Moving Scientific Codes to Multicore Microprocessor CPUs

The case of the acceleration of Piecewise-Parabolic Method Algorithm on the IBM Cell Multicore, and some recipes for the future of supercomputing

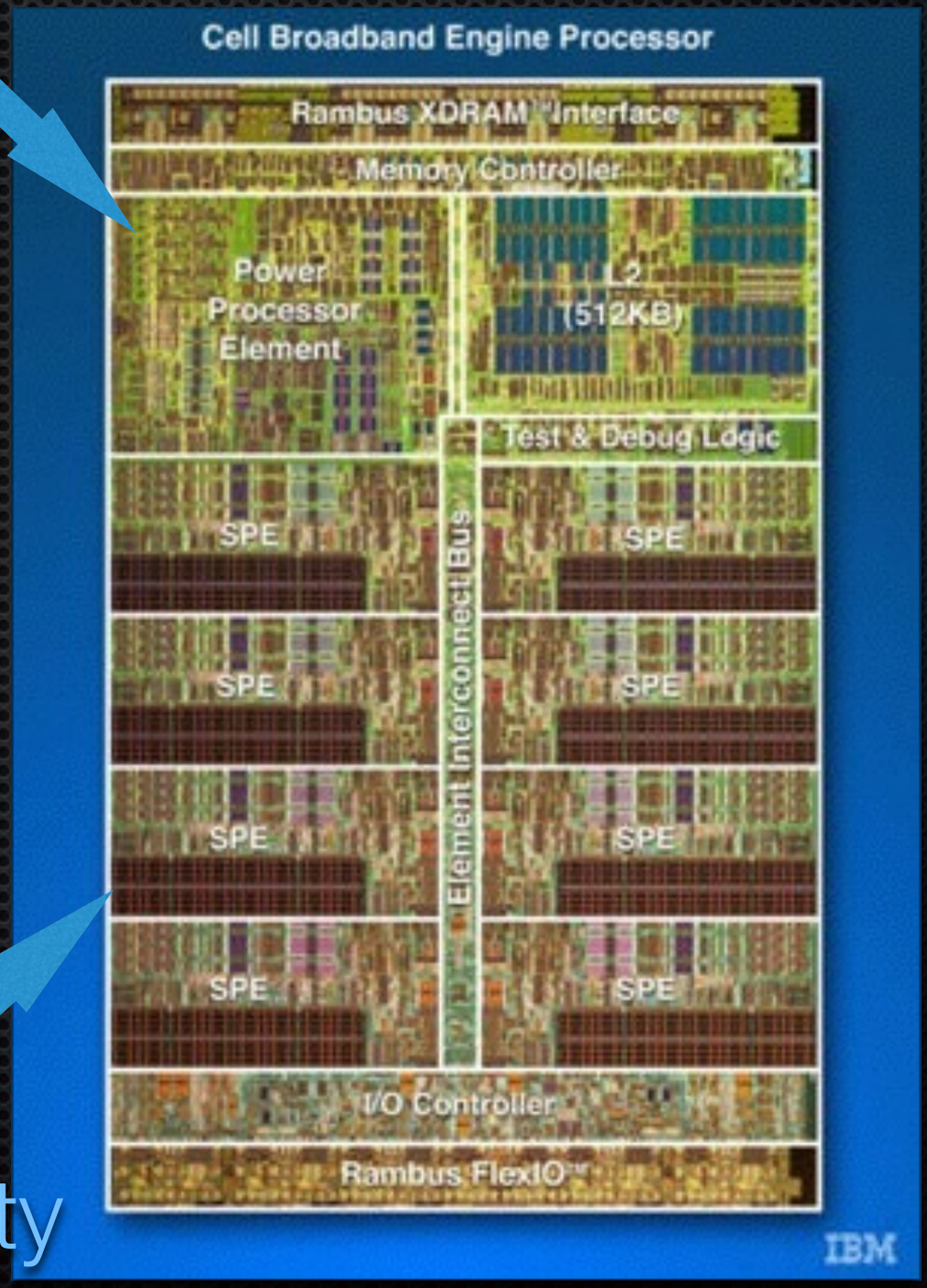
Riccardo Cattaneo

The case study

Why?

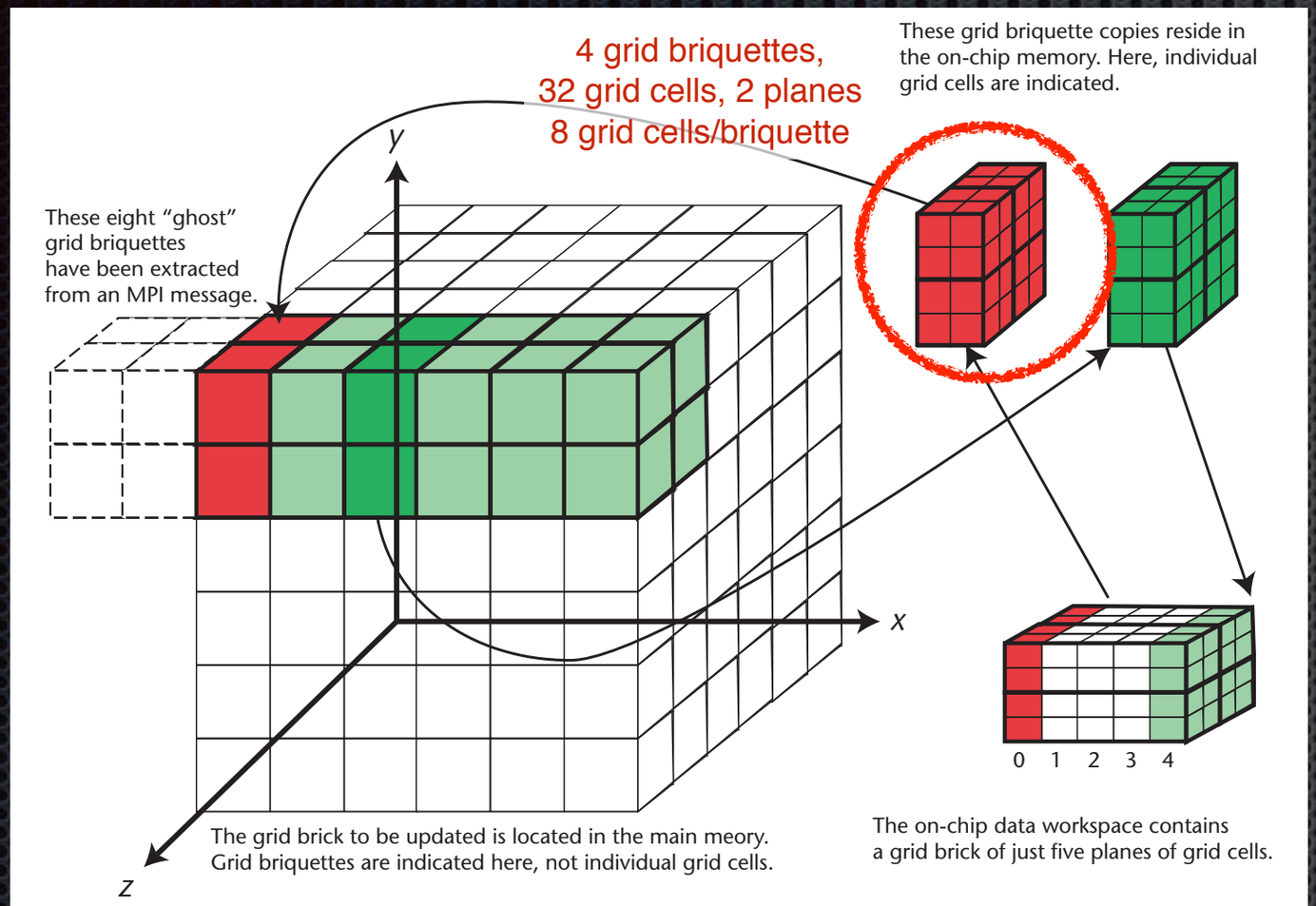
Power PE, mid complexity
32KB L1, 512 KB L2 (LL) cache

