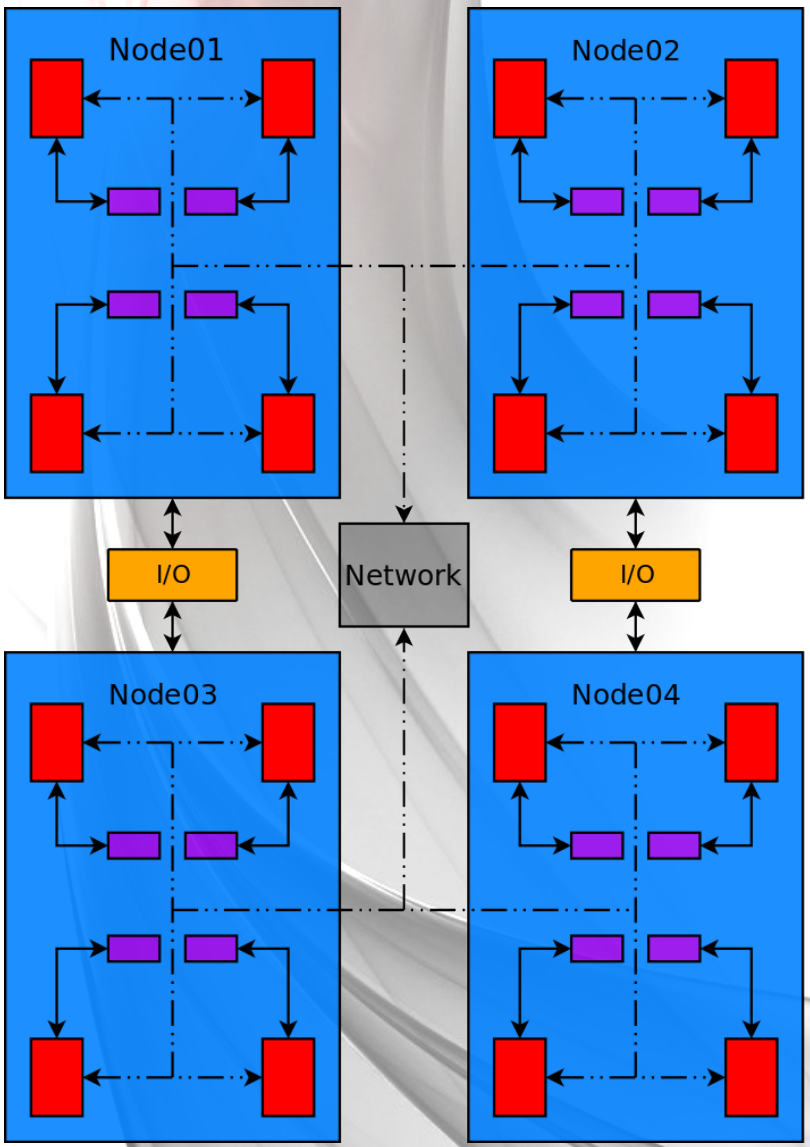
Julian Merten

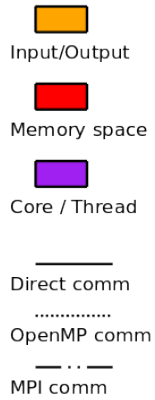
GPU Computing and Alternative Architecture

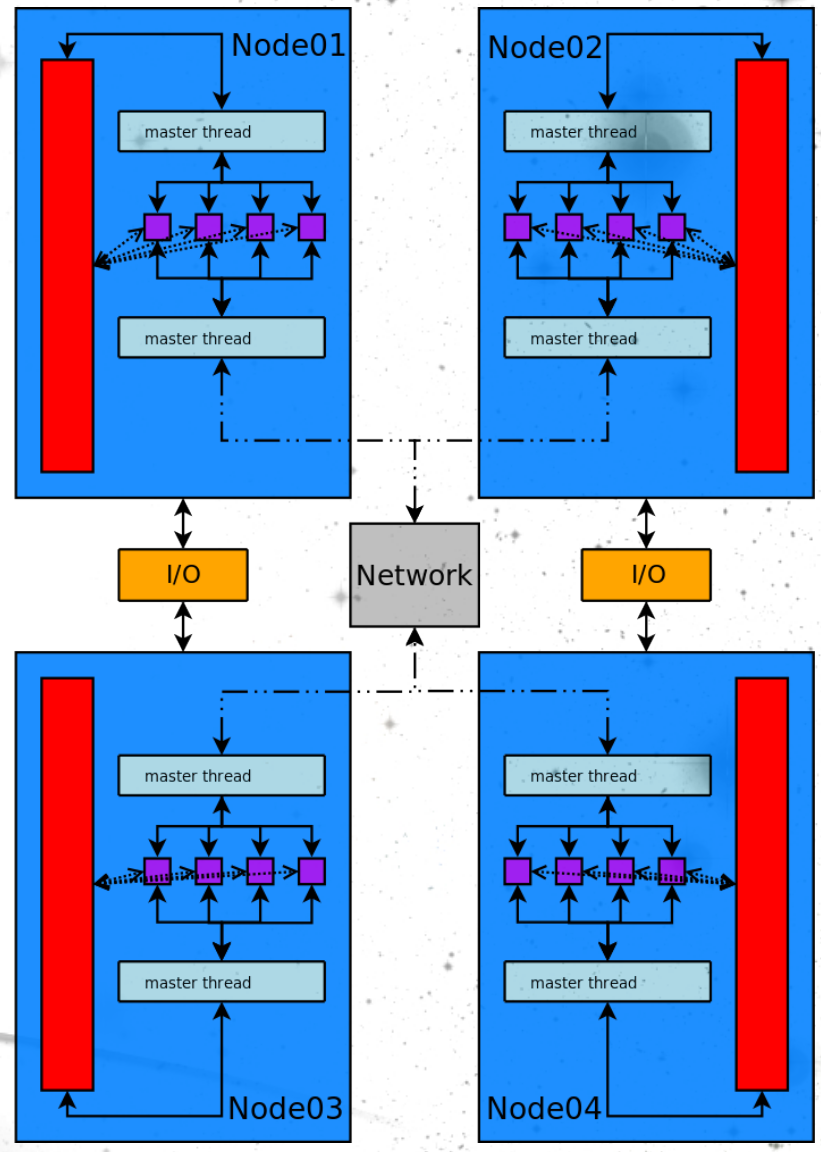
Institut für Theoretische Astrophysik
Zentrum für Astronomie
Universität Heidelberg

