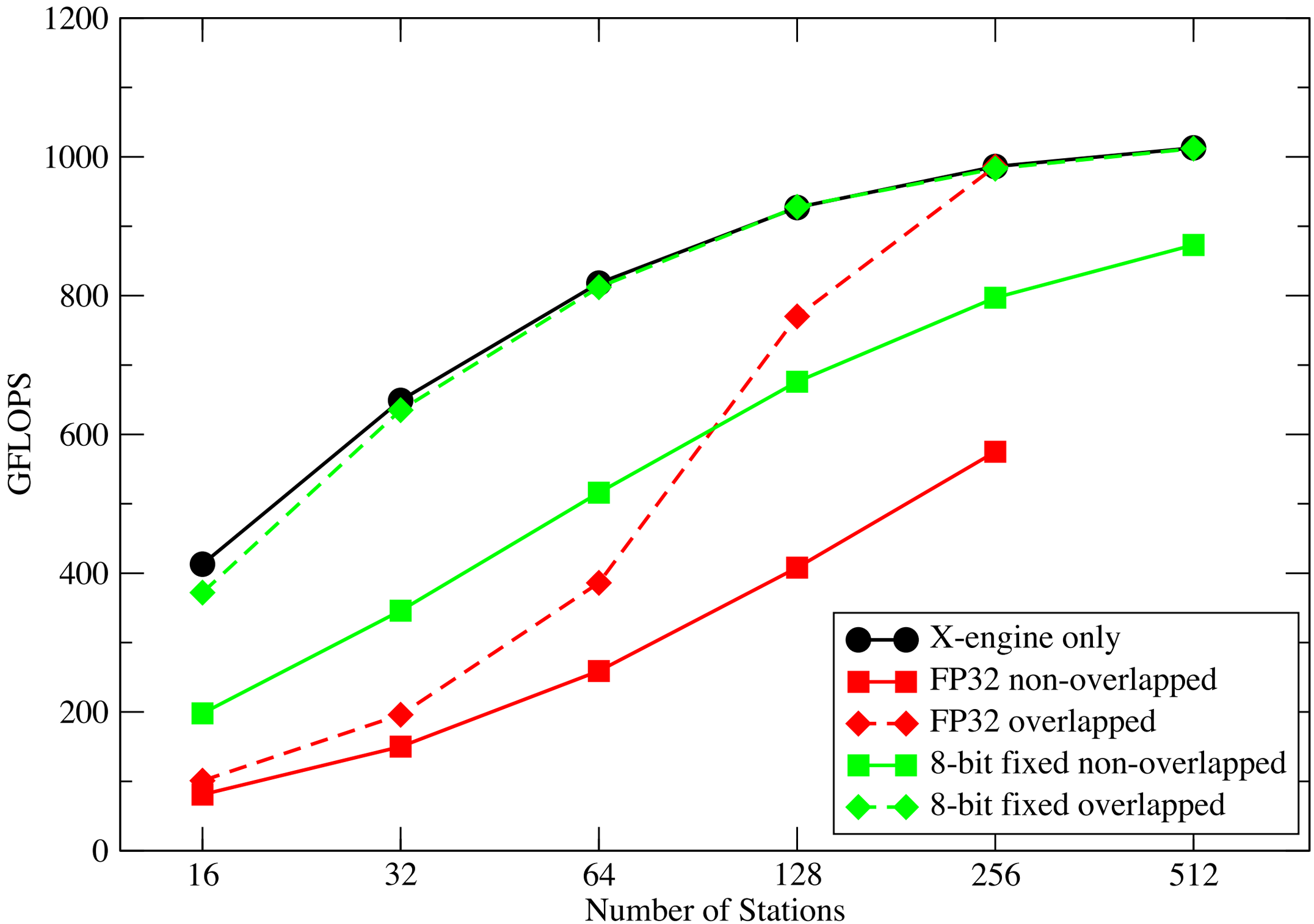
Device

Buffer 

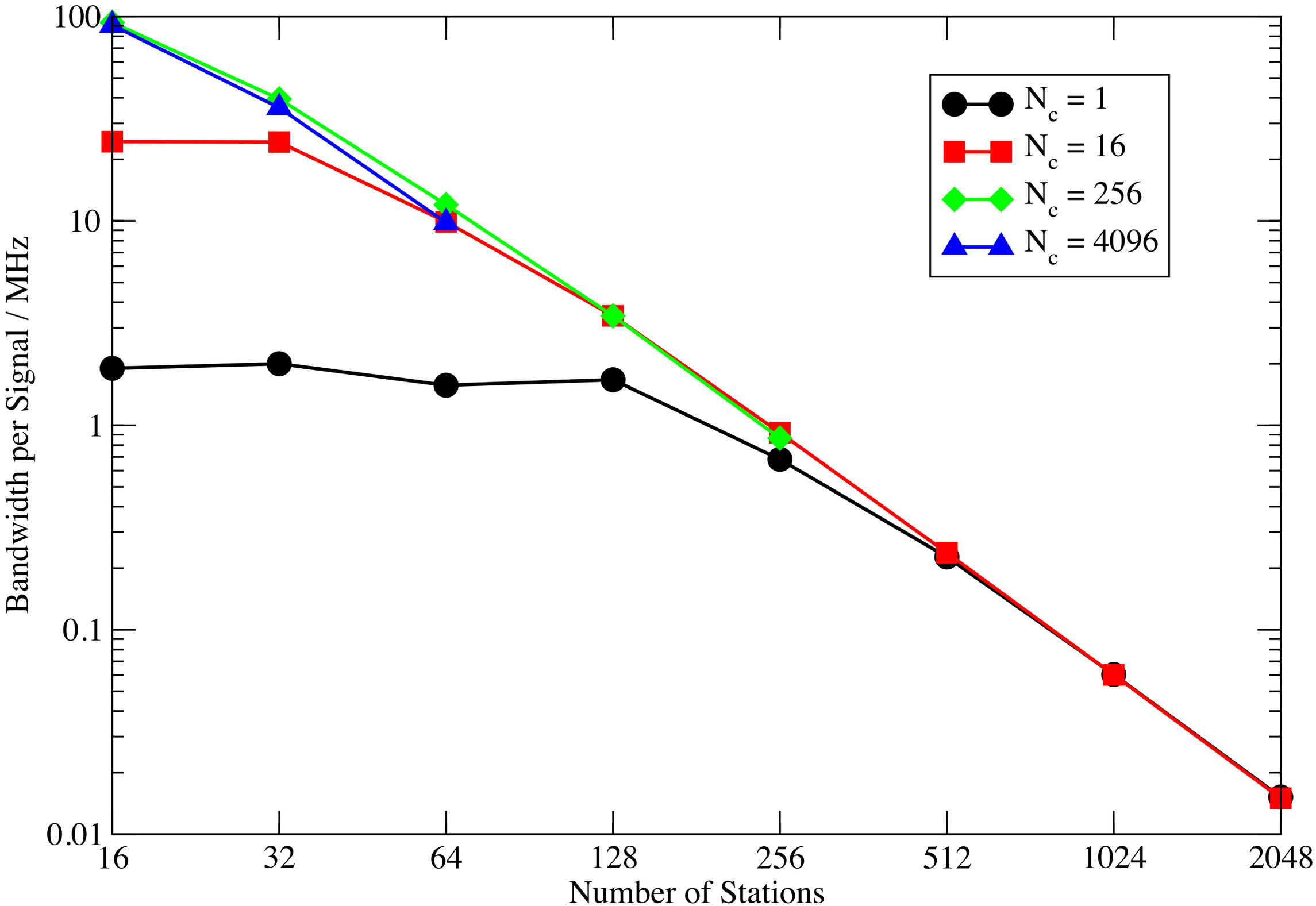
Compute

Sustained X-engine performance

128 frequency channels, 1024 time samples



Bandwidth achievable per signal



Multi-GPU

- Trivial to parallelize across frequency
  - PCIe bus contention manageable
- 4 TFLOPs sustained with 4 GPUs
  - 1300 Watts total
    - 3 GFLOPS/Watt (cf 1684 GFLOPS/Watt #1Green 500)
  - $6K for workstation => $1.50 per GFLOP

