Monte-Carlo simulations of Ginzburg-Landau model on GPU

with comparison with CPU

Piotr Bialas

Faculty of Physics, Astronomy and Computer Science, Jagiellonian University
Department of Game Technology

1 March 2013

J. Kowal, A. Strzelecki

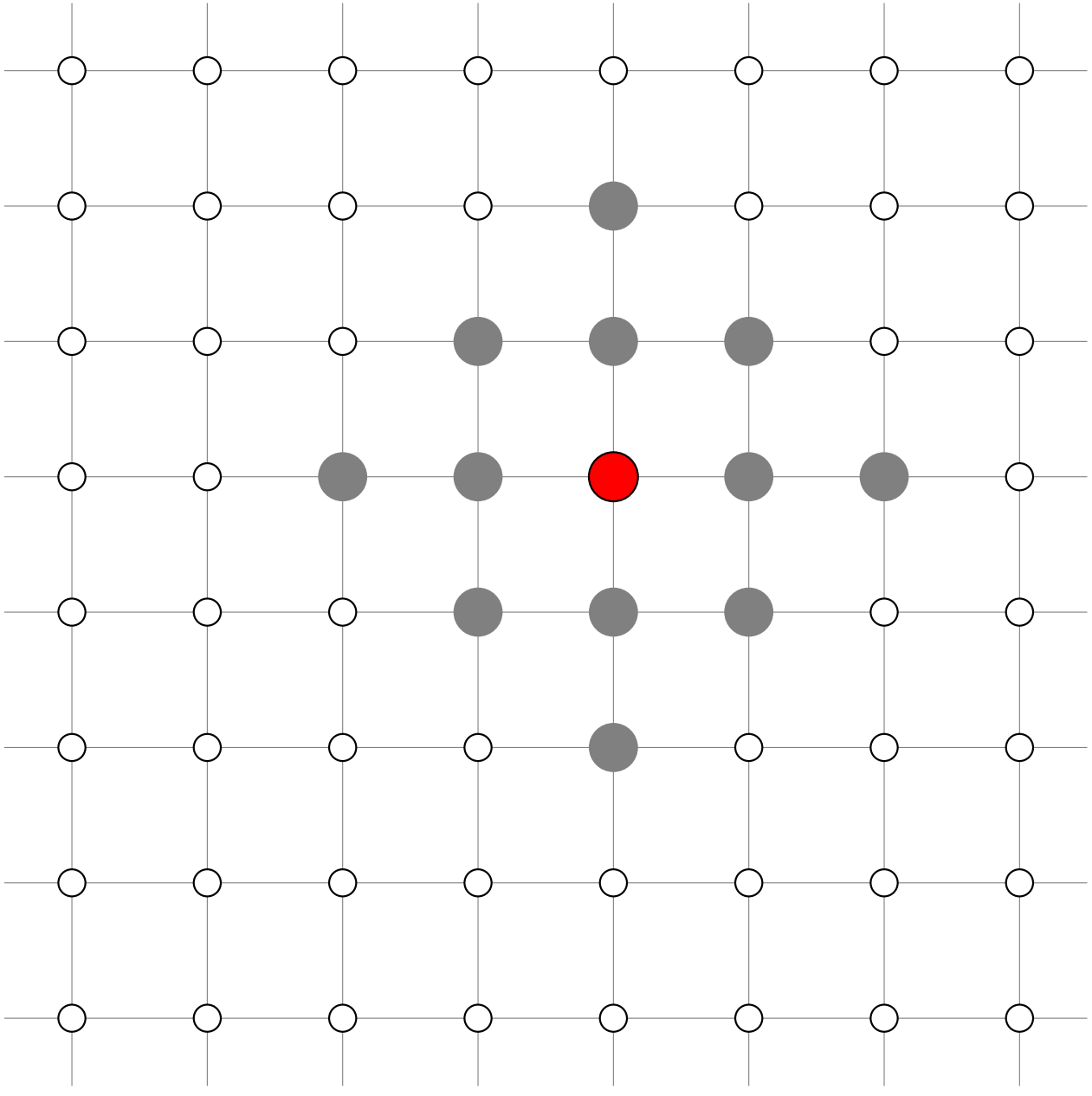
Outline

Ginsburg-Landau model ($\vec{\varphi}^4$)