November 9th, 2010

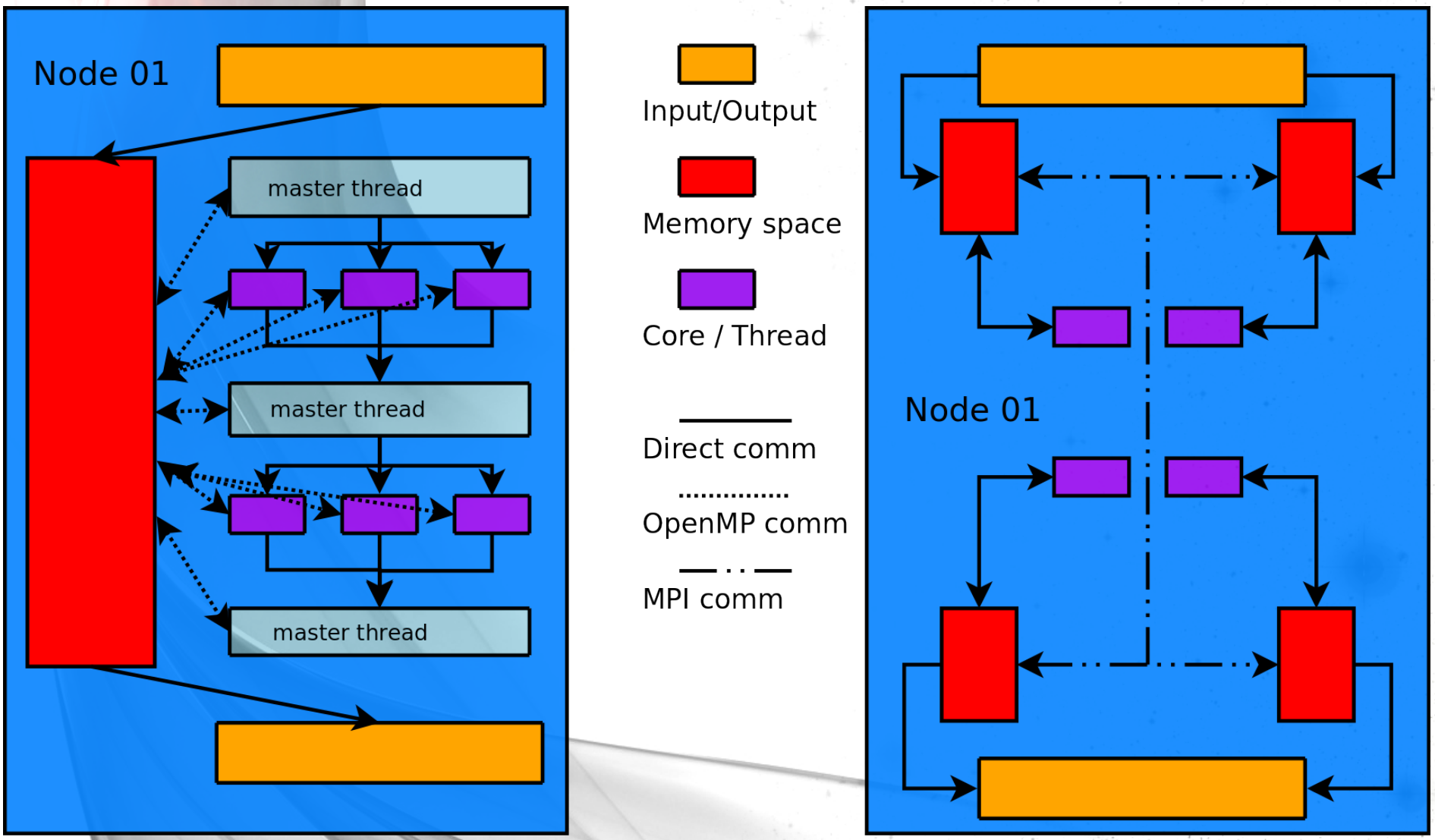
Modern HPC: Parallelism







Modern HPC: Parallelism



HPC extreme: Jaguar at Oak Ridge NL

Jaguar

- 18688 nodes
- AMD Opteron2435 @2.6 GHz
- 224256 cores
- 300 TB main memory
- optical Infiniband

But: The ExaScale!

- Current systems reach $\sim \mathcal{O}(\text{Pflop})$ @ $\mathcal{O}(10\text{MW})$
- Standard CPU design it at the edge of doable Flops/Watt
- No problem: Add more nodes!
- "Hehe...no": $\rightarrow \mathcal{O}(\text{Eflop}) \Rightarrow \mathcal{O}(10\text{GW})$
- Better: Use the same amount of nodes but make them more powerful
- $\Rightarrow$ Hybrid model with accelerated nodes.
- Unfortunately this comes not for free: Additional layer of optimised implementation on each node.

But: The EχaScale!

- Current systems reach $\sim \mathcal{O}$(Pflop) @ $\mathcal{O}$(10MW)
- Standard CPU design it at the edge of doable Flops/Watt
- No problem: Add more nodes!
- "Hehe...no": $\rightarrow \mathcal{O}$(Eflop) $\Rightarrow \mathcal{O}$(10GW)
- Better: Use the same amount of nodes but make them more powerful
- $\Rightarrow$ Hybrid model with accelerated nodes.
- Unfortunately this comes not for free: Additional layer of optimised implementation on each node.

But: The EχaScale!

- Current systems reach $\sim \mathcal{O}$(Pflop) @ $\mathcal{O}$(10MW)
- Standard CPU design it at the edge of doable Flops/Watt
- No problem: Add more nodes!
- "Hehe...no": $\rightarrow \mathcal{O}$(Eflop) $\Rightarrow \mathcal{O}$(10GW)
- Better: Use the same amount of nodes but make them more powerful
- $\Rightarrow$ Hybrid model with accelerated nodes.
- Unfortunately this comes not for free: Additional layer of optimised implementation on each node.

But: The ExaScale!

- Current systems reach $\sim \mathcal{O}(\text{Pflop})$ @ $\mathcal{O}(10\text{MW})$
- Standard CPU design it at the edge of doable Flops/Watt
- No problem: Add more nodes!
- "Hehe...no": $\rightarrow \mathcal{O}(\text{Eflop}) \Rightarrow \mathcal{O}(10\text{GW})$
- Better: Use the same amount of nodes but make them more powerful
- $\Rightarrow$ Hybrid model with accelerated nodes.
- Unfortunately this comes not for free: Additional layer of optimised implementation on each node.

But: The ExaScale!

- Current systems reach $\sim \mathcal{O}(\text{Pflop})$ @ $\mathcal{O}(10\text{MW})$
- Standard CPU design it at the edge of doable Flops/Watt
- No problem: Add more nodes!
- "Hehe...no": $\rightarrow \mathcal{O}(\text{Eflop}) \Rightarrow \mathcal{O}(10\text{GW})$
- Better: Use the same amount of nodes but make them more powerful
- $\Rightarrow$ Hybrid model with accelerated nodes.
- Unfortunately this comes not for free: Additional layer of optimised implementation on each node.

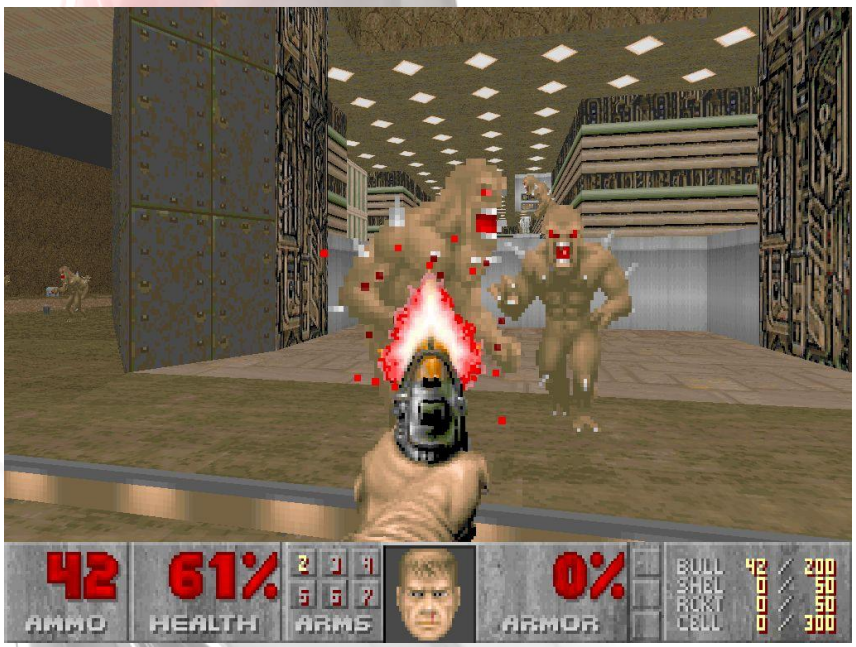
But: The ExaScale!