X-engine Performance Across Platforms

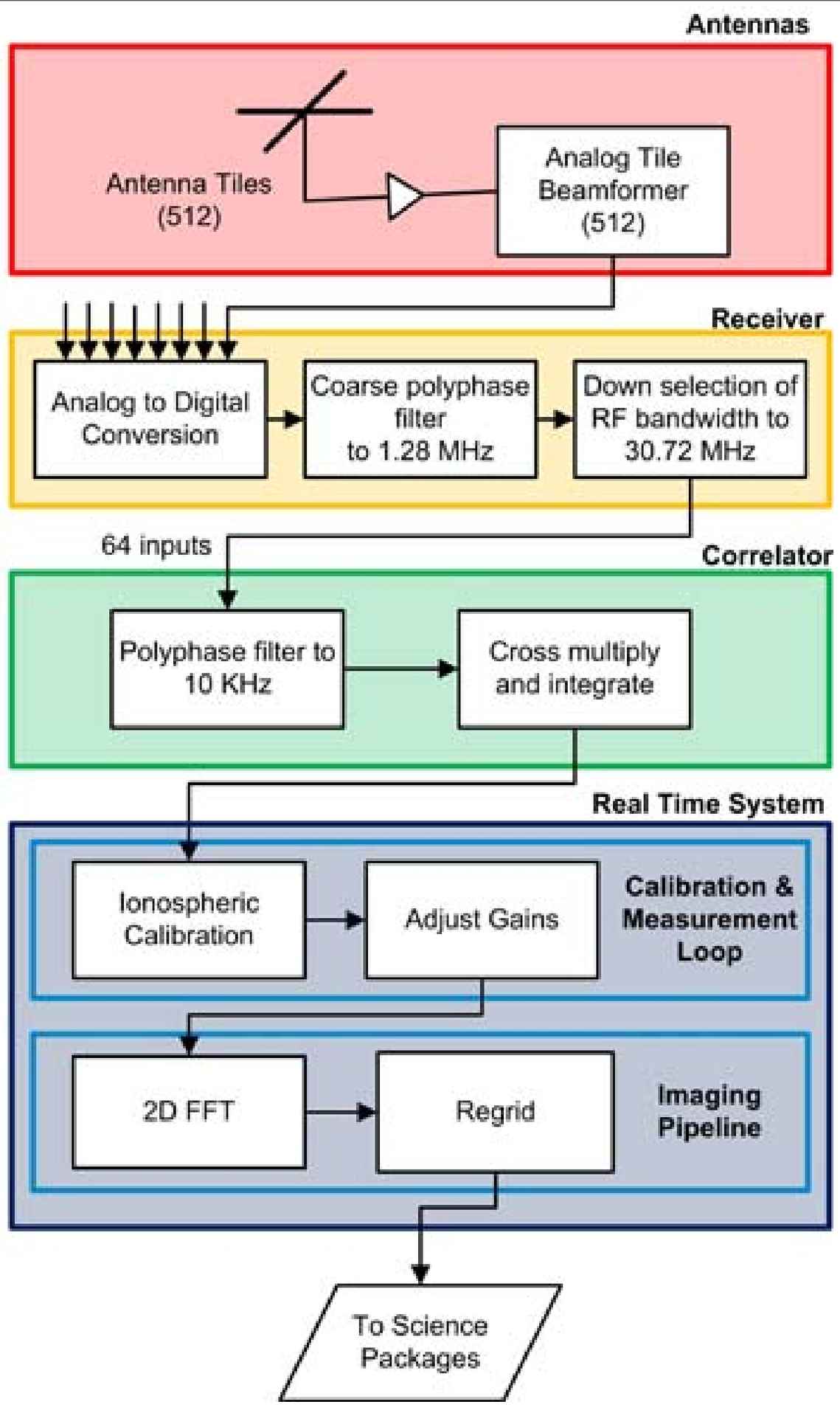
Architecture	GFLOPS	GOPS/Watt per chip (total)
Intel Core i7 (quad)*	48.0	0.4
IBM BG/P*	13.1	0.5
IBM Cell*	187	2.7
Nvidia C1060*	243	1
Nvidia GTX 480†	1042	~4 (3)

* van Nieuwpoort and Romein, †this work, #de Souza *et al* and Lonsdale *et al*, +Manley

X-engine Performance Across Platforms

Architecture	GFLOPS	GOPS/Watt per chip (total)
Intel Core i7 (quad)*	48.0	0.4
IBM BG/P*	13.1	0.5
IBM Cell*	187	2.7
Nvidia C1060*	243	1
Nvidia GTX 480†	1042	~4 (3)
MWA Correlator # (Virtex 4 SX35)		(~10)
Roach II+ (Virtex 6)		(~58)

* van Nieuwpoort and Romein, †this work, #de Souza *et al* and Lonsdale *et al*, +Manley