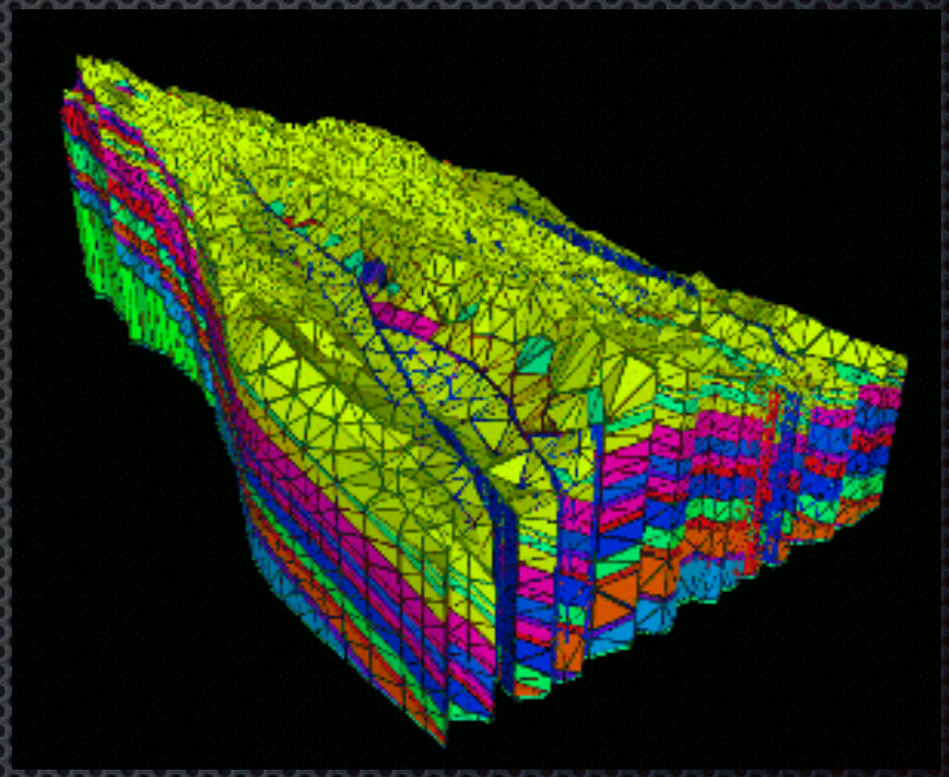
- **Case study** for the parallelization of numerical simulation on a novel (at that time) multicore architecture
- **Demonstrate novelties** (at the time) of Cell architecture, by themselves

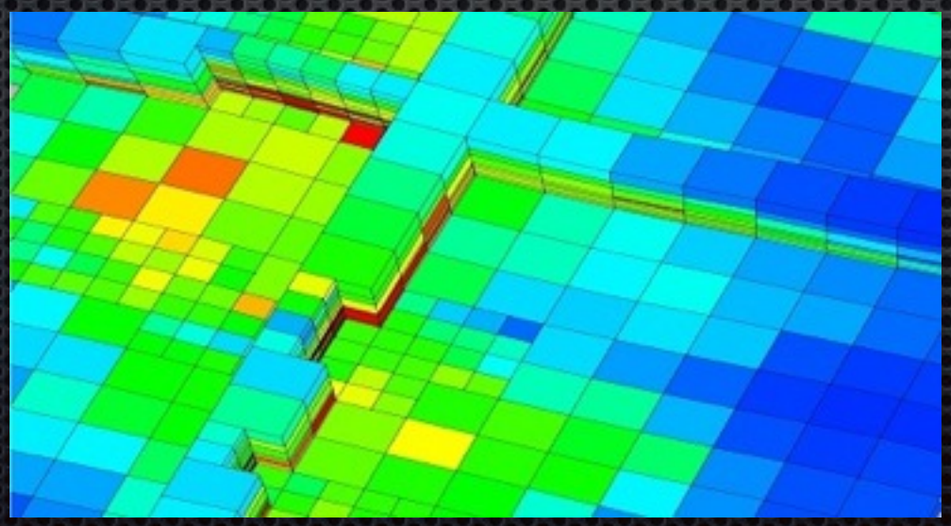


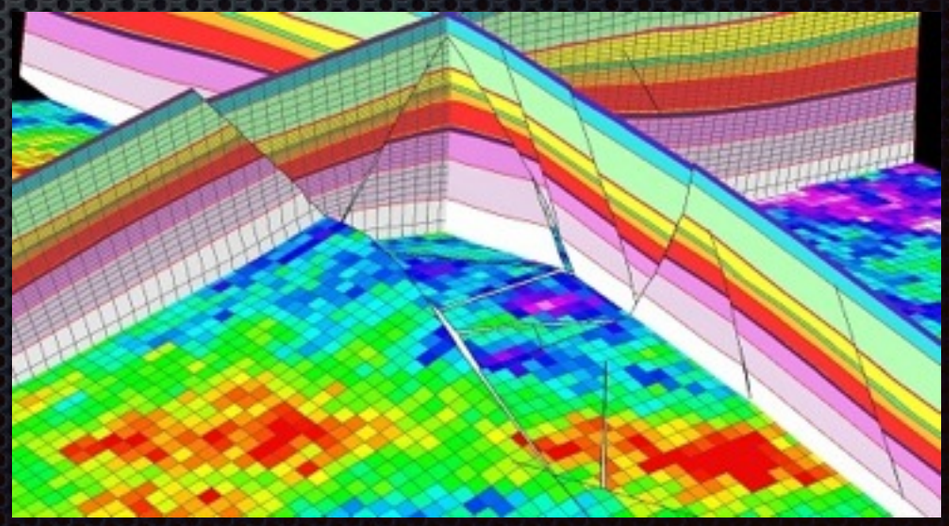
8 Synergistic PE, low complexity
256 KB private, local store