- Current systems reach $\sim \mathcal{O}$(Pflop) @ $\mathcal{O}$(10MW)
- Standard CPU design it at the edge of doable Flops/Watt
- No problem: Add more nodes!
- "Hehe...no": $\rightarrow \mathcal{O}$(Eflop) $\Rightarrow \mathcal{O}$(10GW)
- Better: Use the same amount of nodes but make them more powerful
- $\Rightarrow$ Hybrid model with accelerated nodes.
- Unfortunately this comes not for free: Additional layer of optimised implementation on each node.

But: The ExaScale!

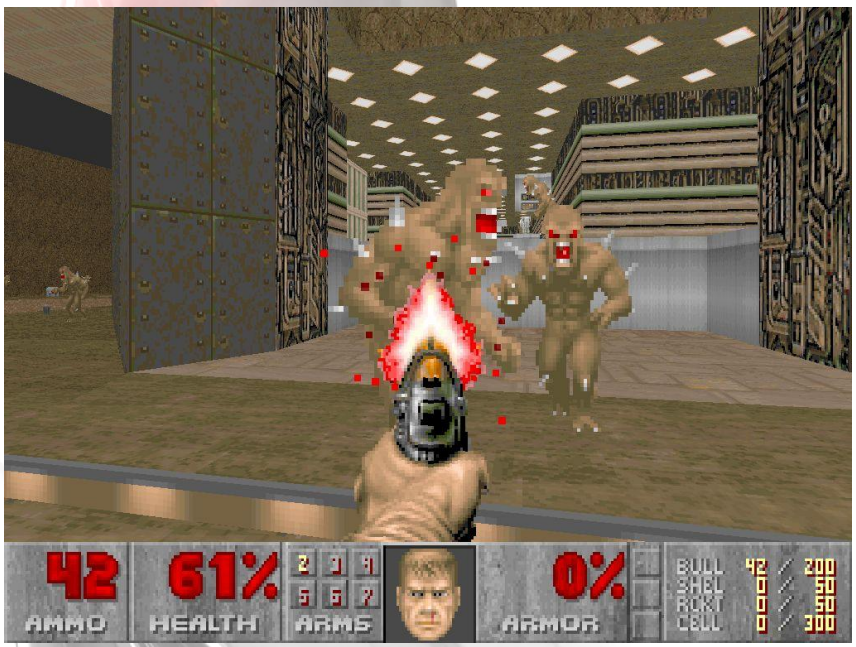
- Current systems reach $\sim \mathcal{O}(\text{Pflop})$ @ $\mathcal{O}(10\text{MW})$
- Standard CPU design it at the edge of doable Flops/Watt
- No problem: Add more nodes!
- "Hehe...no": $\rightarrow \mathcal{O}(\text{Eflop}) \Rightarrow \mathcal{O}(10\text{GW})$
- Better: Use the same amount of nodes but make them more powerful
- $\Rightarrow$ Hybrid model with accelerated nodes.
- Unfortunately this comes not for free: Additional layer of optimised implementation on each node.

The advent of GPU's...or the art of shooting monsters



Carmack et al. 1993

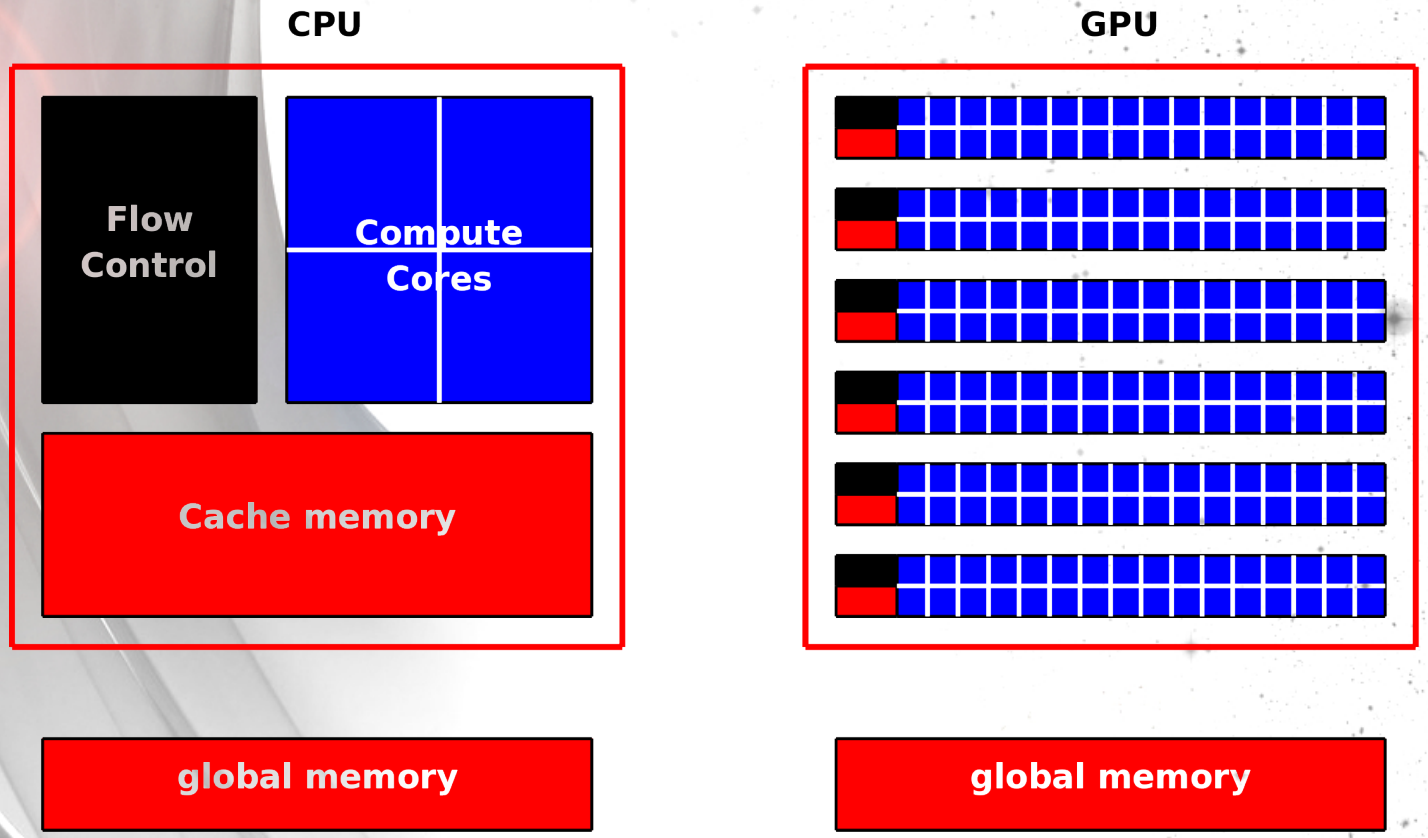
The advent of GPU's...or the art of shooting monsters