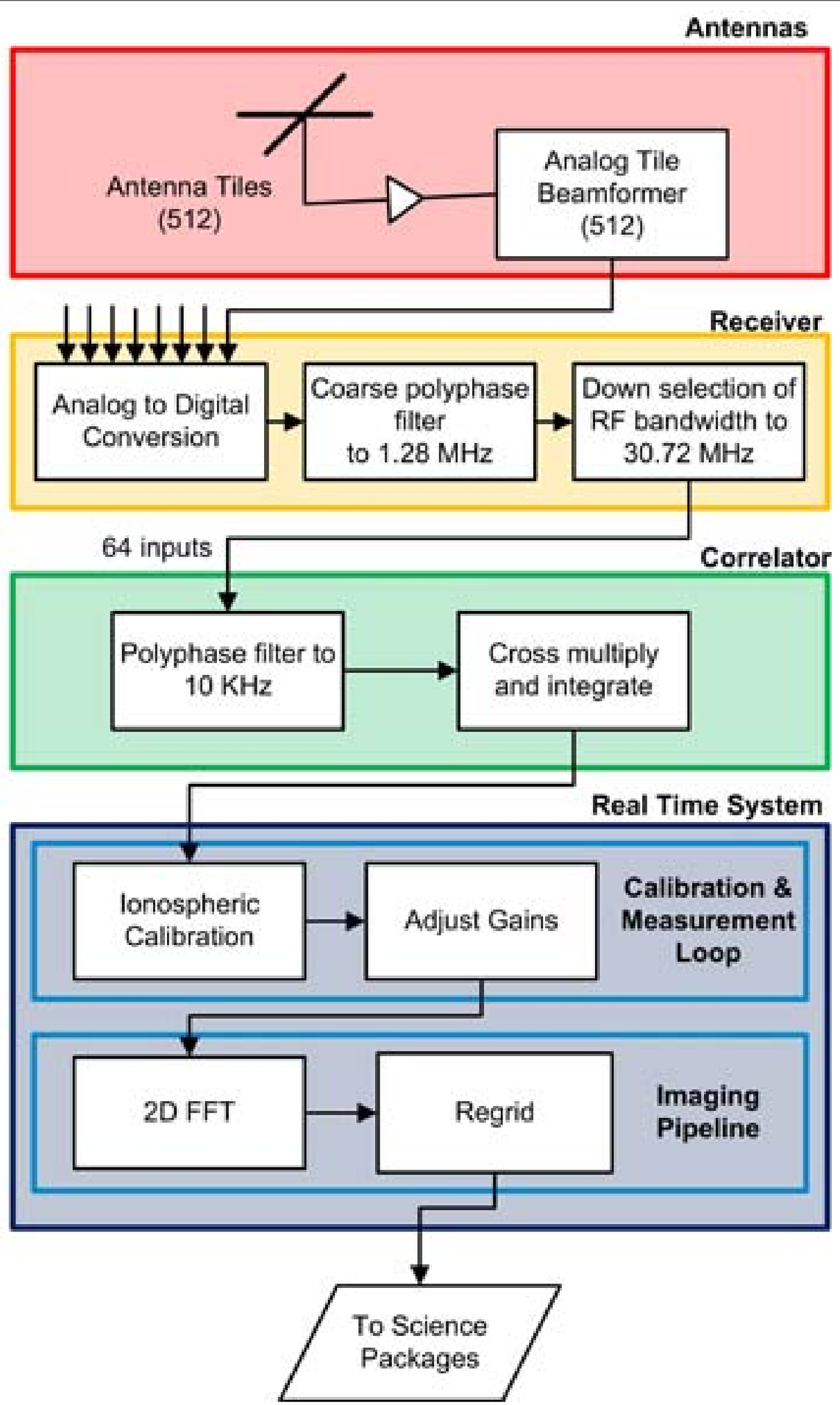
FPGAs
Virtex 4 SX35

16 kW

64 x GPU
cluster

30 kW

46 kW total

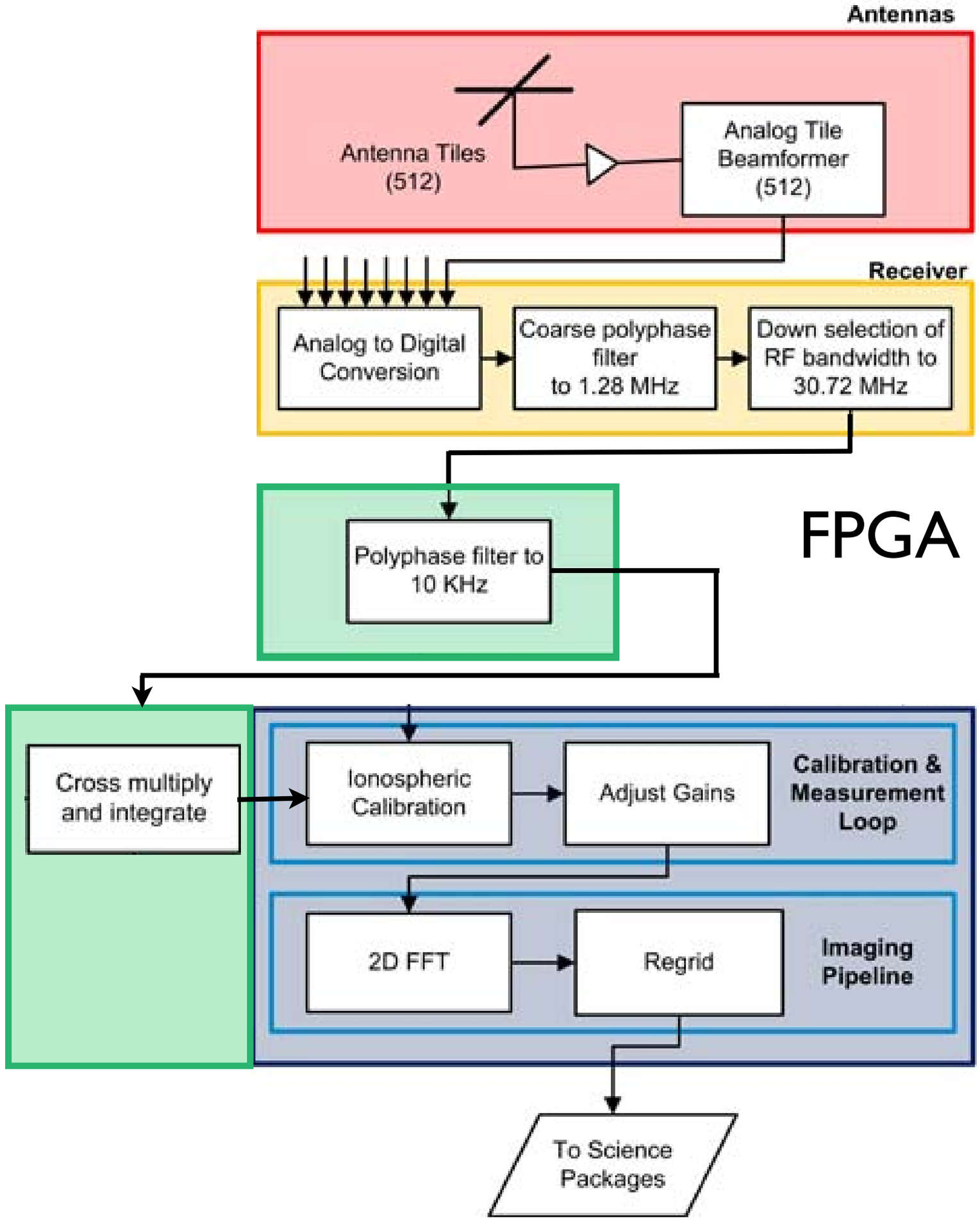


FPGA F-engine 4 kW

128x GPU X-engine 42 kW

64x GPU 30 kW

76 kW total

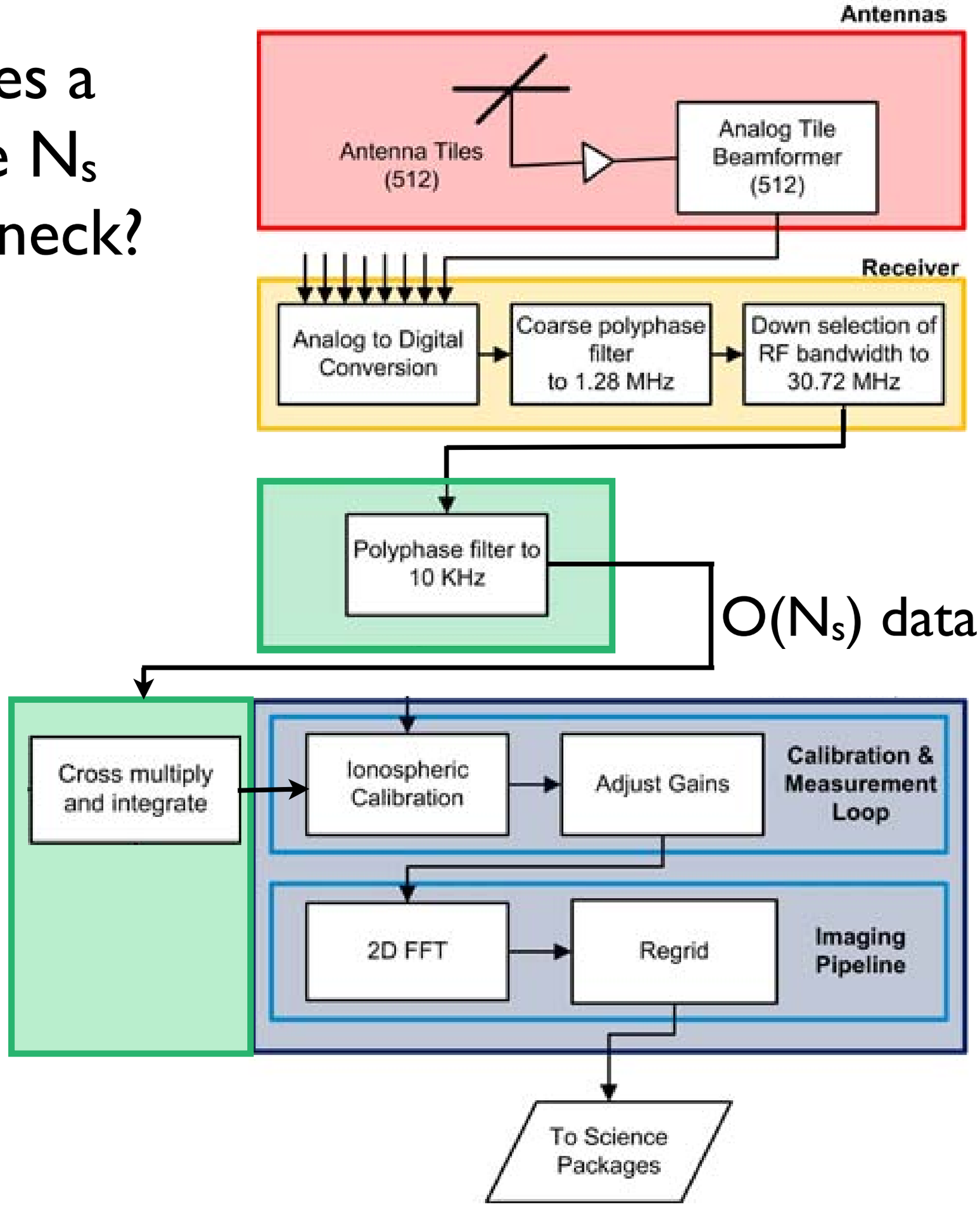


FPGA F-engine (4 kW)

57 kW
192xGPU

61 kW total

Solves a large N_s bottleneck?