Algorithm: data layout, mapping with physical world









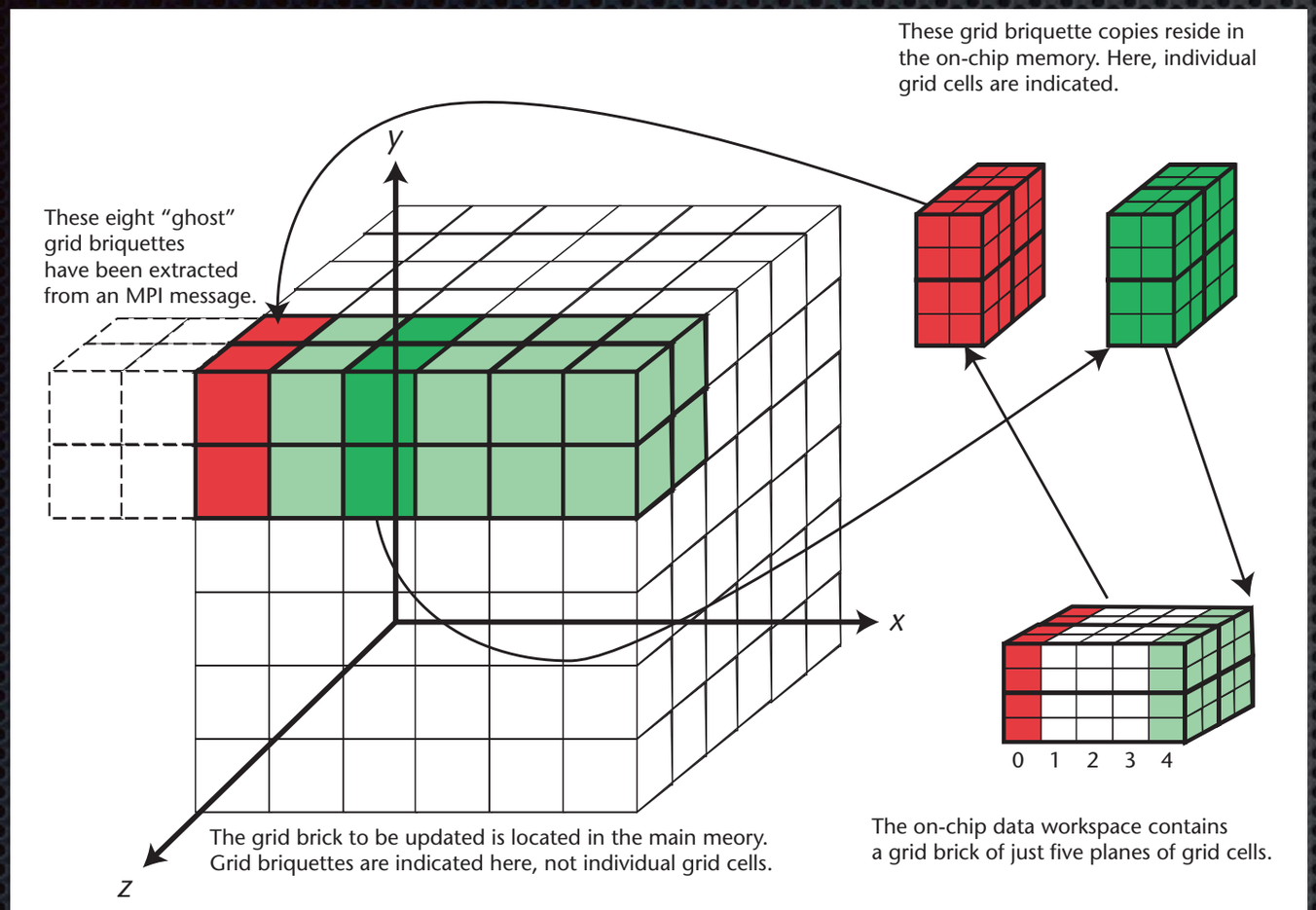
Some challenges

- "[…] The 256-Kbyte local store of the Cell processor SPE core must accommodate **both the program and the data workspace** needed for the computation […]"
- "[…] The four-way SIMD processing that's the only 32-bit floating-point mode offered by the Cell processor SPE **demands that we provide a certain level of computation uniformity** if we want the SPE **to perform**. […]"
- "[…] **Data** can be **streamed** into the local store **using** the **DMA** capability; the program then operates on this data using the local store as a data workspace, and the results are streamed back to main memory using the DMA again […]"
- "[…] [to perform] The idea is to keep the **floating-point units as busy** as possible, which means that they should essentially **never be waiting for data** to arrive from main memory into the local store. The concept of "as busy as possible" is algorithm and CPU core dependent. […]

Both architectural and software design challenges

Specific challenges

- To perform, one must know in advance the "geometry" of data in memory, and the architecture



- As always, local store (LS)/private cache/memory hierarchy plays a fundamental role
- For example: a small instance (one whose code and working sets fits in LS, and thus maximizes *peak* flops) performs roughly 7.7 Gflops per second per core (Gflop/s/core), which represents only approximately 30 percent of the peak performance

- What about (peak) bandwidth requirements?
  - The computational intensity is 62 flops per word (flops/ word)
  - At 30 percent of peak perfor- mance, we execute 2.4 flops per clock cycle,
  - so we have 26 or 16 clock cycles in which to transfer the needed word.
  - The eight SPU cores on the Cell processor share an ability to stream four words per clock, each can stream only half a word per clock cycle. We need, in principle, only two clock cycles to make our data exchanges with main memory,; ok because we have 26 or 16 available to us.
- So? It is possible to sustain peak performance of this algorithm on Cell
  - however, this is a platform specific computation for a specific algorithm… **difficult to generalize!**