Carmack et al. 1993

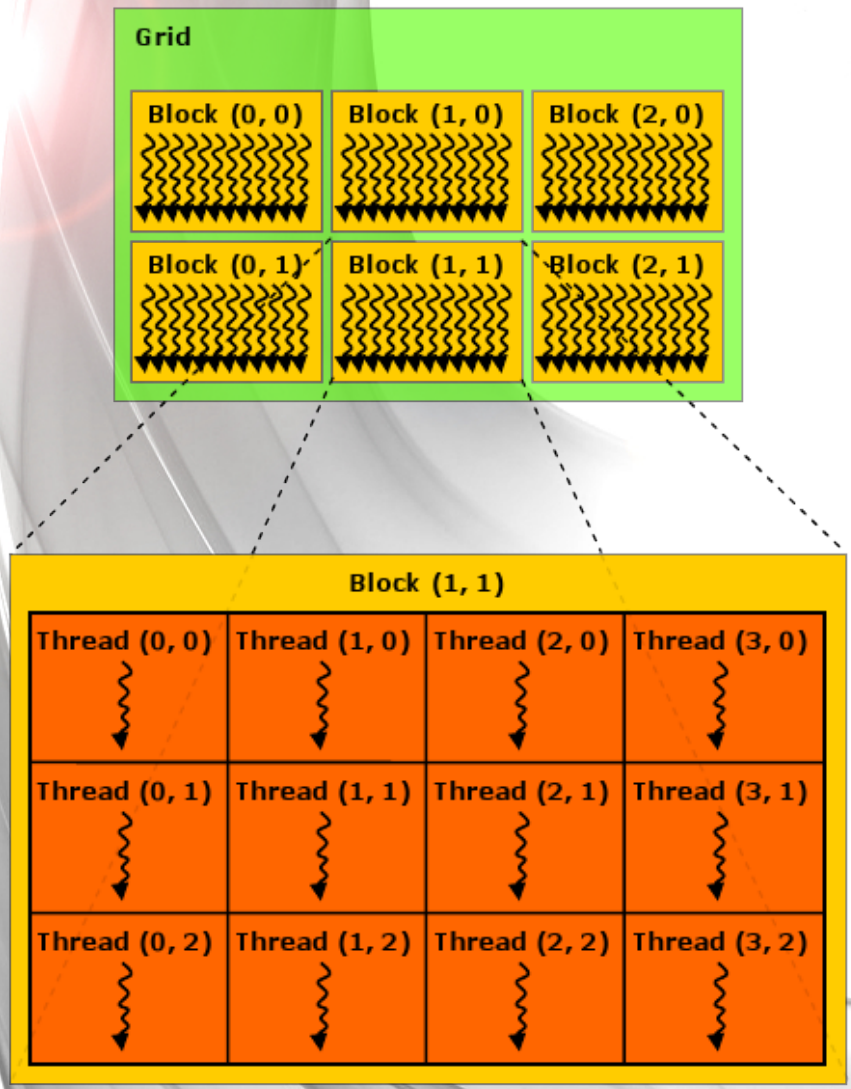
Carmack et al. 2004

A massively parallel chip design



- You have to keep those cores busy.
- Extreme parallelisation by SIMT concept, similar to the vector concept.
  $\Rightarrow$ Split a problem into thousands/millions of identical threads.

Threads must be simple and well-ordered

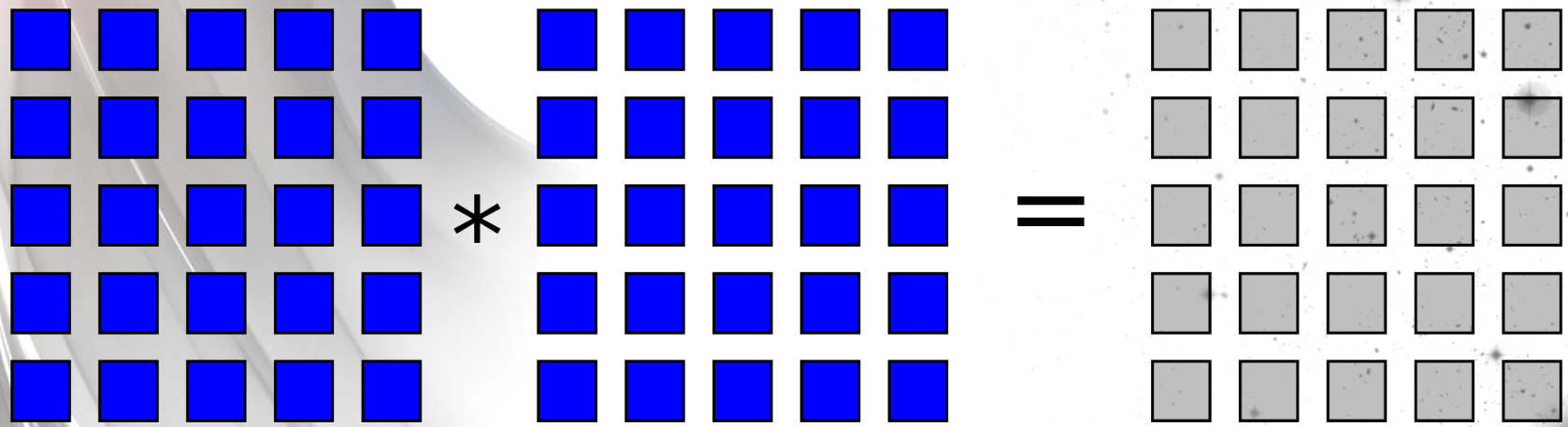


From NVIDIA CUDA C Programming Guide 3.1.1

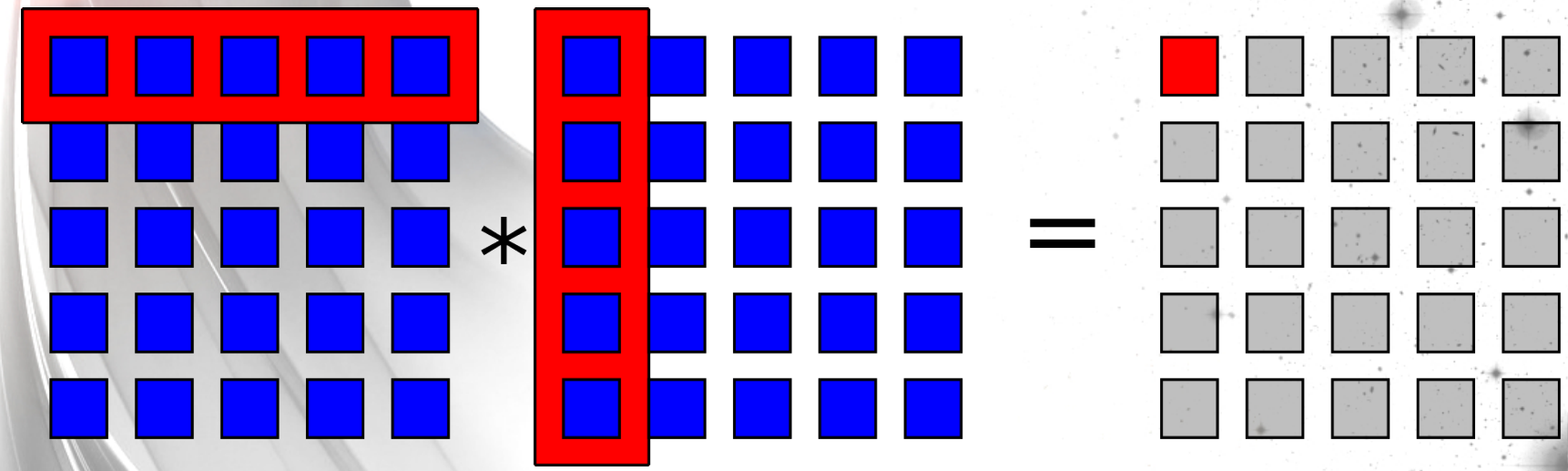
Rules

- Blocks run on MP's
- Keep cores on MP busy with threads
- Sync only possible within block
- Use memory hierarchy (register, shared/L1, global)
- Threads must be "close" to the data
- "Do not disturb the collective."
- Find balance between parallelisation and work per thread