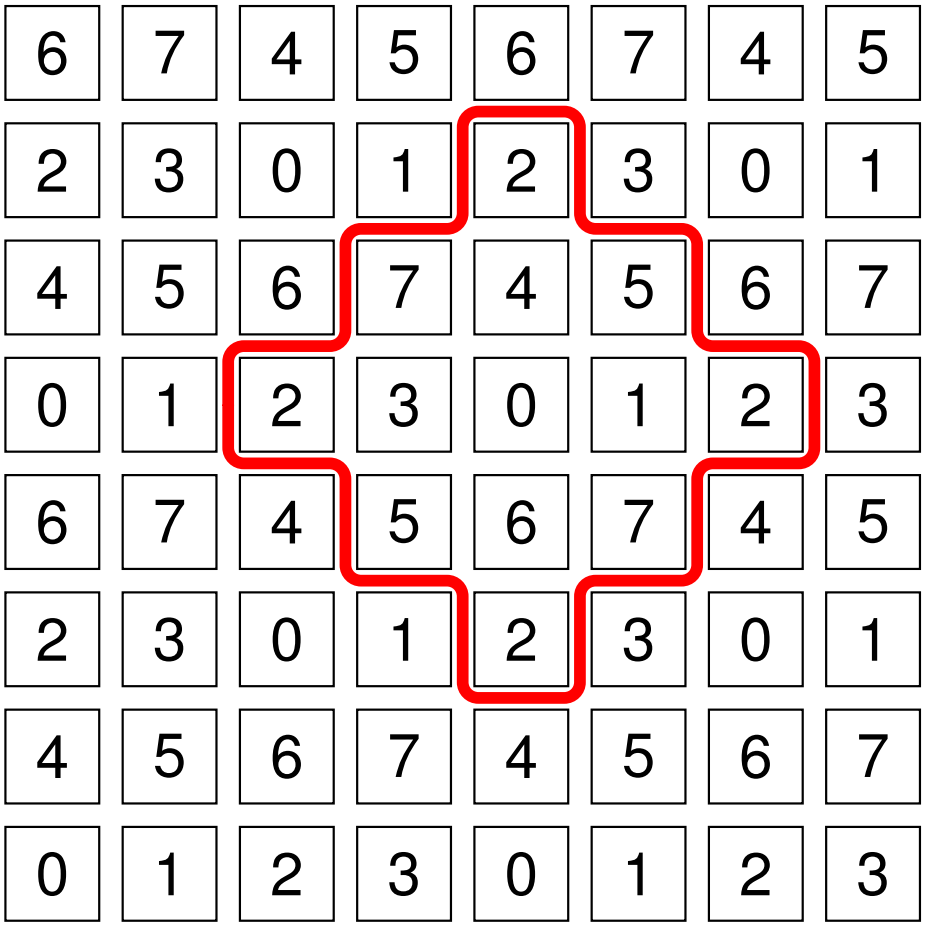
$$H[\varphi] = \sum_{\vec{i}} \left(\frac{1}{2} \sum_{\mu=1}^{d} (\vec{\varphi}(x_{\vec{i}} + \vec{a}_\mu) - \vec{\varphi}(x_{\vec{i}}))^2 \right.$$
$$+ \frac{\mu^2}{2} |\vec{\varphi}(x_{\vec{i}})|^2 + \frac{g}{24} (|\vec{\varphi}(x_{\vec{i}})|^2)^2$$
$$\left. + \frac{1}{2\Lambda} \left(\sum_{\mu=1}^{d} (\vec{\varphi}(x_{\vec{i}} + \vec{a}_\mu) - 2\vec{\varphi}(x_{\vec{i}}) + \vec{\varphi}(x_{\vec{i}} - \vec{a}_\mu)) \right)^2 \right).$$

$$P[\varphi] \propto \exp\left(-H[\varphi]\right)$$

Ginsburg-Landau model ($\vec{\varphi}^4$)