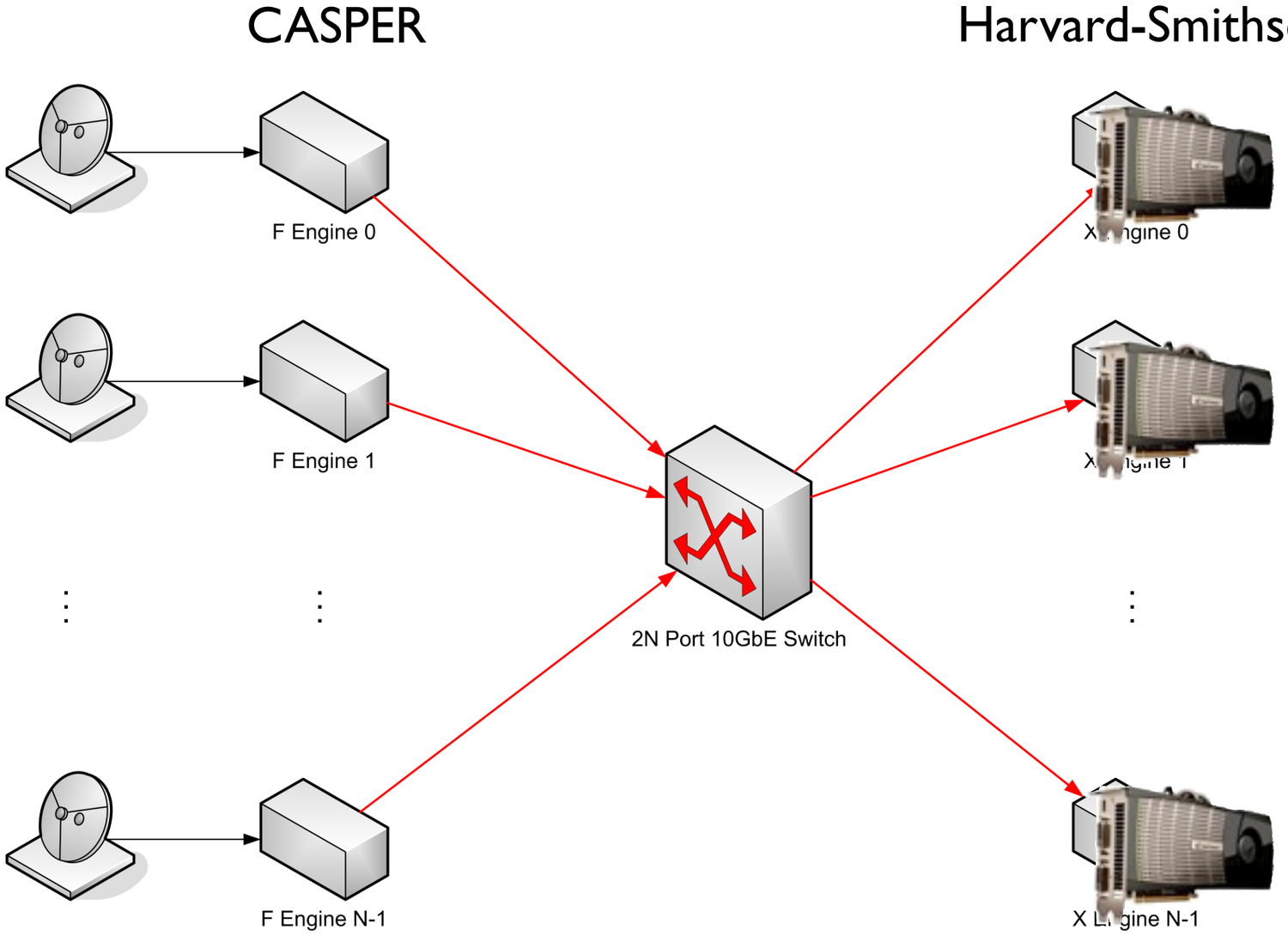


$O(N_s)$ data

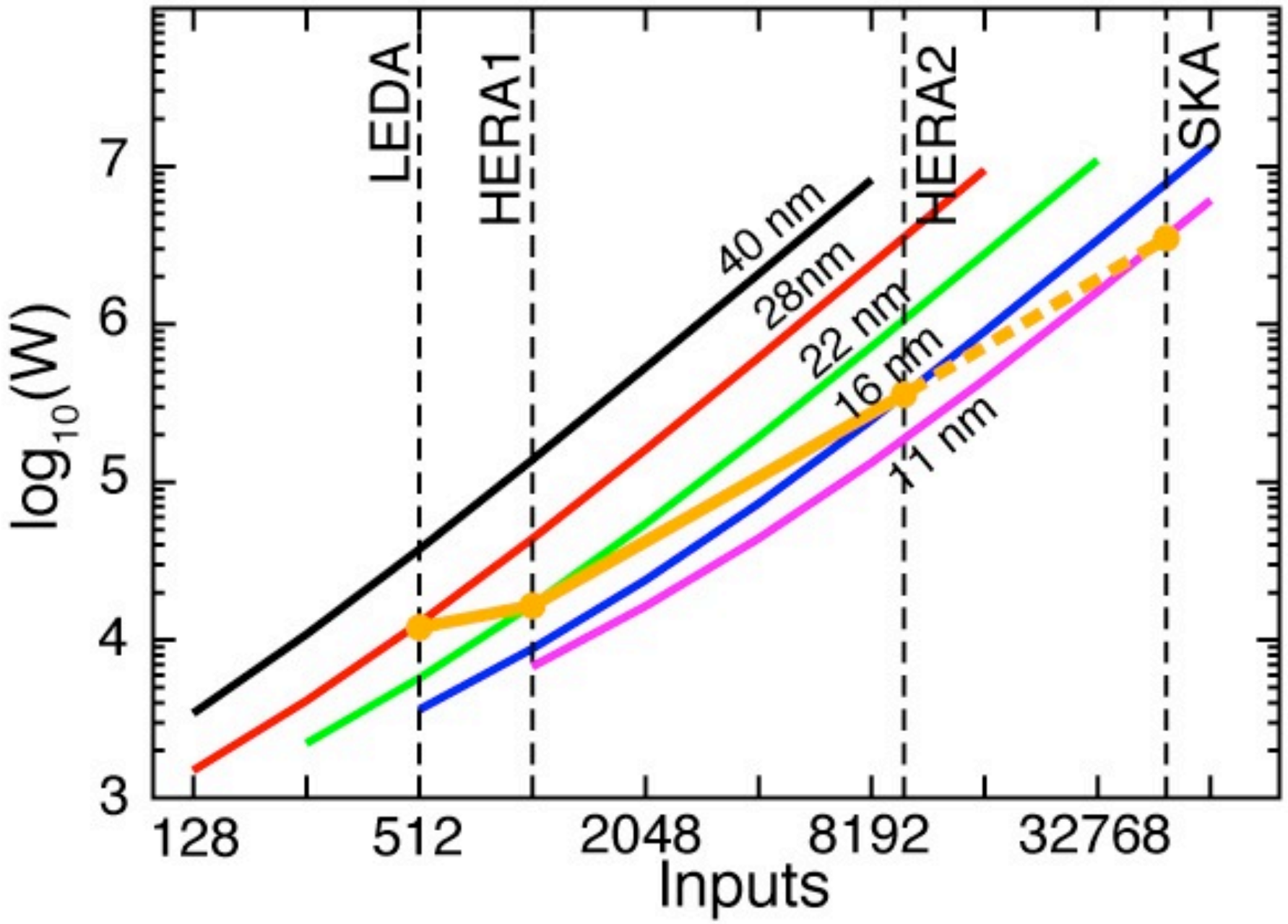
$O(N_s^2)$ data kept in GPU memory

Hybrid Correlator

- Proposed correlator for LEDA



Scaling to the Future



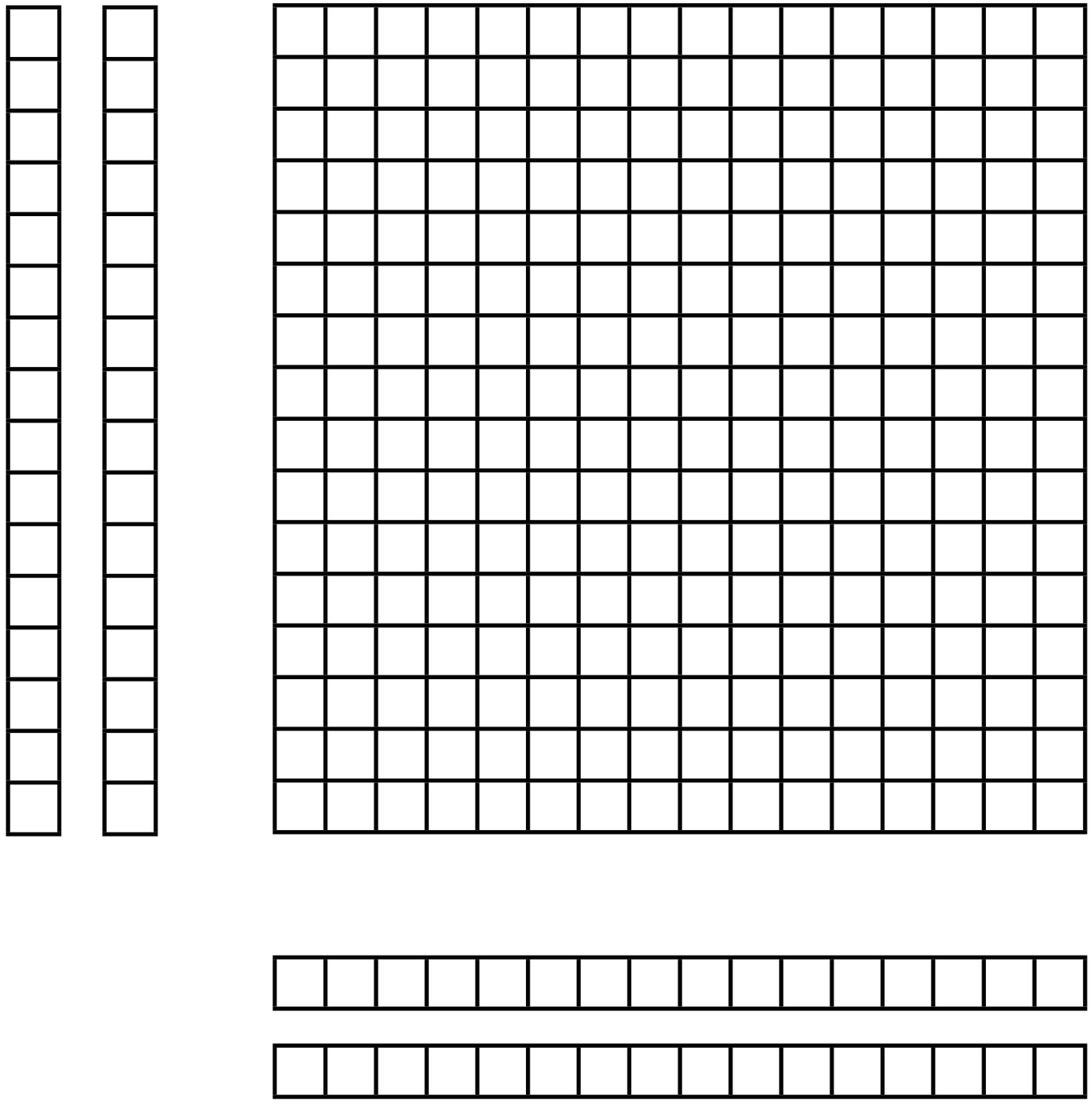
Summary and Conclusion

- X-engine is a perfect match to the GPU
  - 78% peak performance
- Low cost and development time
  - Easy to keep with bleeding edge
- Not (yet) power competitive with FPGAs
- Future: hybrid correlators?
- Combine X-engine with Calibration and Imaging