Specific challenges

- Moreover:
  - Code size is non neglibile, as it is generated by aggressive loop unrolling, SIMD and loop vectorization; however, data and code must fit in the 256KB of private memory.
    - not only a data-size/throughput problem, but also a code generation one
    - in a slight variant this algorithm, using a dual fluid scheme, code and data cannot fit into the memory (5K lacking)
  - No cache-level sharing among cores imply that all data transfers meant for sharing are realized via DMA from cores to DRAM
    - explicit (or compiler + directives-generated) DMA transfers

“[…] Our tools will assume that the target programmer is highly expert and willing to communicate to our tools (via directives) with special knowledge about the program […]”

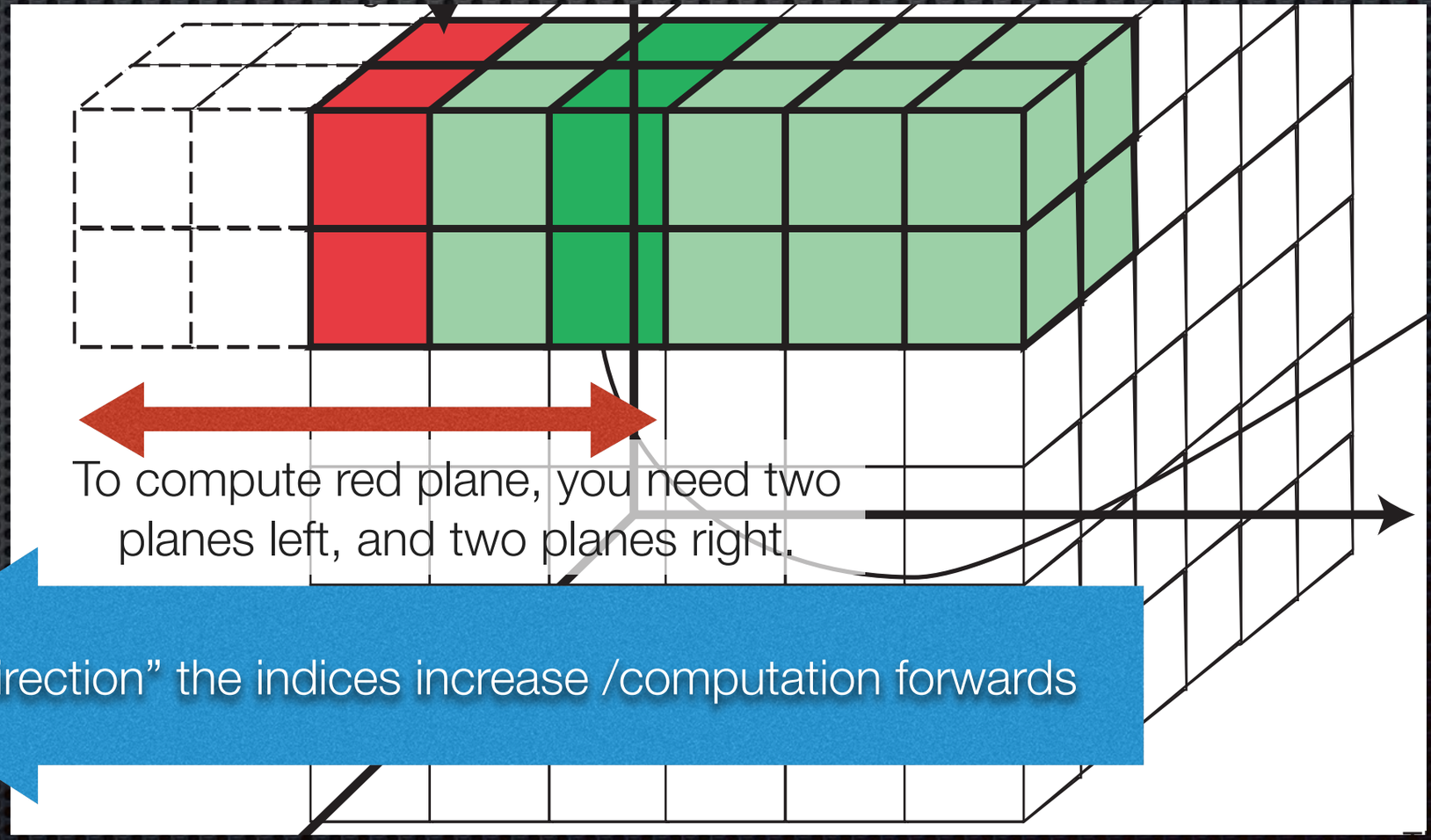
Algorithm and compiler challenges

- "[…] compilation of our codes in the standard fashion reveals that the performance benefits we've obtained **can't be achieved through simple compilation**. The ***code must be transformed first*** and then compiled. A possible reason for this is the extensive nature of the code transformations we use […]"
  - Code restructuring
- "[…] For single-fluid PPM, which updates six variables per grid cell, the data record corresponding to the grid briquette is 192 bytes, **which is somewhat inefficiently small**. However, for this code, we can read these records in eight at a time if we pack them appropriately. […]"
  - *Explicit* memory-related (thoughput) considerations at compilation/language-level
- "[…] We can express the numerical algorithm, assuming that all subroutines have been inlined (easily done automatically) as a series of **triple loop nests**. In each such loop nest, the computation extends over the entire grid **briquette** plus some portion of the ghost cell regions. […]"
  - A "standard" pattern for physical simulations