Parallel partitions

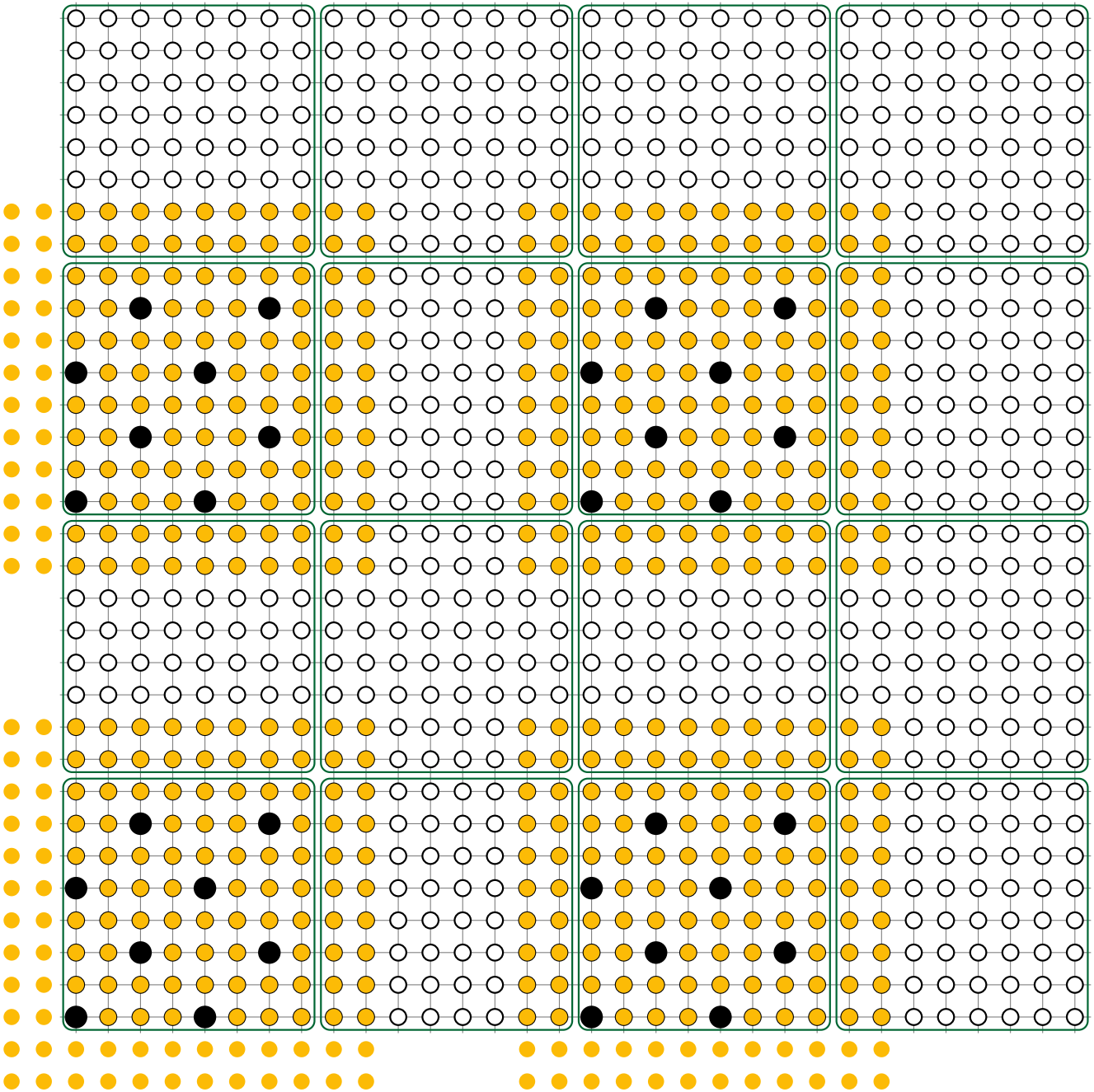


CUDA architecture – SIMT

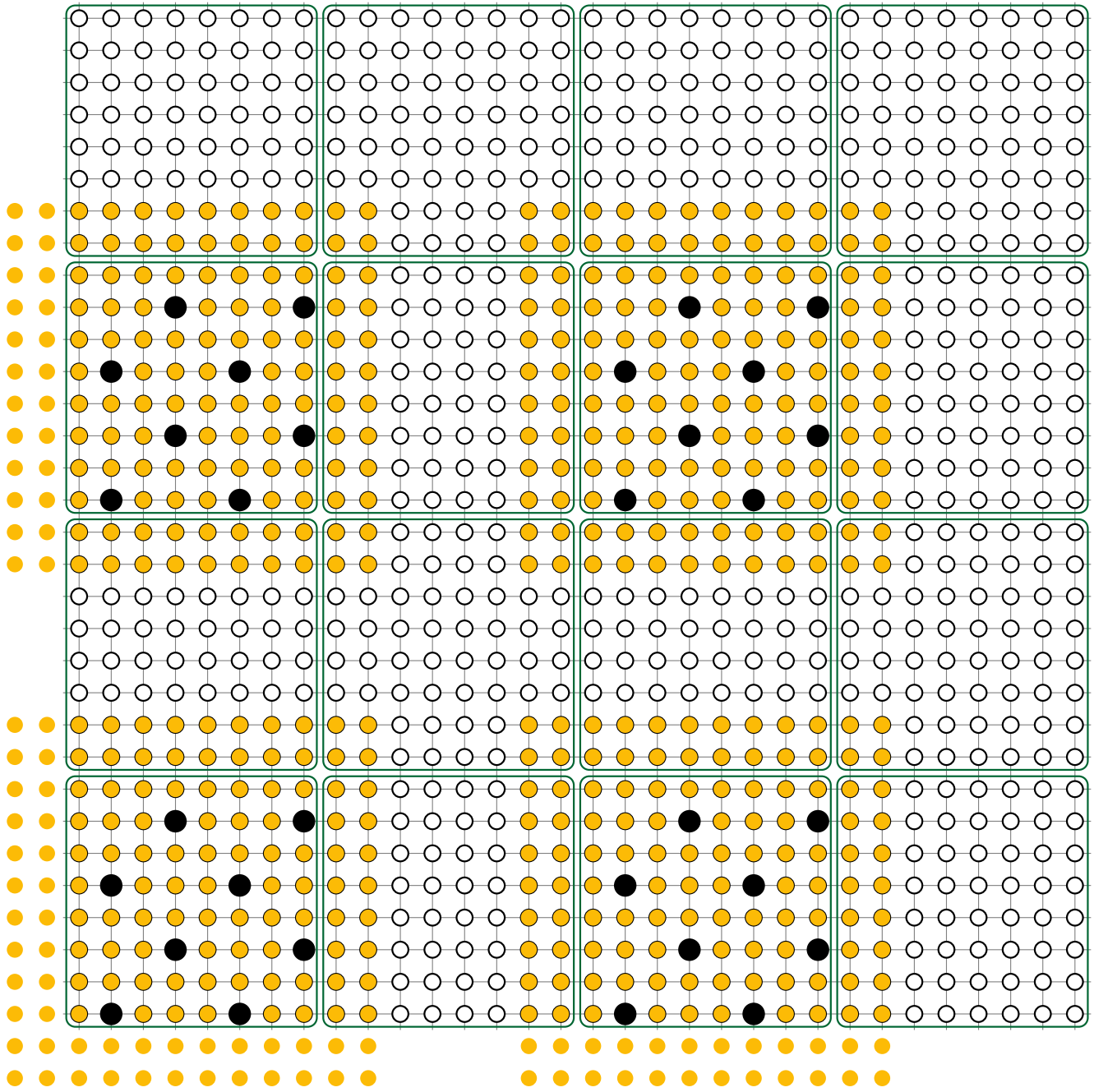
- GPUs are massively parallel (*cores* $>$ 500).
- Logically CUDA program is run as a collection of *lightweight* concurrent *threads*.
- Every thread is executing same program called *kernel*.
- Threads are grouped in thread blocks.
- Usually there should be much more threads then cores.