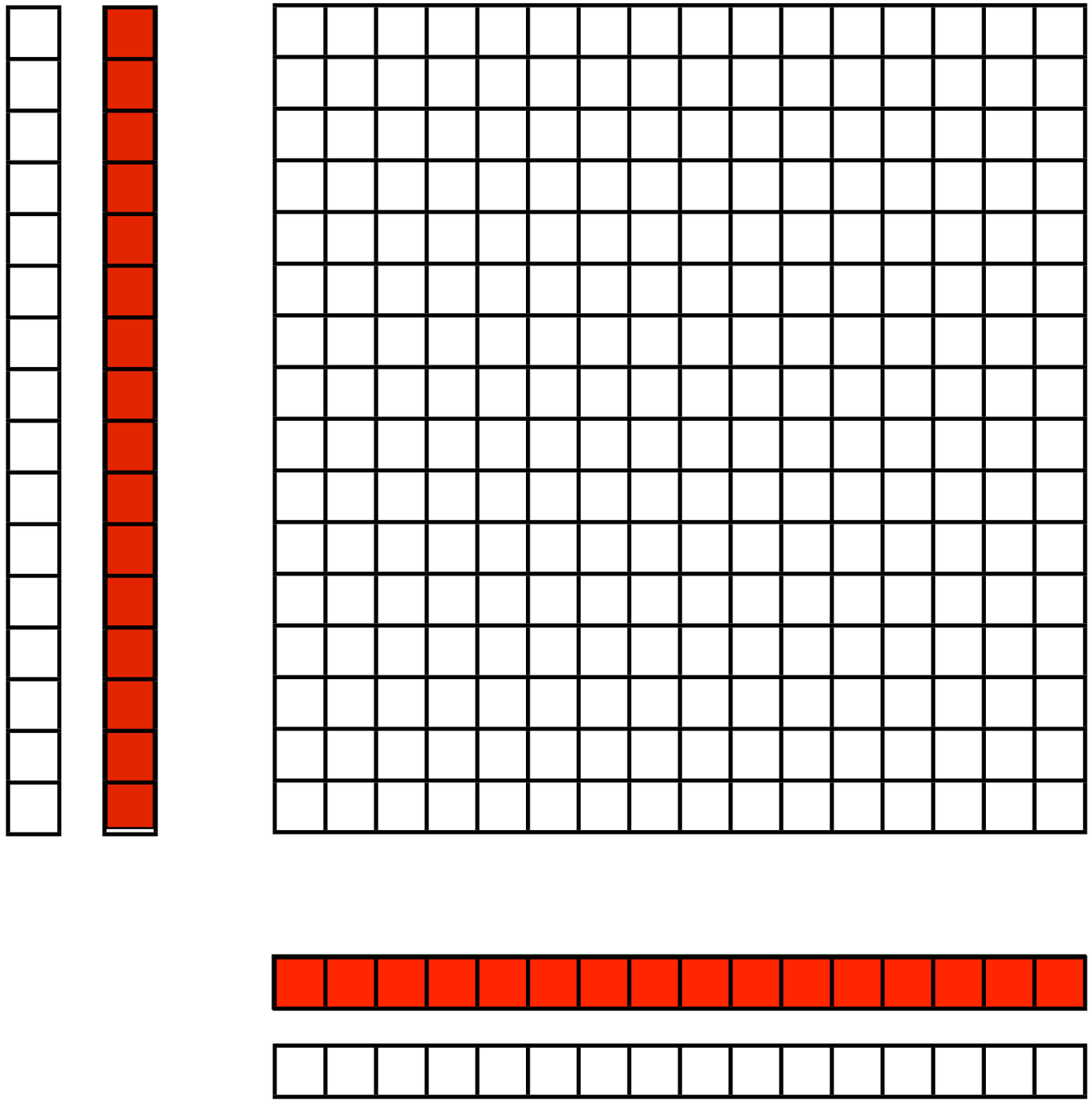
Optimization lessons

- Shared memory AND register tiling critical
- Minimize integer arithmetic
  - Precalculate offsets
  - Texture lookup
- Thread synchronization is costly
- The compiler doesn't always know what it's doing

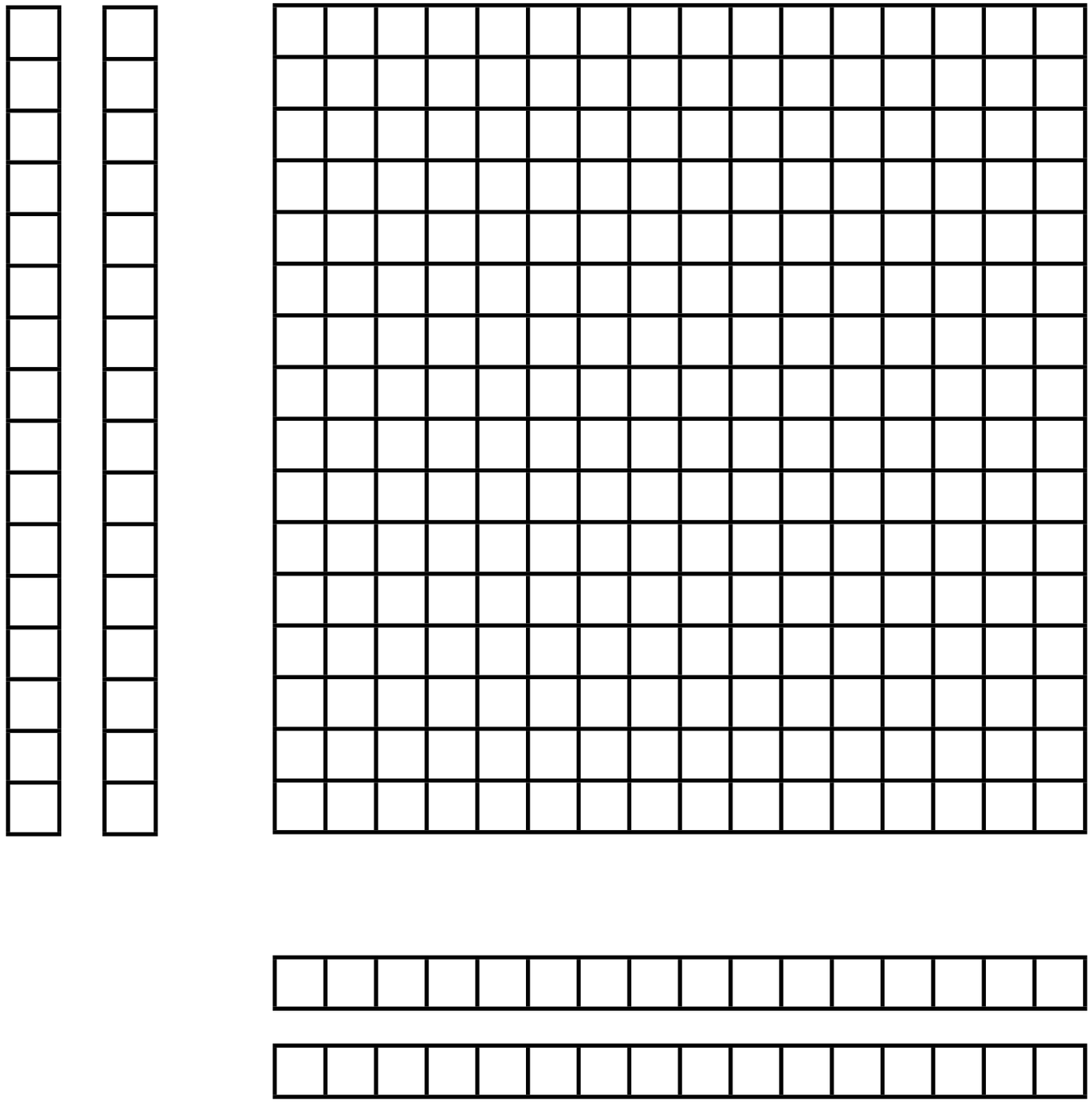
Double buffer shared memory storage to reduce synchronization



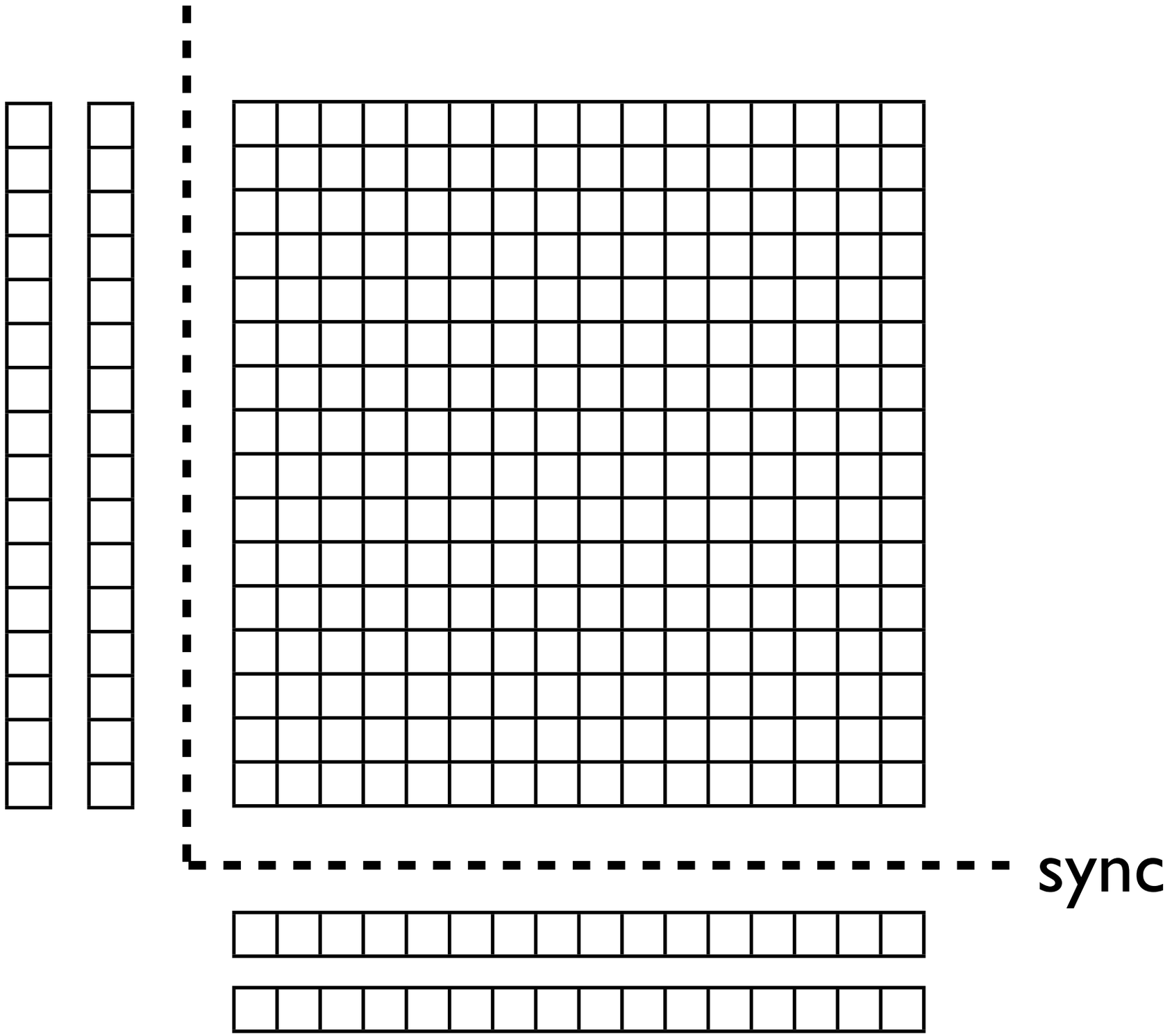
Double buffer shared memory storage to reduce synchronization