(MANUAL) Memory locality improvements

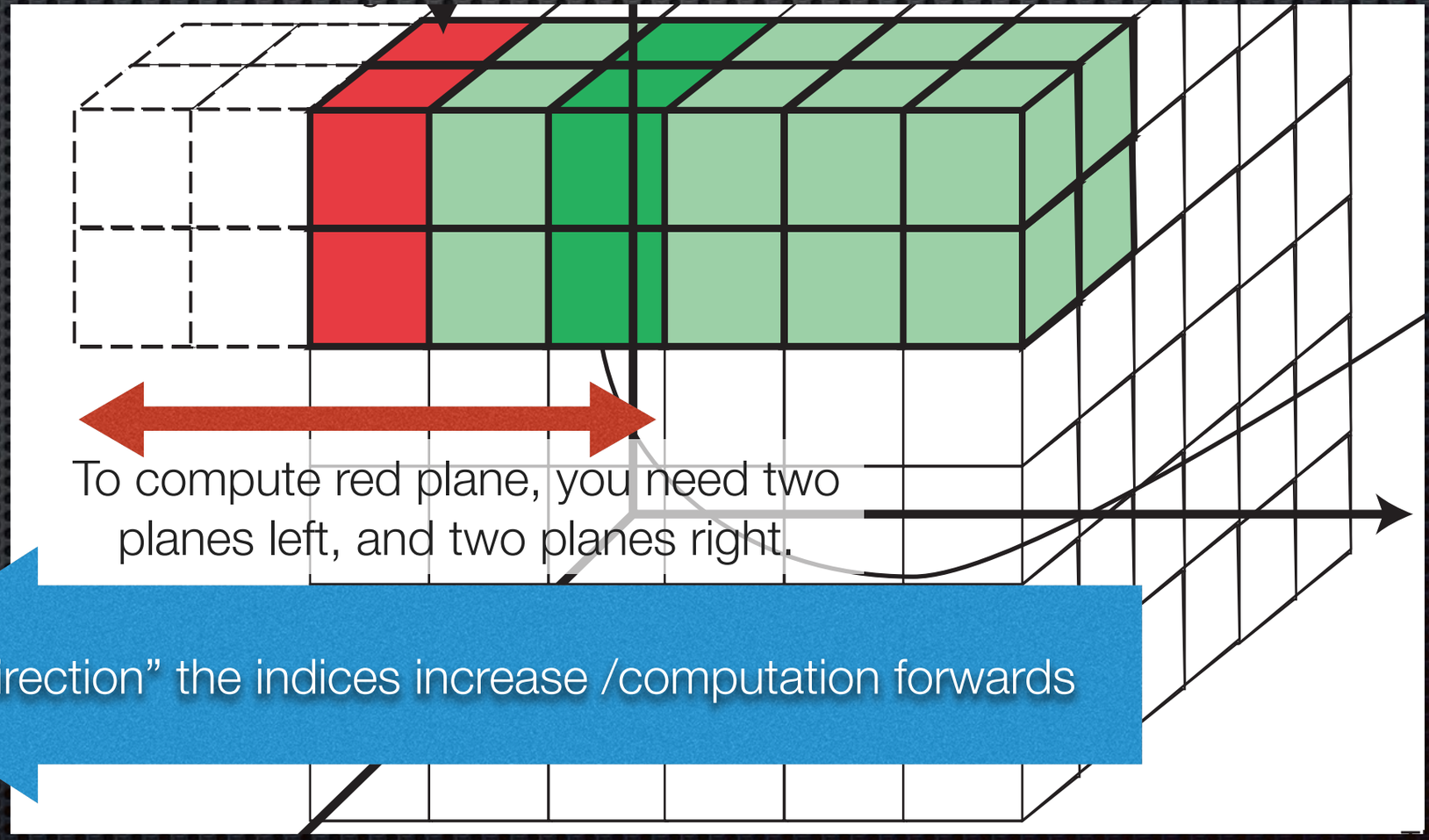
"Natural" way to express the algorithm: one *block* at once
"Efficient" way: one *plane* at once

(intuitively, I store more of the data that will be needed further down the computation in the private, local storage. This, copuled with SIMD vectorization (4 words at once, observe the briquette's face is 2x2=4 words wide) allows the *pipelined* and *efficient (as data reuse is in place)* computation of briquettes/cubes)



(MANUAL) Memory locality improvements

This implies that the code must be restructured, or a language feature must be in place to aid the compiler to layout data, generate streams and generate code.
Author's opinion is that *both* code restructuring *and* compiler techniques must be used to optimize the execution of this workload (at least, on the Cell processor)



In

Fr

[..