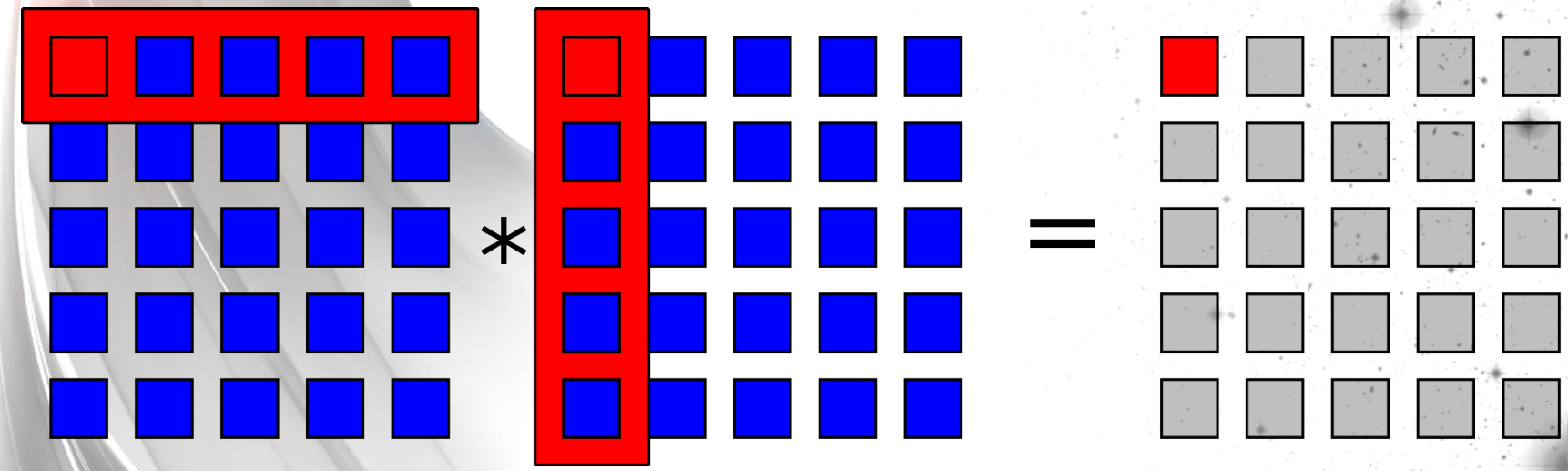
Finally an example



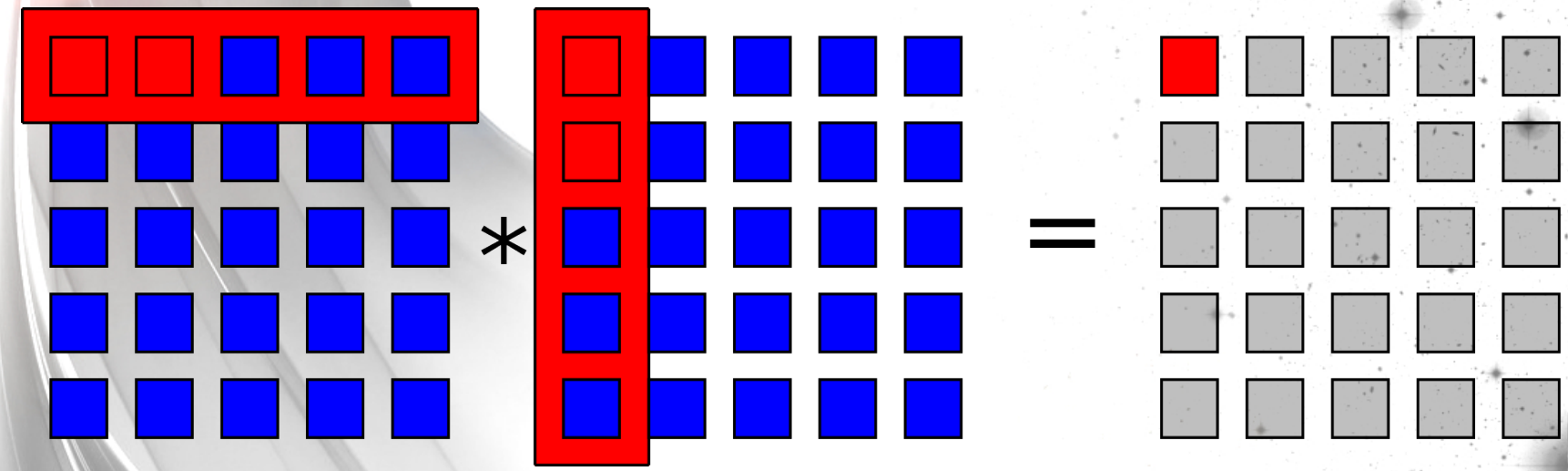
Finally an example



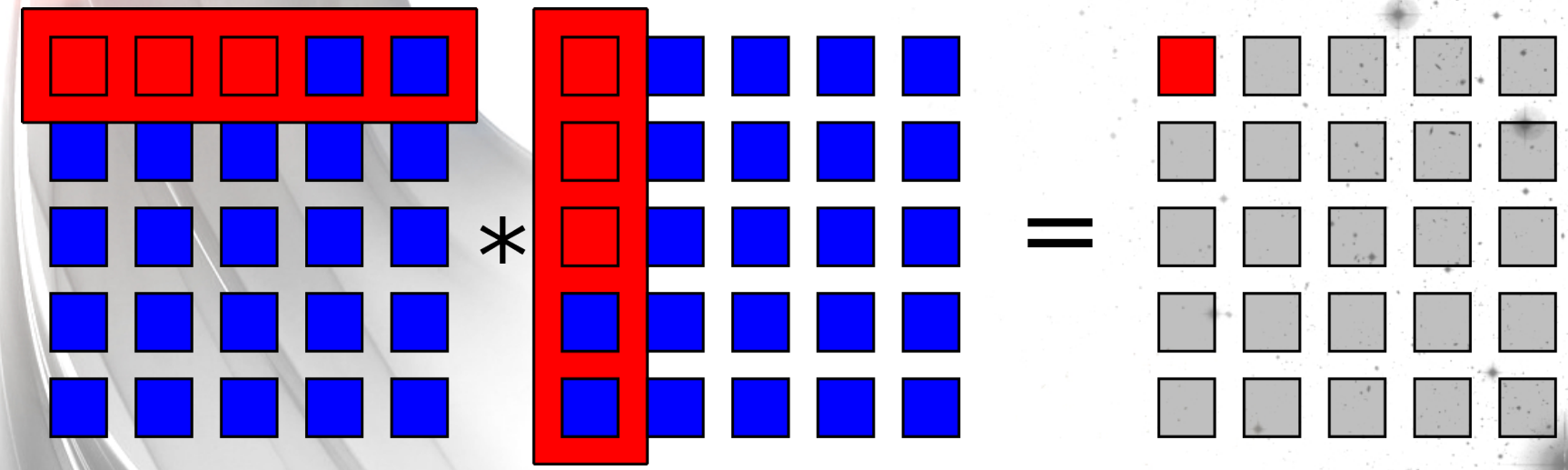
Finally an example



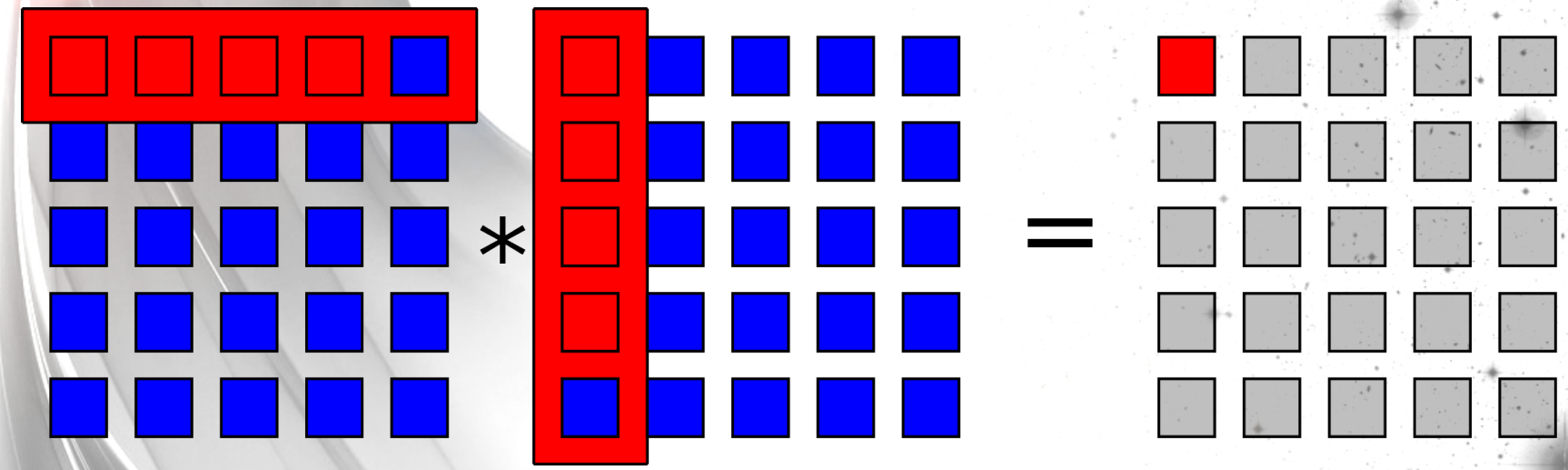
Finally an example



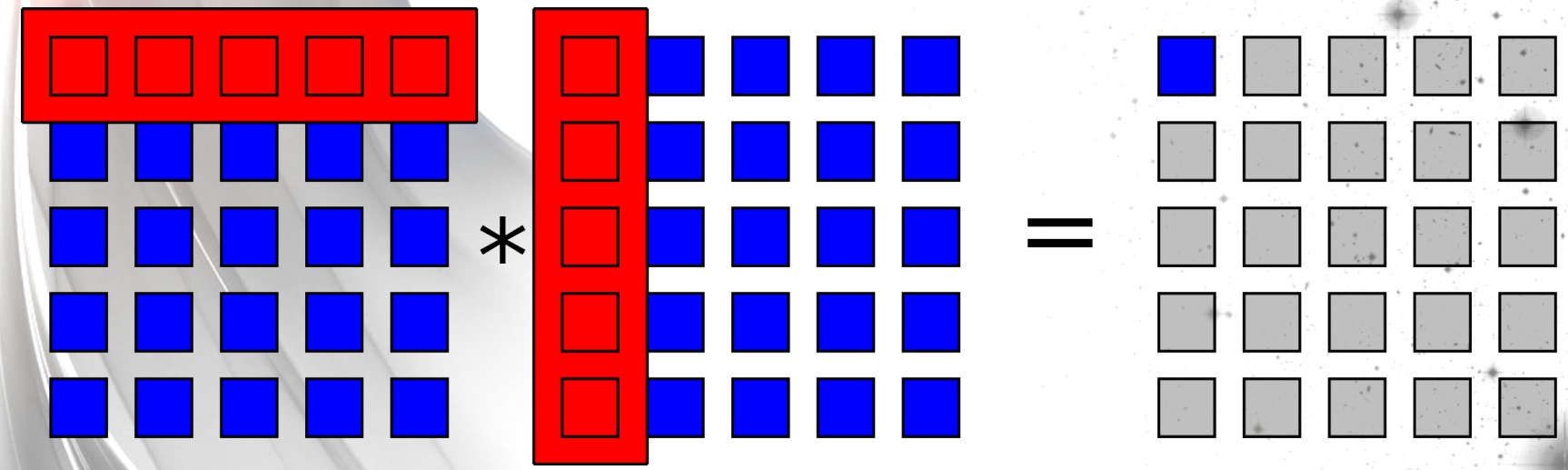
Finally an example



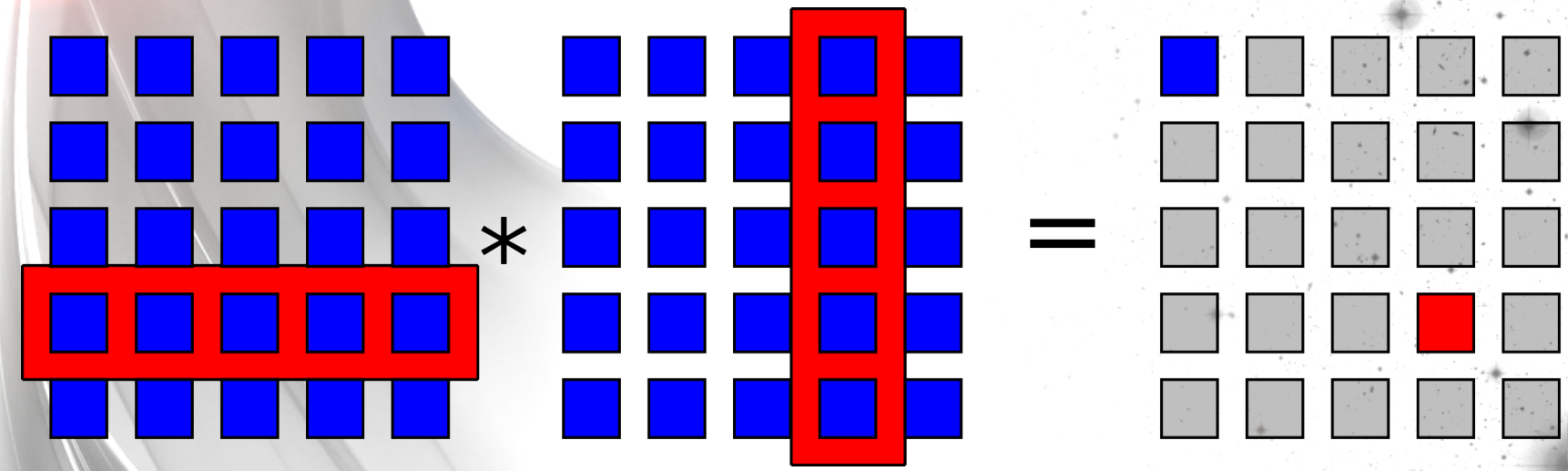
Finally an example



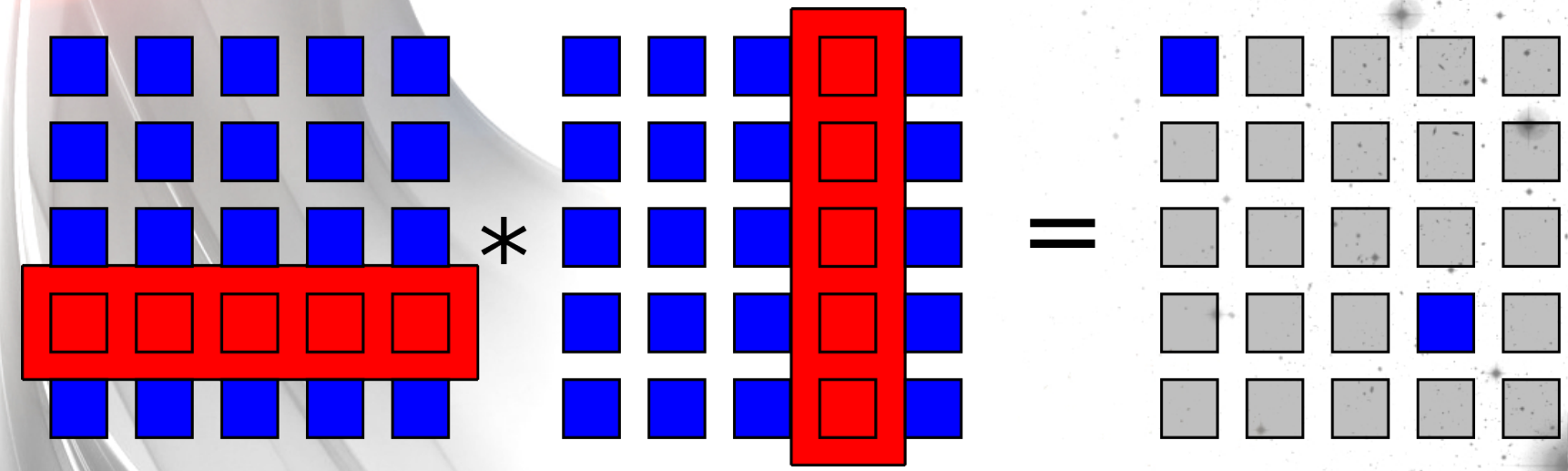
Finally an example



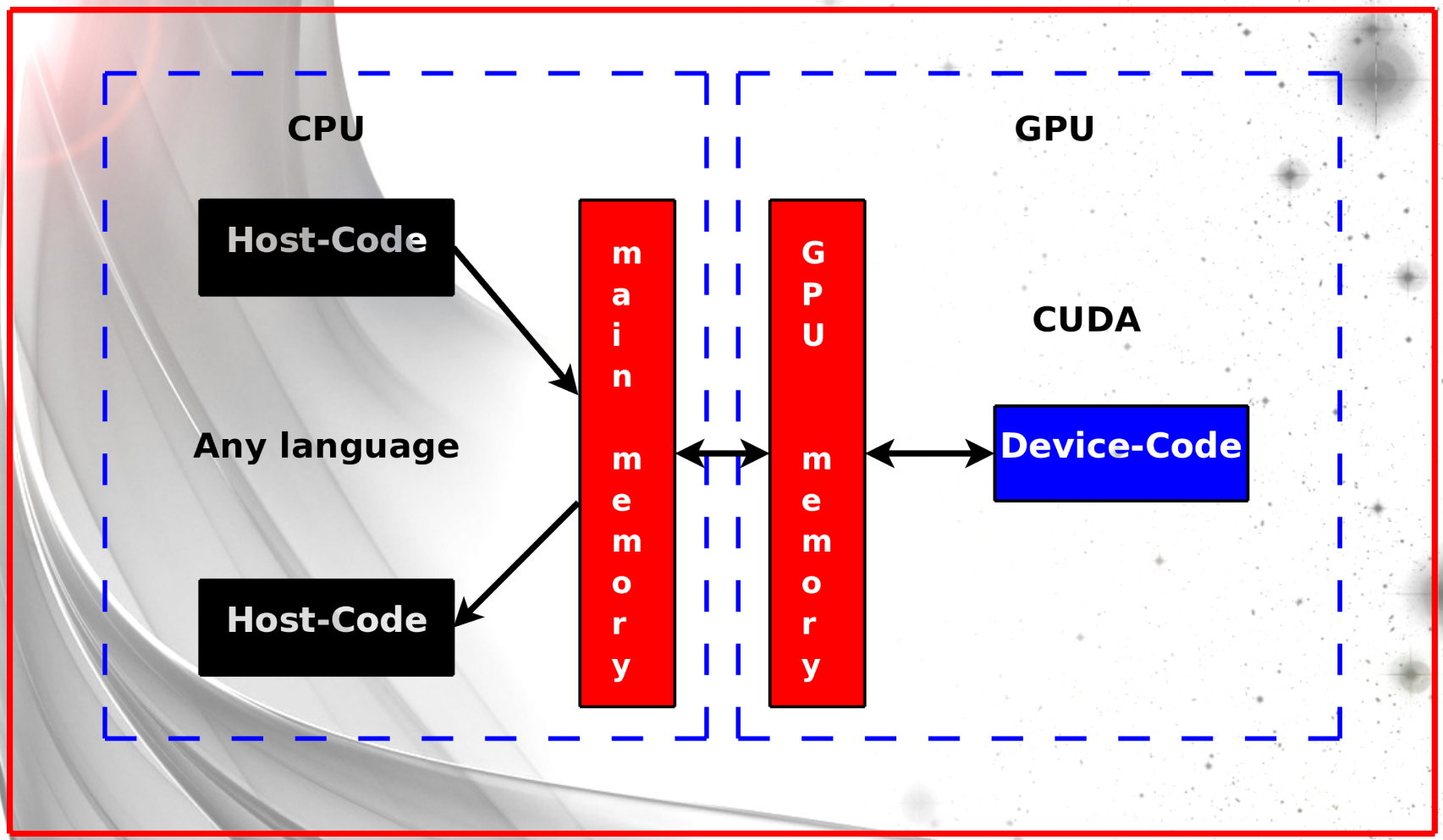
Finally an example



Finally an example



Finally an example



Tesla C870

- $\sim$ 500 Mflops peak in single
- NO double
- 1.5 GB global memory
- 16 MPs
- 128 cores
- 8 KB register size
- 16KB shared memory

Tesla C1060

- $\sim$ 1 Gflop peak in single