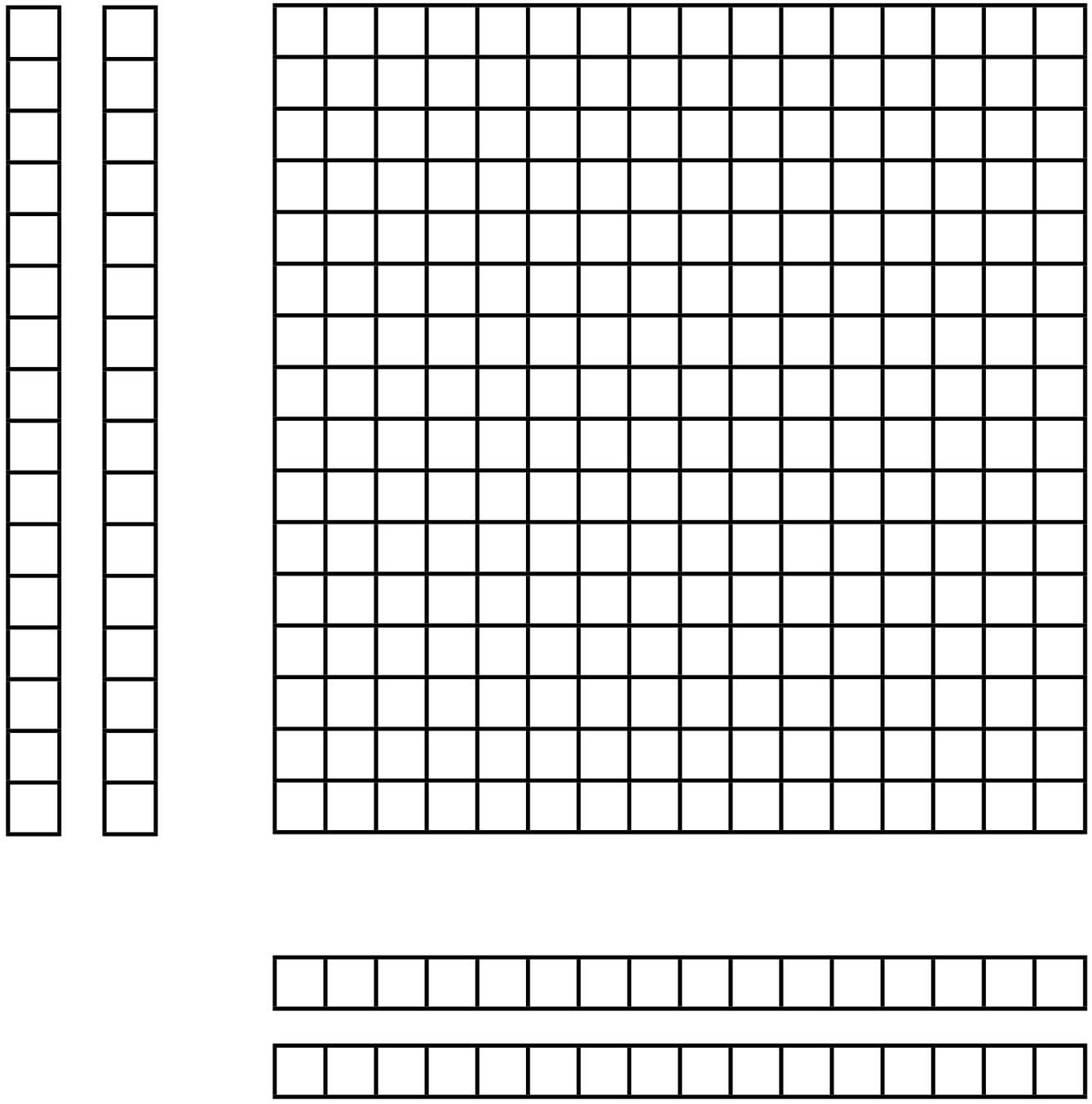
Double buffer shared memory storage to reduce synchronization



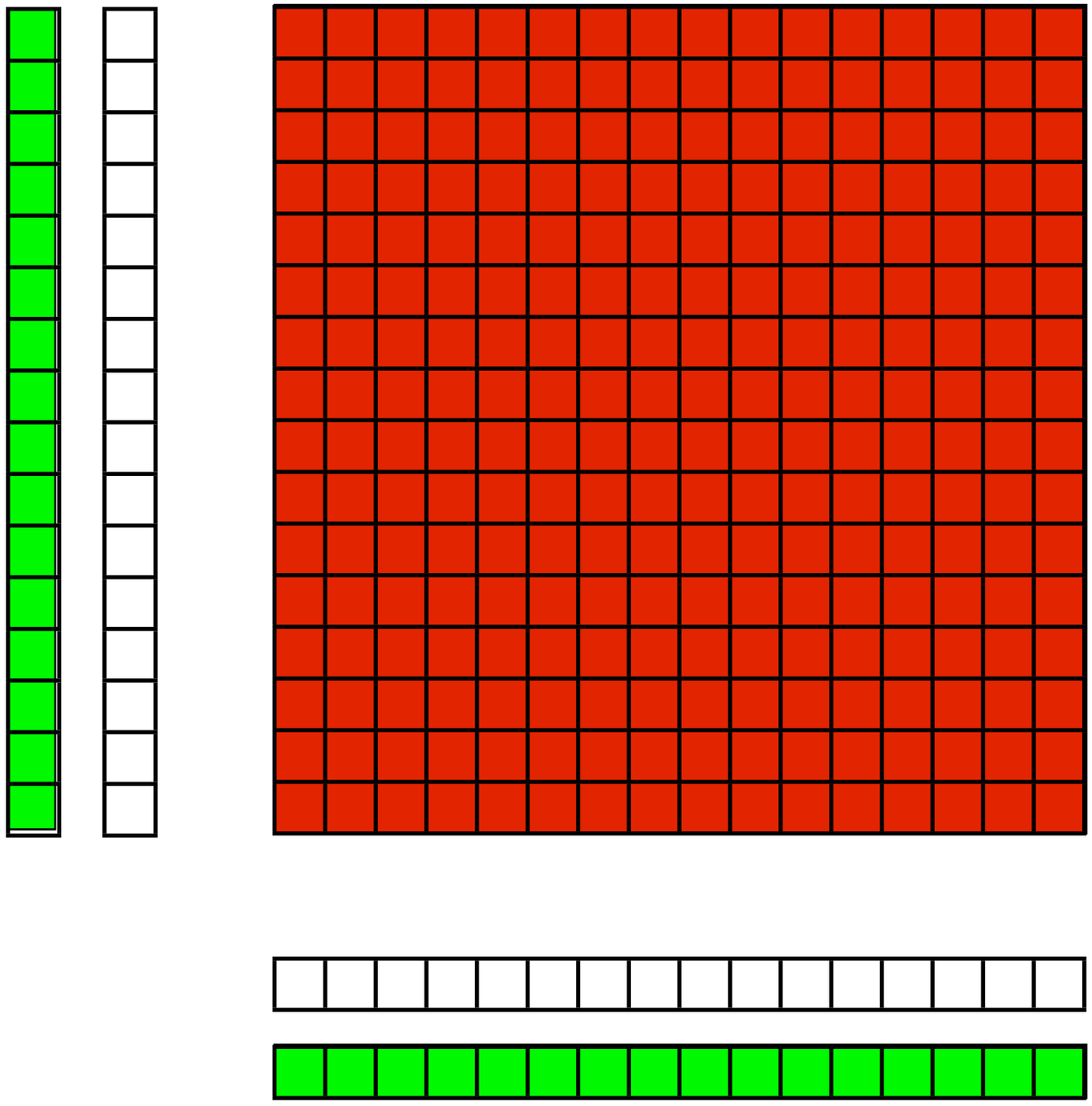
Double buffer shared memory storage to reduce synchronization