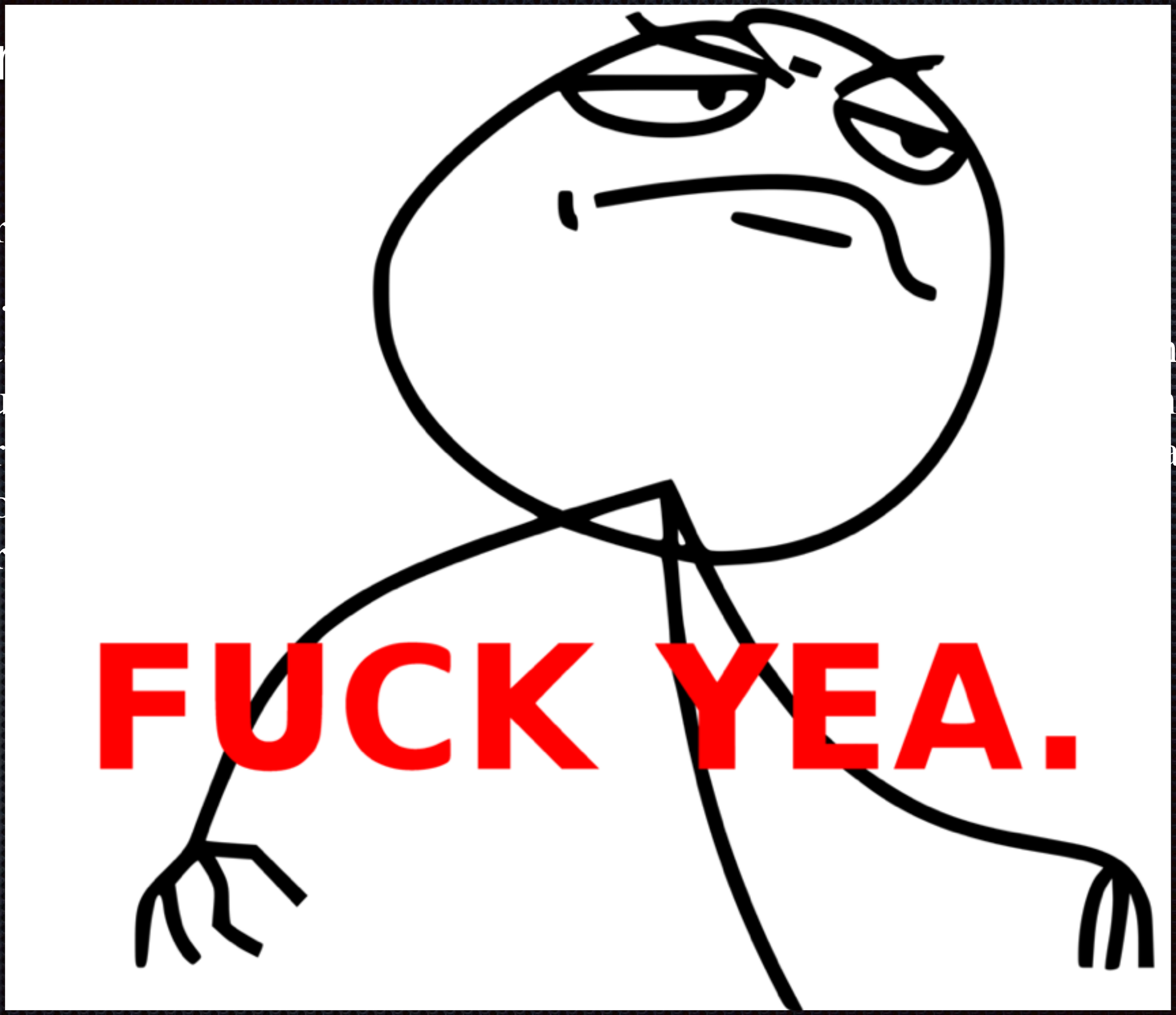
fa

ru

or

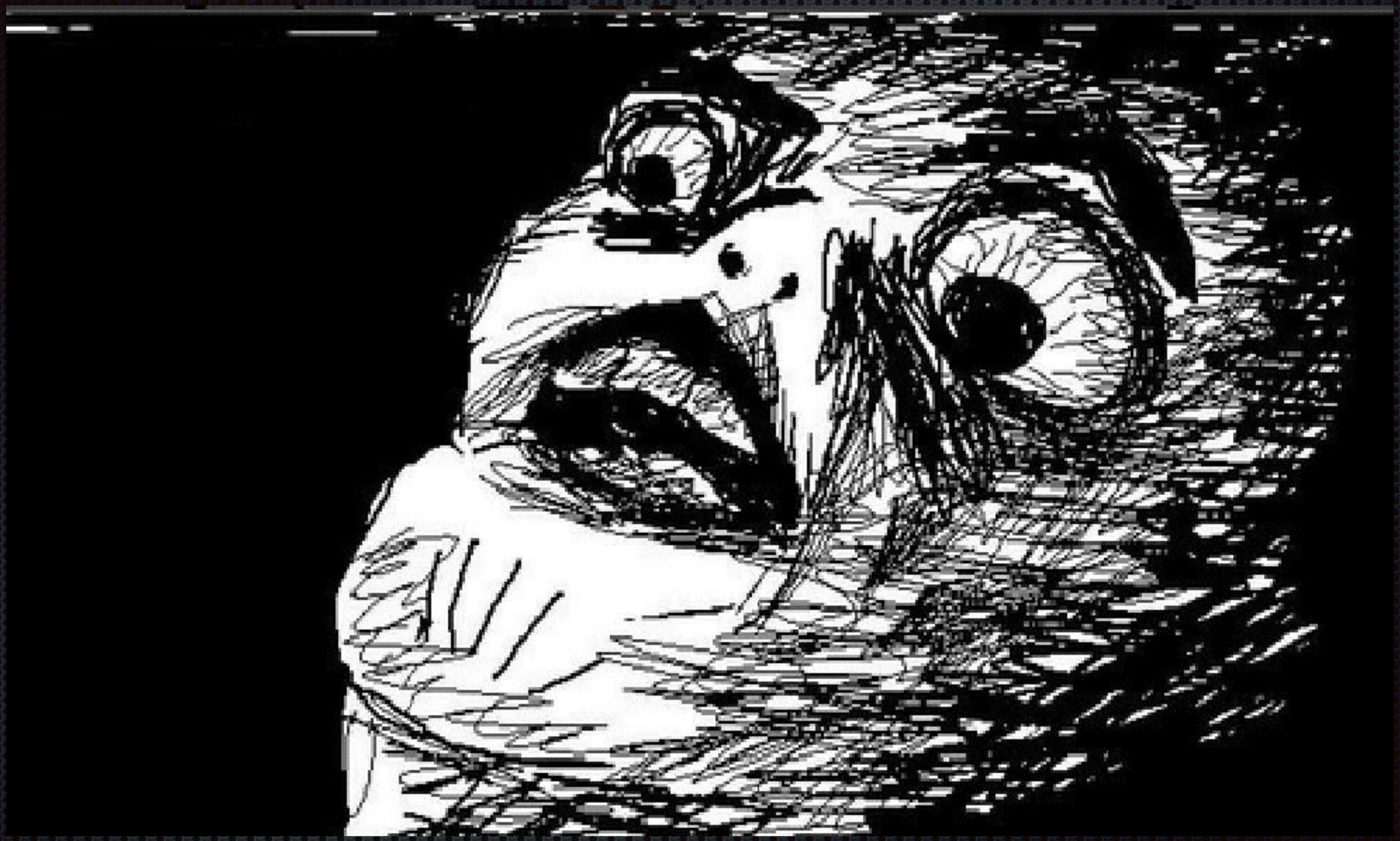
po

gr



However…

Intermediate conclusions (!)



The present, a possible future

The present, a possible future

THE design contraint of nowadays: power consumption.
Supercomputing is limited by power walls as other platforms do.
How do we further improve on efficiency?

The Green500 List

Listed below are the June 2013 The Green500's energy-efficient supercomputers ranked from 1 to 10.