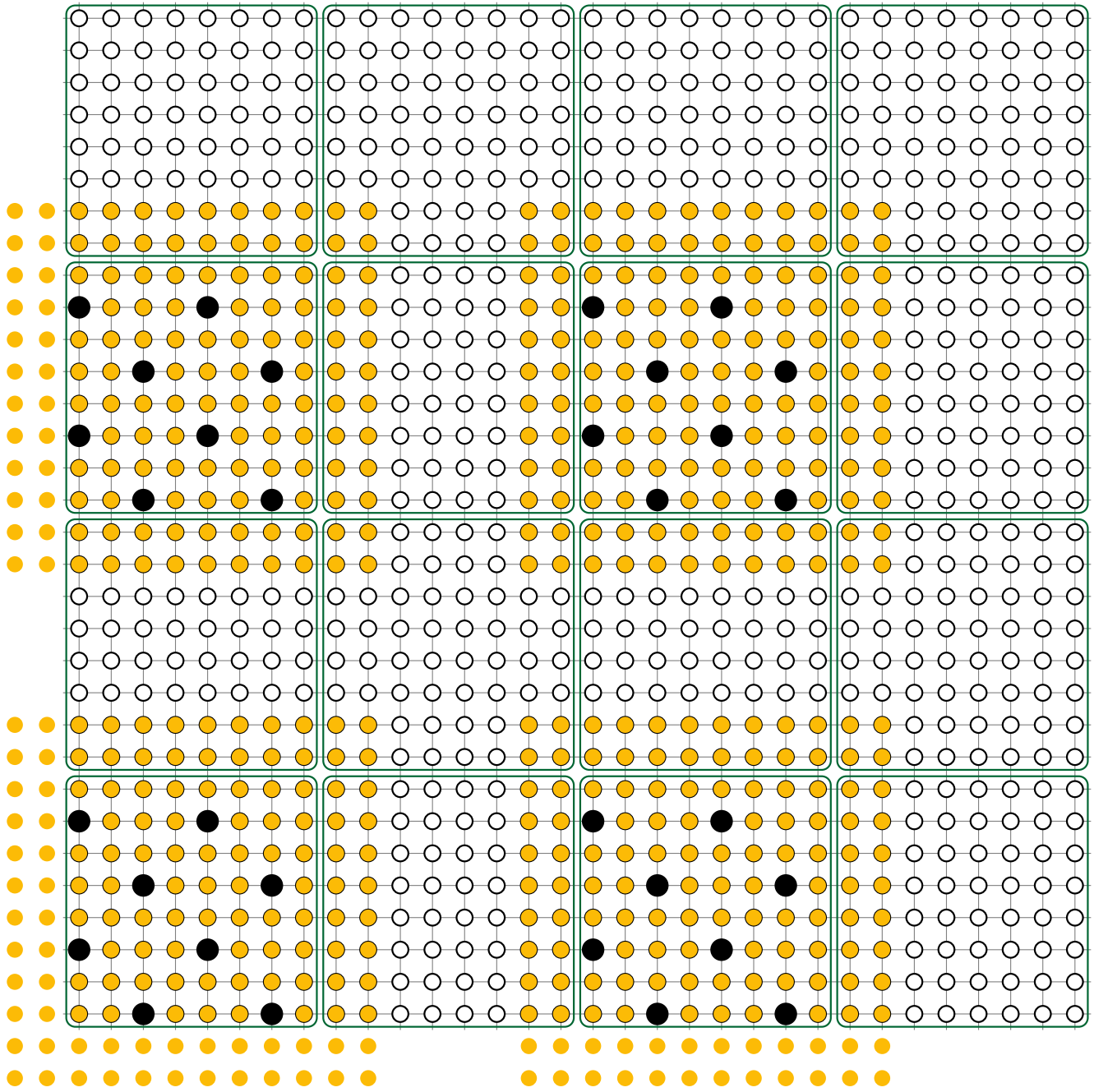
Memory hierarchy

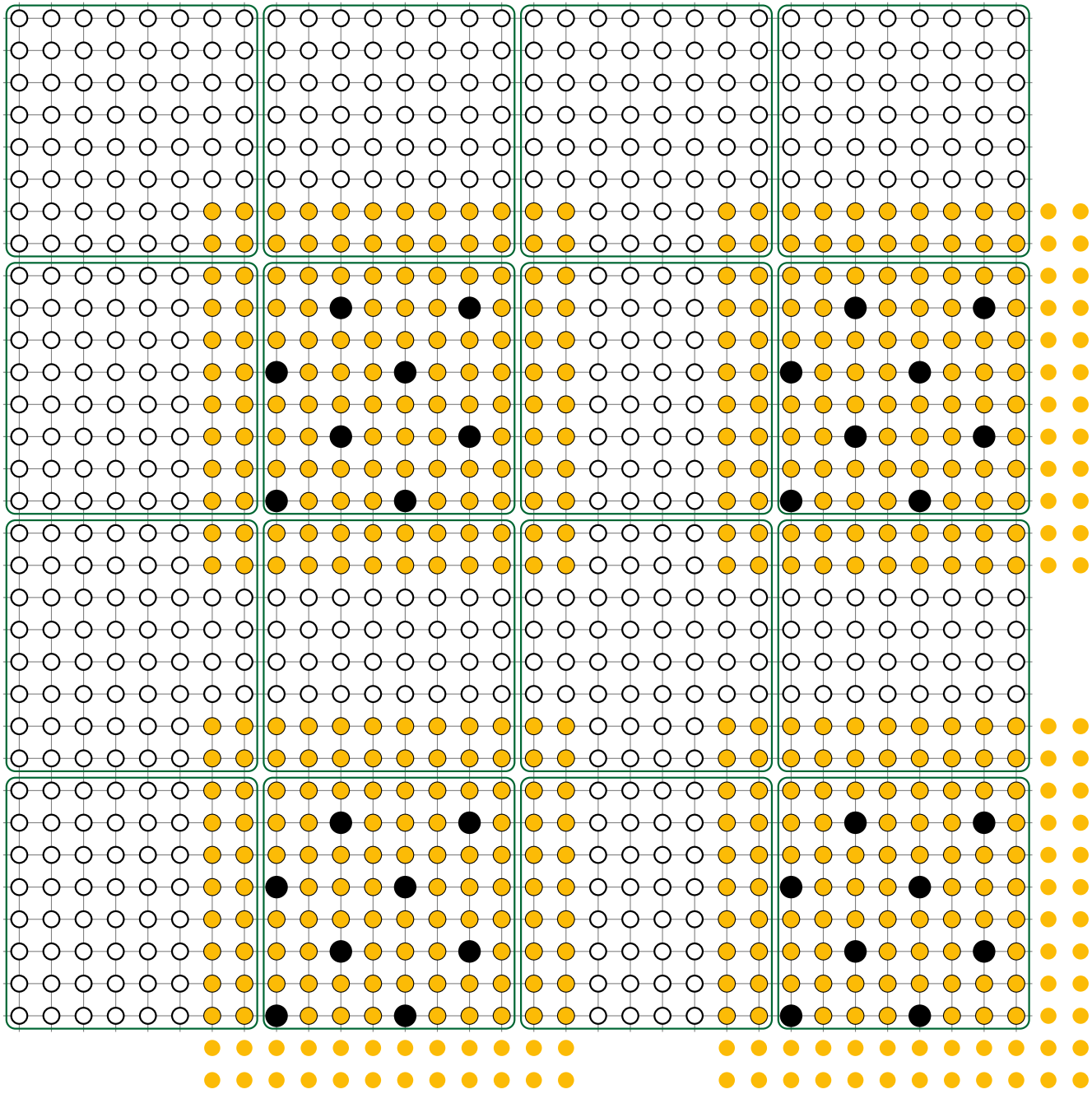
- Registers.
  - 8.07 TB/s
- Shared memory/L1cache
  - 4 x 32 x 15 x 1401MHz/2 = 1.3TB/s
- Global memory
  - 156GB/s
- CPU memory
  - 5GB/s