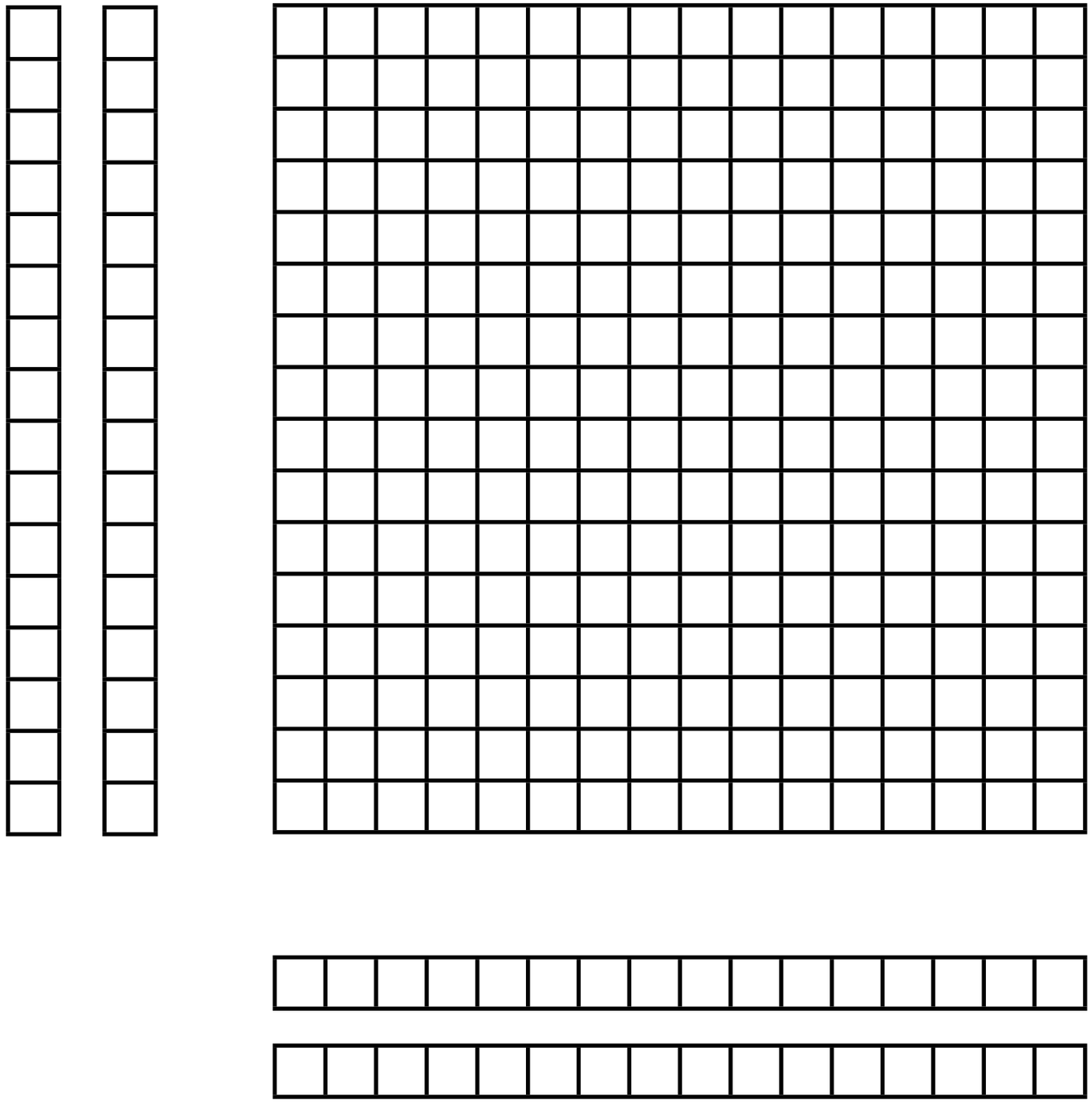
Double buffer shared memory storage to reduce synchronization



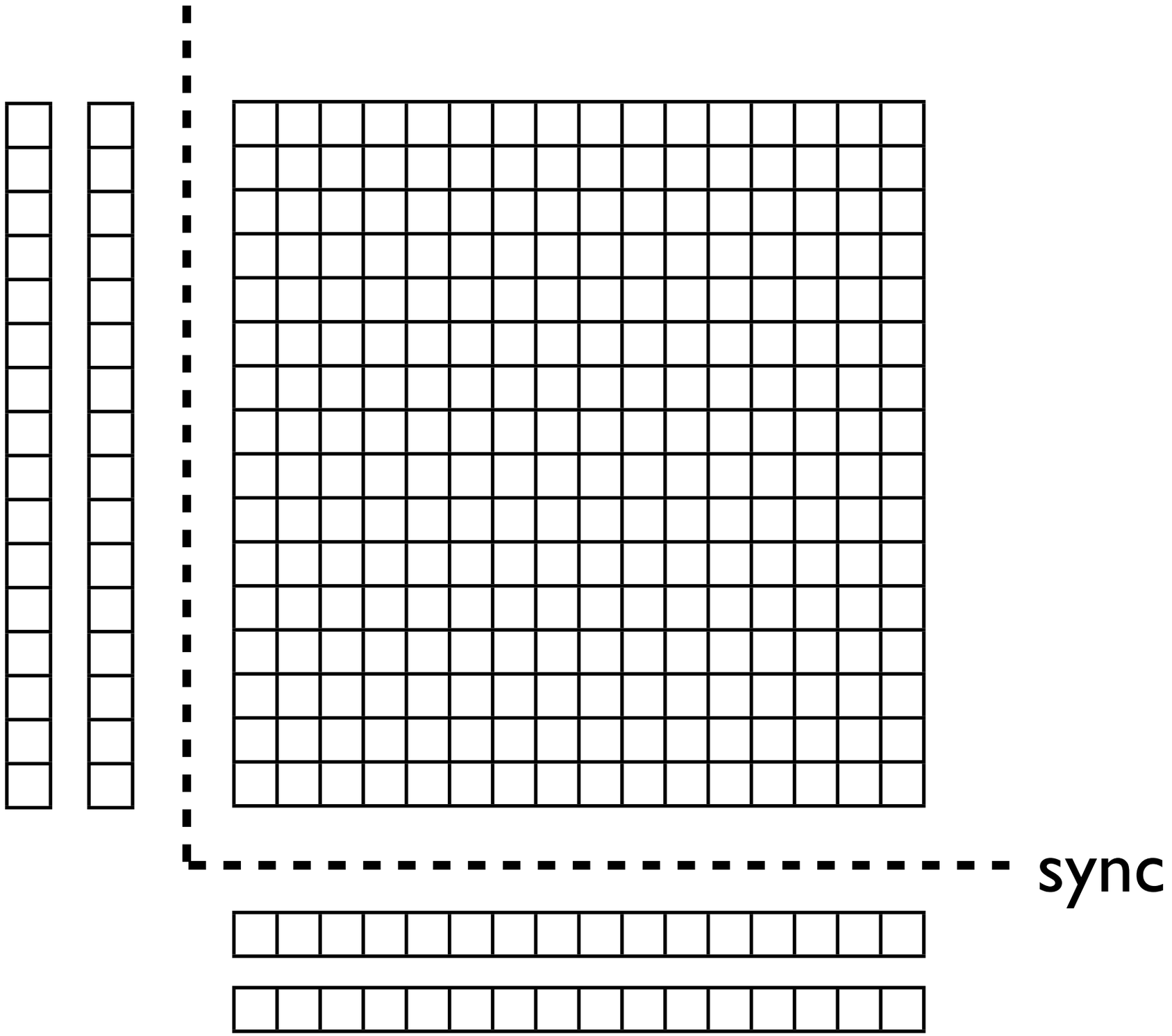
Double buffer shared memory storage to reduce synchronization



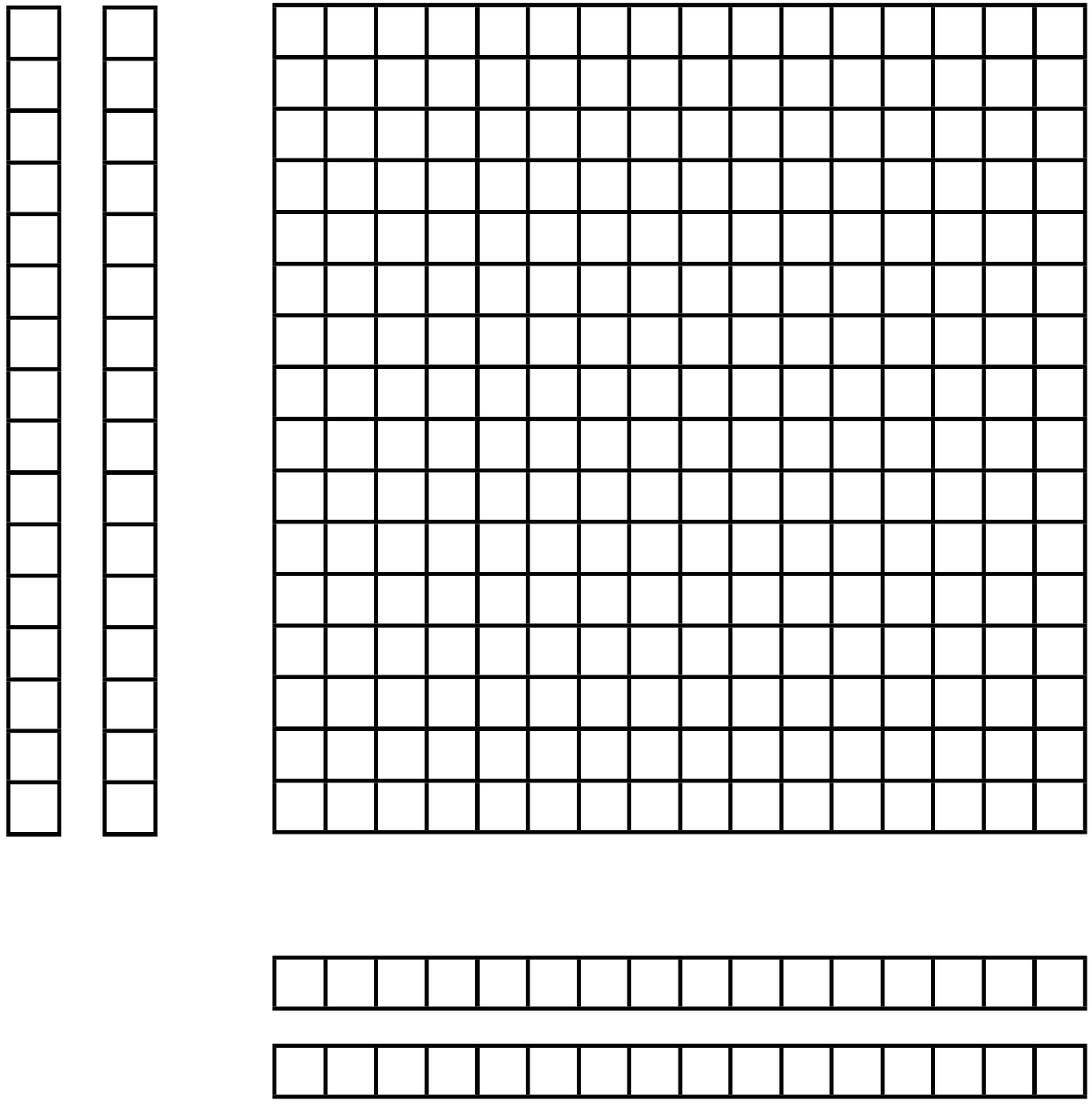
Double buffer shared memory storage to reduce synchronization