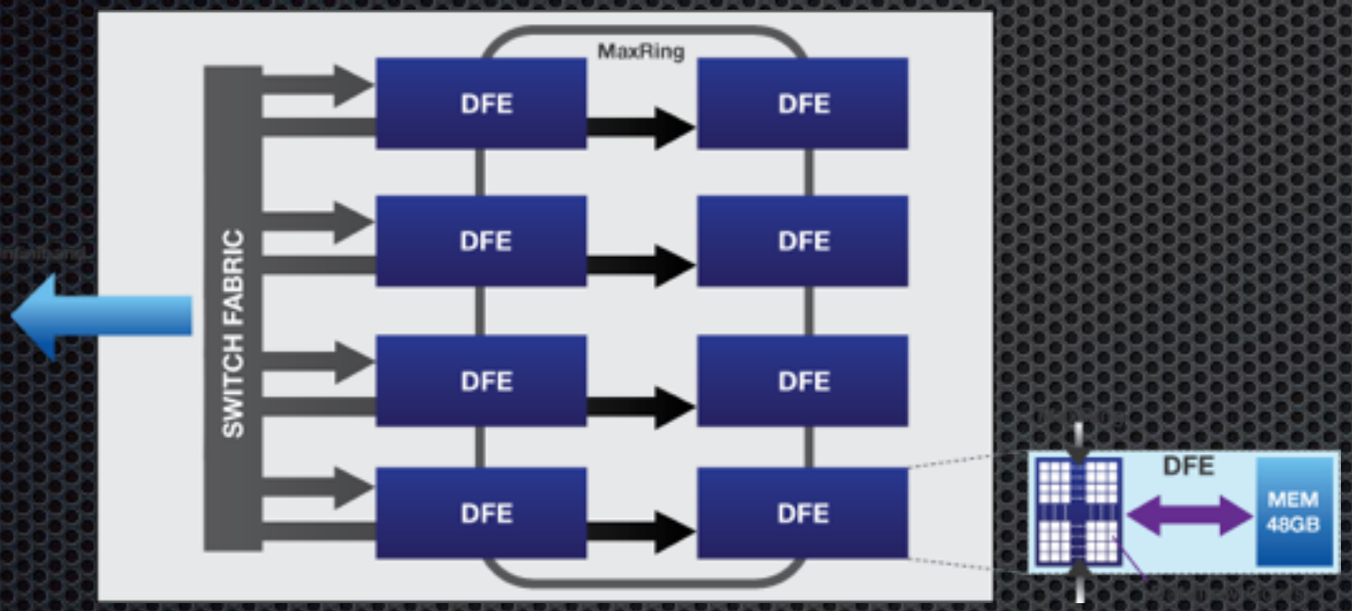
Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	3,208.83	CINECA	Eurora - Eurotech Aurora HPC 10-20, Xeon E5-2687W 8C 3.100GHz, Infiniband QDR, NVIDIA K20	30.70
2	3,179.88	Selex ES Chieti	Aurora Tigon - Eurotech Aurora HPC 10-20, Xeon E5-2687W 8C 3.100GHz, Infiniband QDR, NVIDIA K20	31.02
3	2,449.57	National Institute for Computational Sciences/University of Tennessee	Beacon - Appro GreenBlade GB824M, Xeon E5-2670 8C 2.600GHz, Infiniband FDR, Intel Xeon Phi 5110P	45.11
4	2,351.10	King Abdulaziz City for Science and Technology	SANAM - Adtech, ASUS ESC4000/FDR G2, Xeon E5-2650 8C 2.000GHz, Infiniband FDR, AMD FirePro S10000	179.15

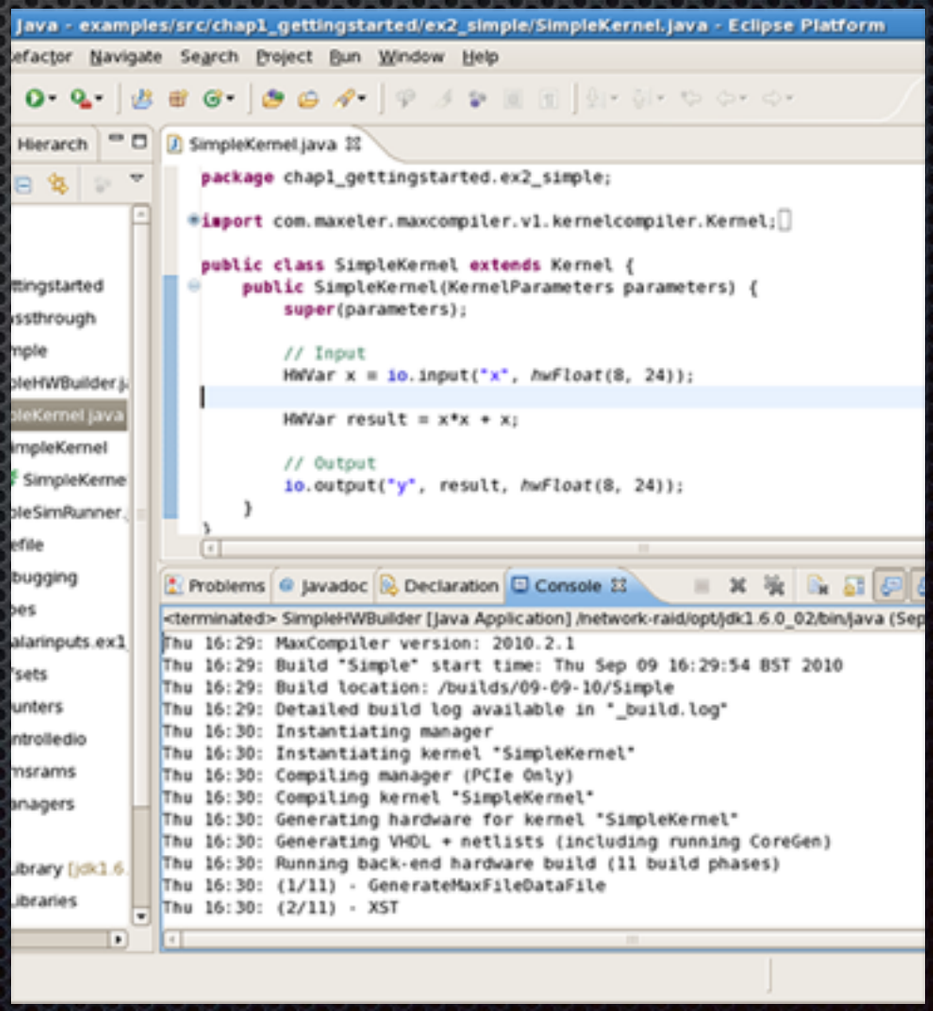
Heterogeneity.

(Italian-flavoured :P)

Custom hardware accelerators: Maxeler's approach

Focus on automatic optimization of data transfers, "synchrony" of computation and data transfers, custom data flow languages, efficient automatic translation from high level programming language to RTL