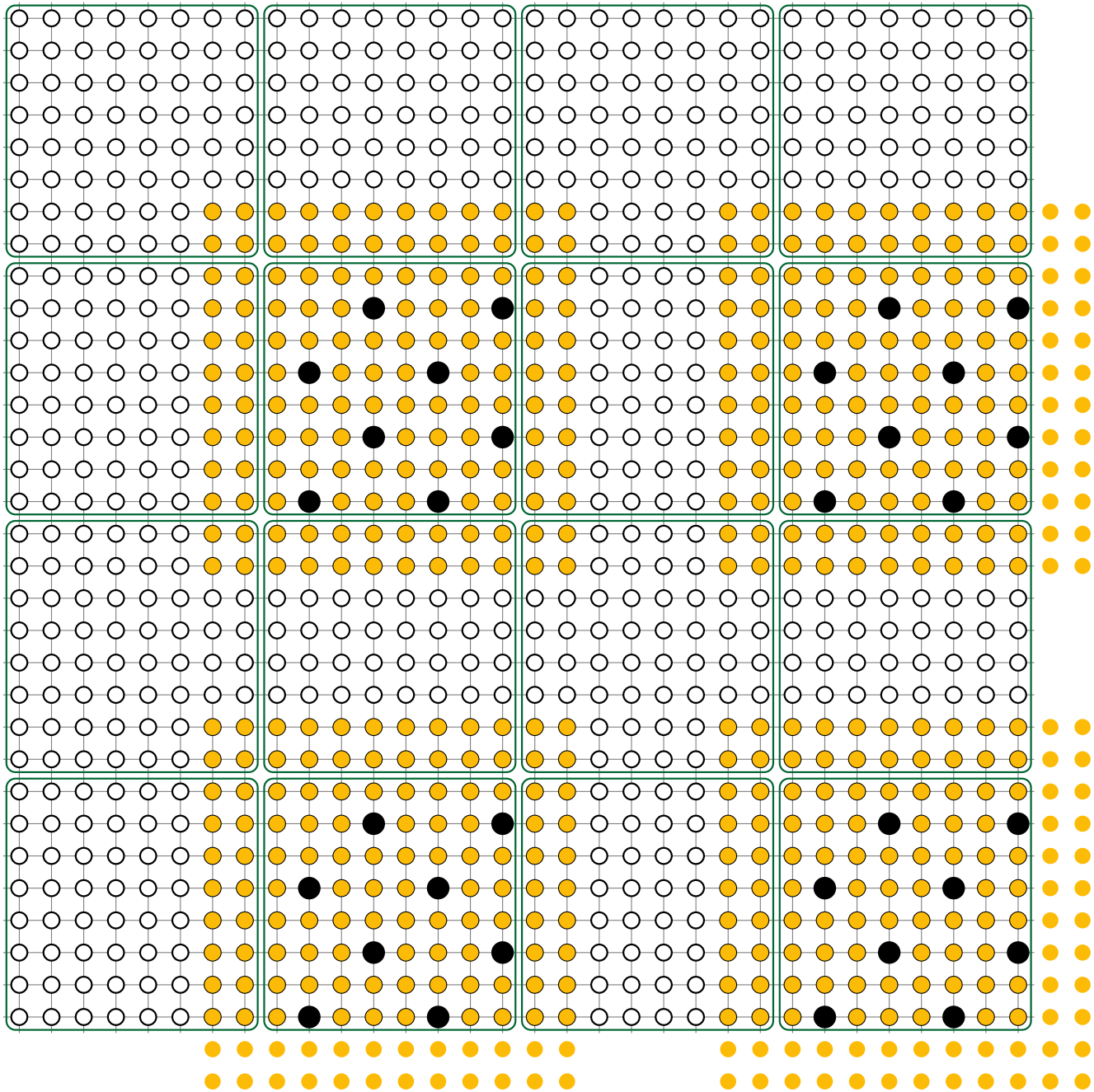


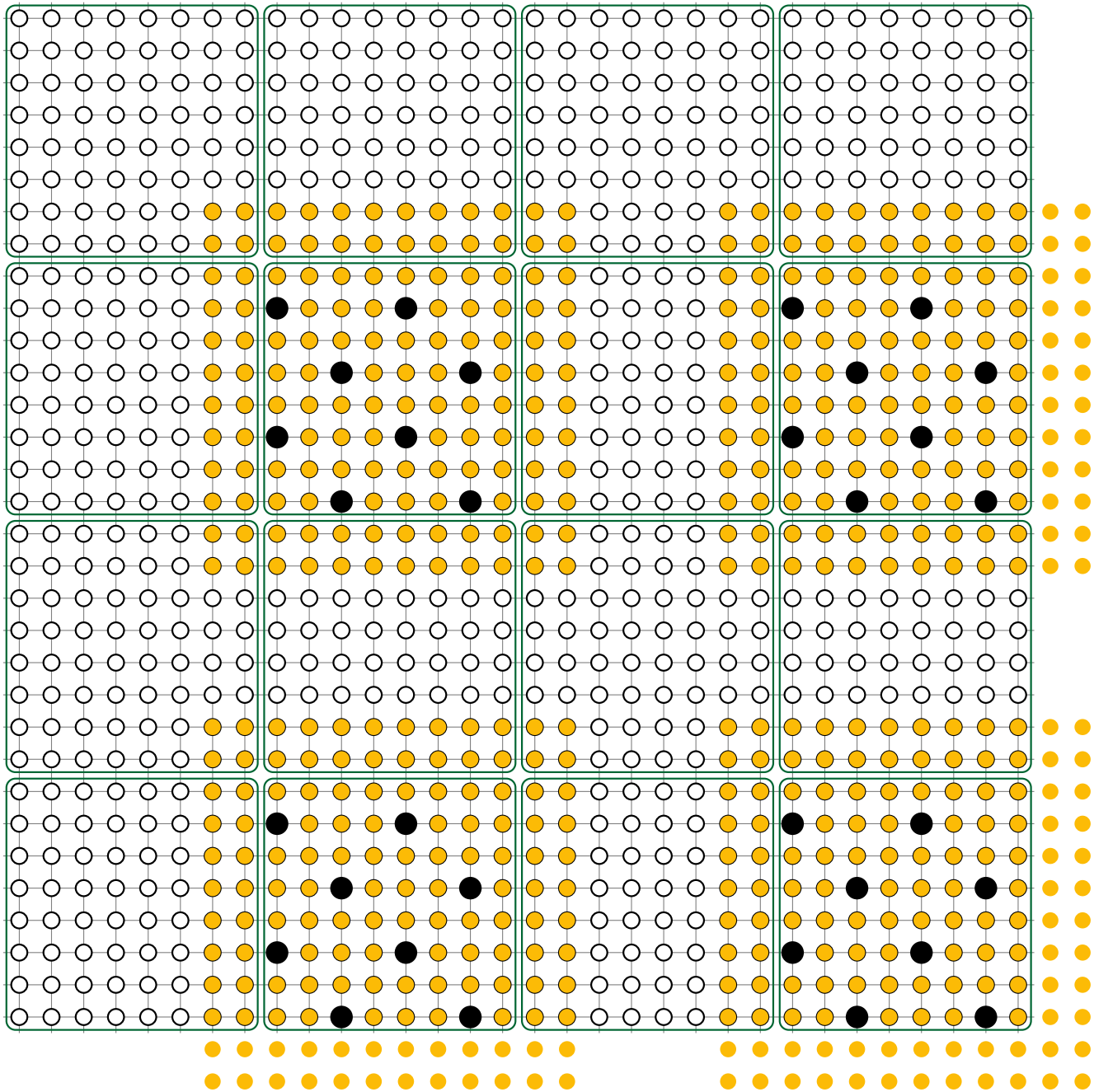
M. Weigel, J. Comput. Phys. 231 (2012) 3064.