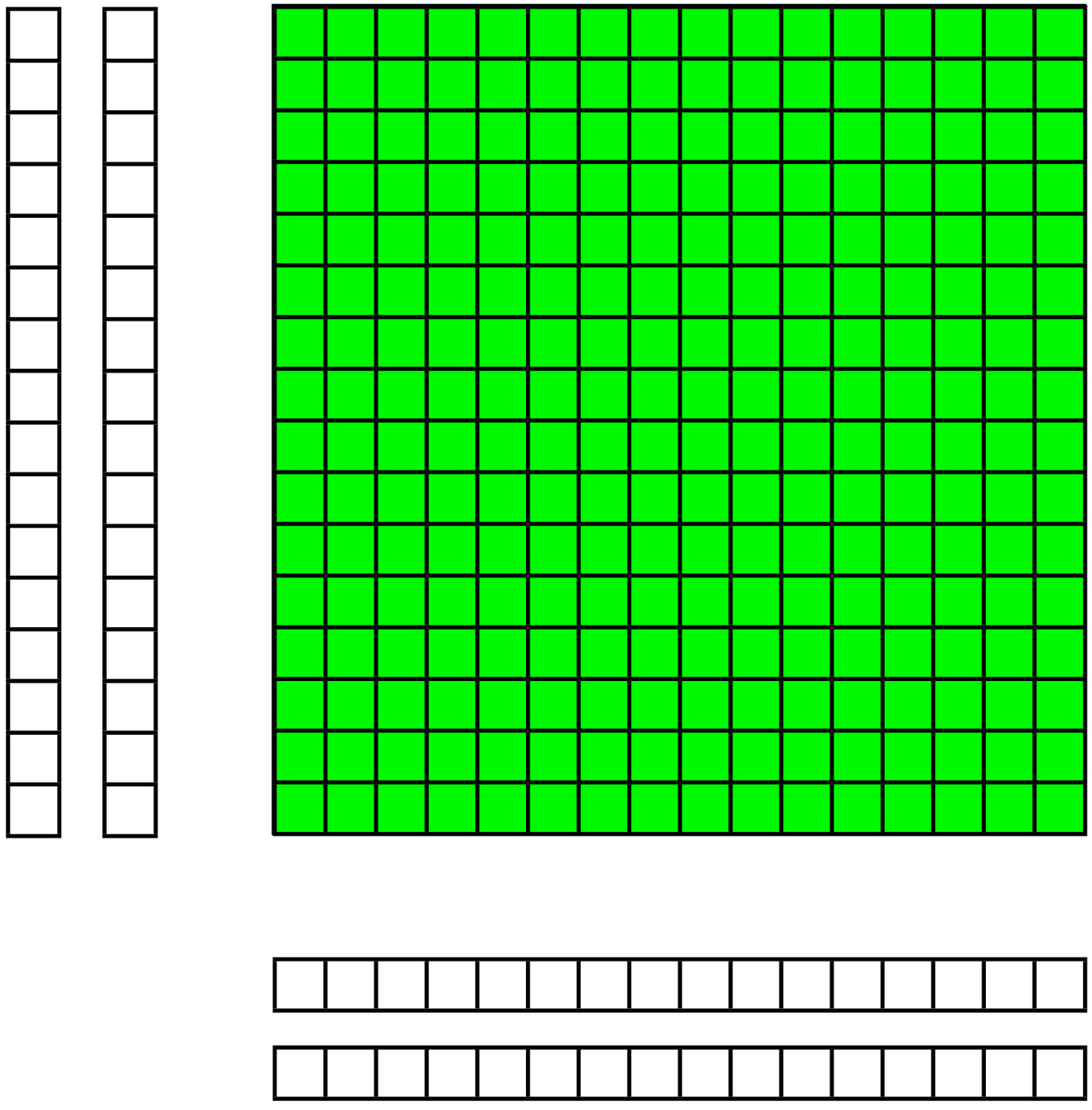
Double buffer shared memory storage to reduce synchronization



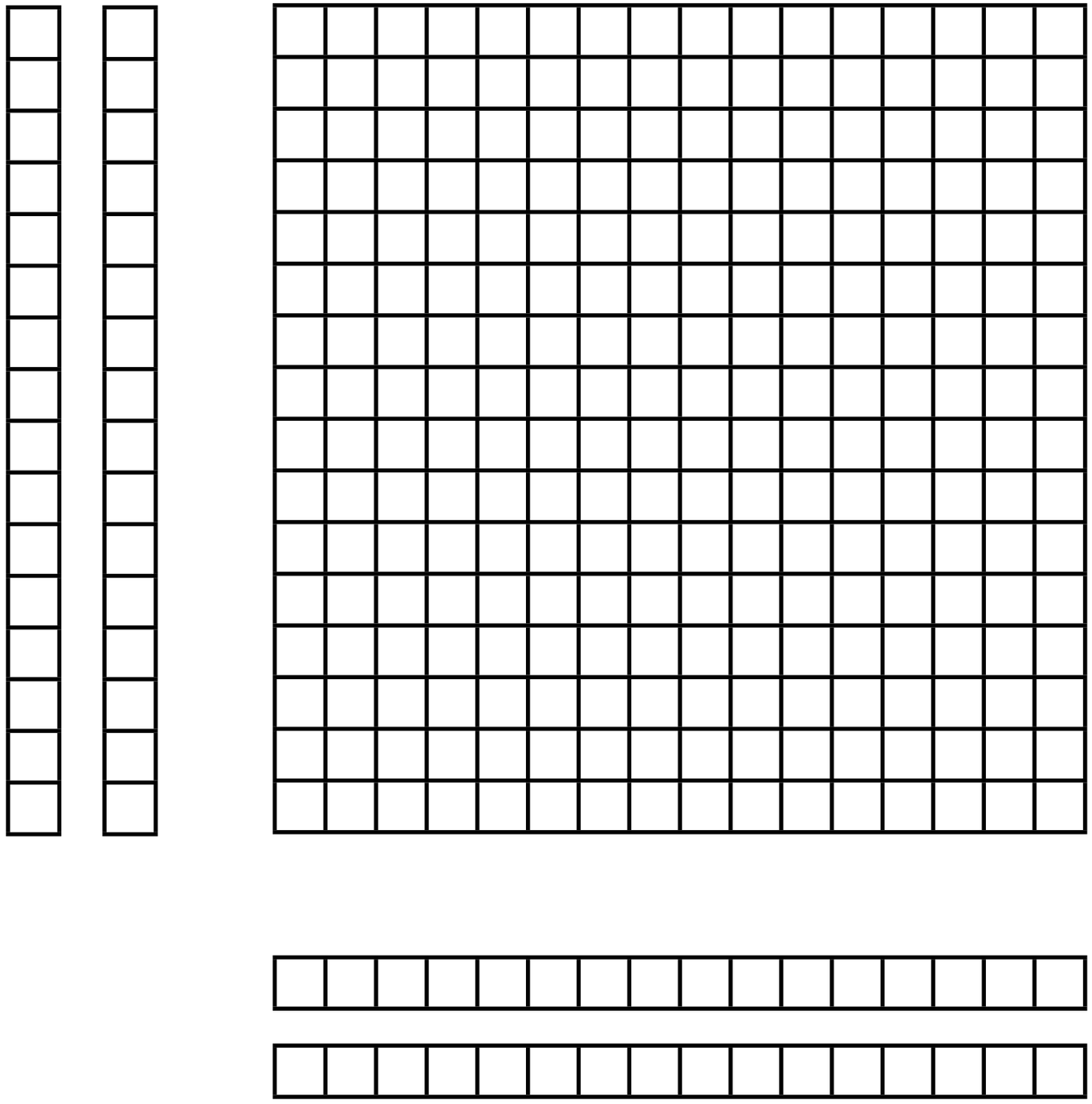
Double buffer shared memory storage to reduce synchronization



Double buffer shared memory storage to reduce synchronization



Double buffer shared memory storage to reduce synchronization



Don’t trust compilers

- Compare these “identical” code fragments

```
a += b*c + d*c + e*f + g*h;
```

770 GFLOPS

```
a += b*c;
a += d*c;
a += e*f;
a += g*h;
```

1020 GFLOPS