- `A lot` less in double
- 4 GB global memory
- 30 MPs
- 240 cores
- 16 KB register size
- 16KB shared memory

Tesla C/M2050 (Fermi)

- $\sim$ 1 Gflop peak in single
- $\sim$ 500 Mflops peak in double
- 3 GB ECC global memory
- 14 (16) MPs
- 448 (512) cores
- 32 KB register size
- 48 KB shared memory

Tuning tips: "The GPU is a funny beast"

- Make sure that you have enough threads per block to keep the MP cores busy.
- Always keep the problem in GPU memory, never-ever even think about node memory access.
- Use memory coalescence and shared memory. Avoid global GPU memory within block calculations.
- Know your thread scheduler, it enables you to use the word "warp" in a scientific talk.
- Avoid thread divergence, no conditionals in the threads.
- Exploit the parameter space (which is large in GPU programming).
- Try to stay calm when NVIDIA changes the architecture and you have to start from scratch.

Programming models: CUDA and OpenCL

CUDA C available from NVIDIA and CUDA Fortran available from PGI (commercial).

Pros

- Full control over all features.
- Achieves best performance.
- Large set of libraries already available (CUBLAS, CUFFT, CUSPARSE, CURAND, CULA).

Cons

- Learning curve.
- Limited to NVIDIA architecture.
- Separate compiler.

My thoughts on OpenCL: Basically identical to CUDA, open, but harder to learn (compare CUDA Driver API).

Programming models: Directives

Available from `PGI` and `HMPP` (commercial).