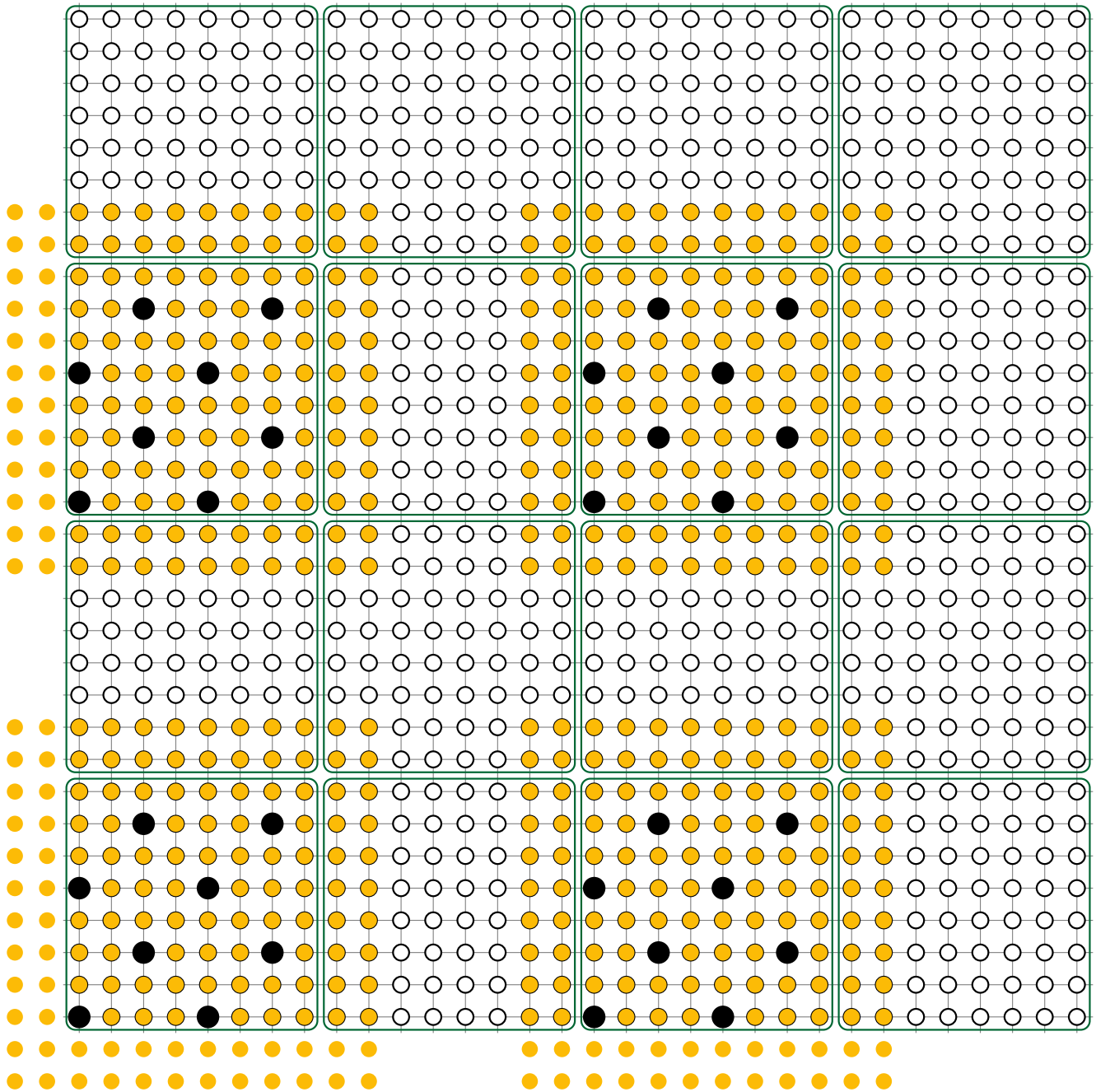


CPU

- First measurements shoved over 300 fold performance increase over CPU!
- Not fair.
- Compared to a non–parallel, non-vectorized program.

CPU implementation

- We use same structure as on GPU.
- We use OpenMP to distribute lattice blocks across cores.
- In each block we use SSE instructions to process 4 or 8 lattice points in parallel.



AVX/SSE - Objective

- Drop in vector replacement for scalar type variables
- Use compiler AVX/SSE code-gen via `__attribute__((vector_size(ncomp*tsize)))`
- Extend operators for simple POD types in C/C++!?

AVX/SSE - Solution

- Custom class wrapping `vector_size`
- Drop in vector replacement for scalar type
- Defines all operators available for scalar types
- Existing function just take vector arguments now
- Rest remains the same

```
action -= phi2*(quadratic_coef+gr*phi2)
```

Performance

	GPU[1]			CPU[2]			
size	time(ns)	Gflops	Gexps	time(ns)	Gflops	Glogs	boost
32^3	1.26	27.1	0.8	3.88	8.8	0.26	3.1×
64^3	0.17	206.8	6.1	3.09	11.0	0.32	18.8×
128^3	0.13	273.0	8.0	3.31	10.3	0.30	26.5×
256^3	0.11	302.0	8.9	3.33	10.3	0.30	29.5×
	peak	≈ 1000		peak	≈ 160		

[1] *GeForce GTX 470* 1.2 GHz, 1.25GB, 448 CUDA cores

[2] *Intel Core i5-2400S* 2.5 GHz, 4 core, 6MB cache, SSE4.x/AVX

Conclusions

- GPU has more computing power.
- GPU is more efficient: easier to reach closer to the peak.
  - Possible thanks to the use of fast shared memory
  - When counting on cache is just not enough
  - Shared memory simplifies indexing
- The CUDA (SIMT) execution model can be ported to CPU:
  - No shared memory, only cache, little control
  - No gather/scatter vector load operations
  - Vector assembly from memory is a bottleneck
  - Half-wide only (128-bit) vector integer operations

  - *Xeon* ϕ & unreleased *Haswell* address these issues