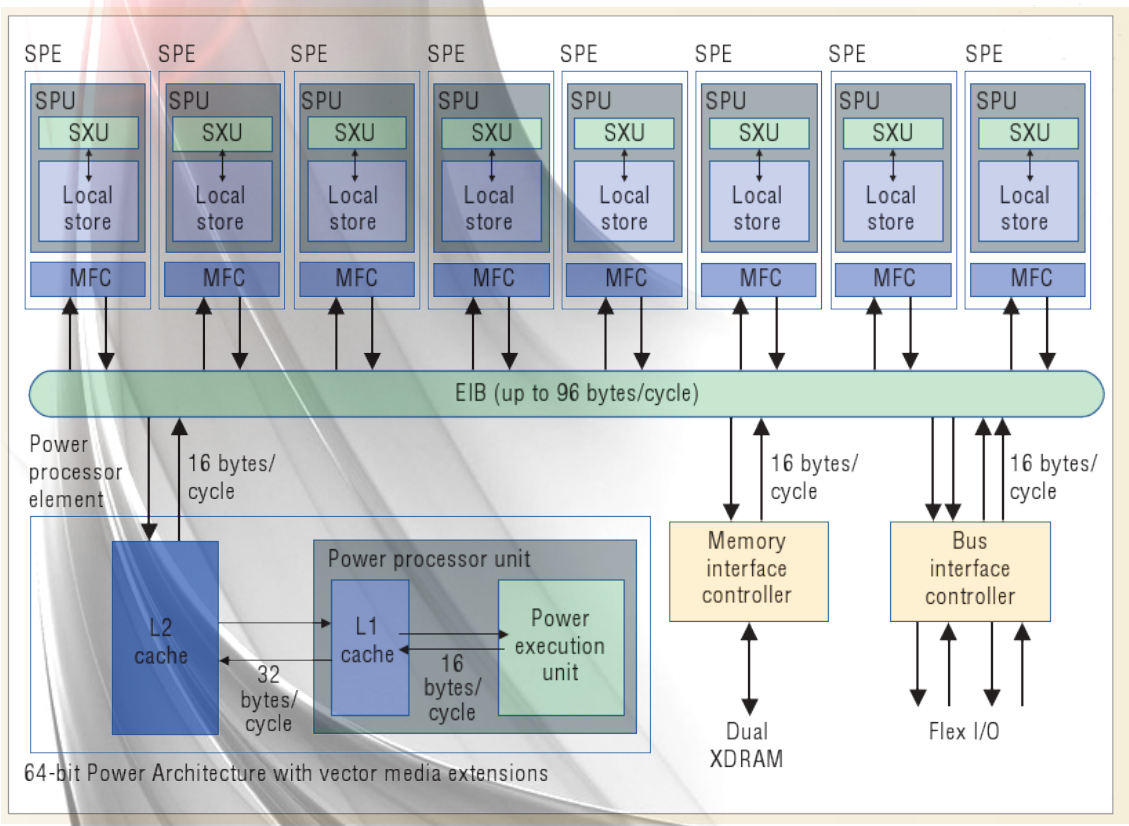
Pros

- Much easier to learn, similar to `OpenMP` code.
- Codes are much easy to port.
- Might be the model for the future.

Cons

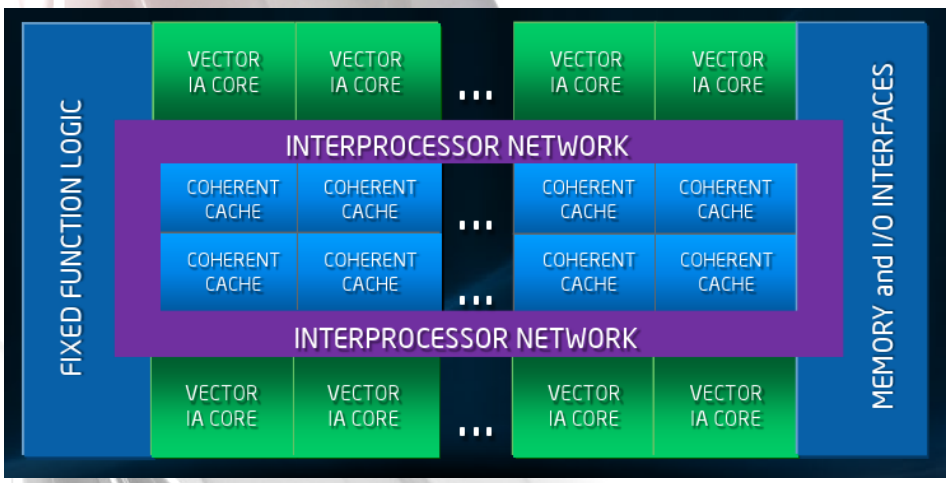
- Slower than `CUDA` and harder to tune.
- By now, much worse documented and accepted by the community.
- Not ready for large production runs.

Alternative architectures: `IBM Cell`



- Not really my field, sorry.
- Powers the `PlayStation3`.
- Eight core design.
- Peaked with `Roadrunner`.
- Development and programming models available.
- Not wide-spread.

Alternative architectures: Intel Knight's Ferry



Courtesy of Intel Corp.

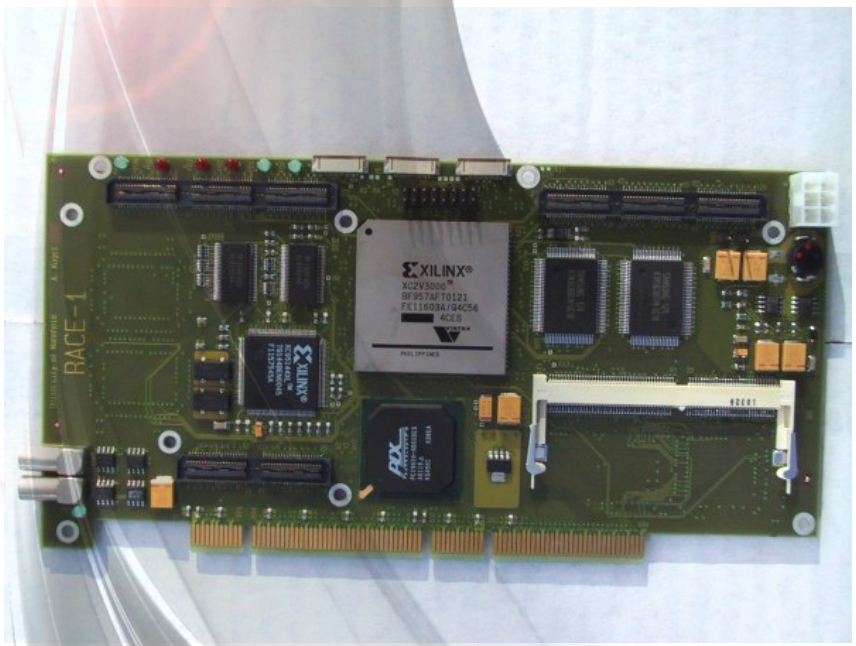
- 32 x86 cores @ 1.2 GHz
- 4 threads per core
- PCI-E interface to host
- C(++), Fortran compilers
- Better behaved regarding cache coherence and core synchronisation
- Might become the main competitor to the Fermi architecture
- Designs with >50 cores are planned, Knight's Corner

Alternative architectures: `Intel Knight's Ferry`

Courtesy of `Intel` Corp.

- 32 x86 cores @ 1.2 GHz
- 4 threads per core
- PCI-E interface to host
- C(++), Fortran compilers
- Better behaved regarding cache coherence and core synchronisation
- Might become the main competitor to the Fermi architecture
- Designs with >50 cores are planned, `Knight's Corner`

Alternative architectures: FPGA's



MPRACE board
using a `Xilinx` FPGA chip,
as e.g. used in Titan @ ARI Heidelberg

- Calculations are basically hard-wired on-chip
- Extremely high performance on extremely specialised hardware
- LHC uses those devices for triggering and event-selection
- Disadvantages obvious: You need to be a specialist, to set-up, run and maintain those systems
- Only few systems are available

Conclusions

1. Extreme parallelisation on the node achieves tremendous speed-ups on special hardware.
2. `Fermi` GPUs seem to be the preferred accelerator model at the moment. Curious for `Knight's Ferry (Corner)` .
3. `CUDA` is probably the best way to programme them.
4. Already small/medium size systems are very powerful (e.g. 40-96 node systems in Heidelberg). Accelerated single-node systems might be even more useful (analyses etc.).
5. Nevertheless, porting and designing codes is not straight-forward and involves a lot of specific knowledge. A GPU version of e.g. `Gadget` will look very different and yes, you can trash your career with this.
6. Many core programming is not completely mature yet, software but esp. hardware is still developing. Anyway, esp. cosmology is about to miss the HPC train.
7. Whoever is interested in a collaboration on a GPU project: Let me know.