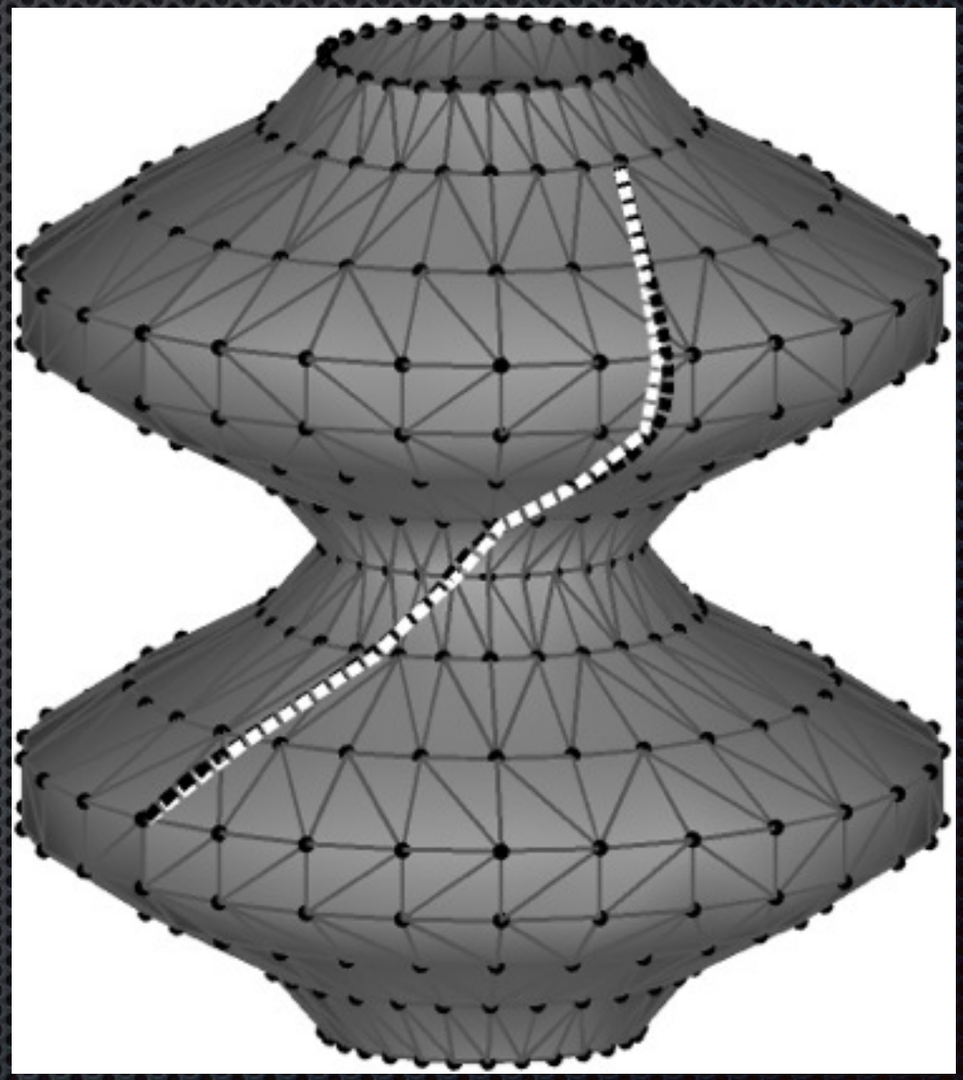


Custom Reconfigurable Hardware, and Software Development Kit

(however, other companies are in this business, too: picocomputing, convey computing, etc…)

Polyhedral Analysis

- Formal, mathematical representation of static codes as Z-polyhedra
- Optimal code tiling for automatic improvement of locality and parallelism
- For example, automatic optimization of previous code is feasible, as the code is a Weakly Dynamic, Affine Program (probably even a SCoP/ SANLP)
  - veeeery regular — static, affine — kinds of programs



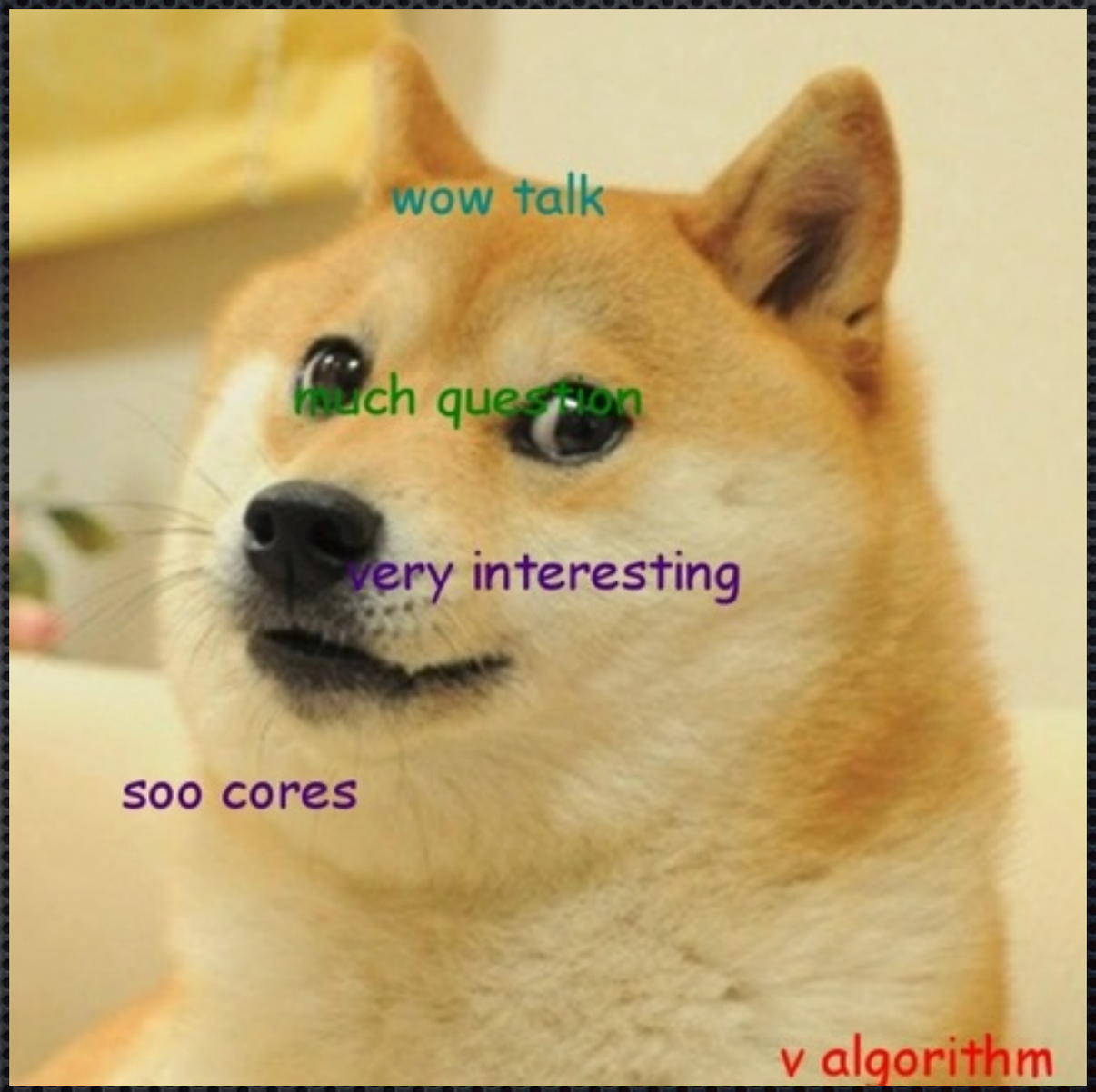
(Custom) Hardware accelerators: LIME's approach

Focus on automatic compilation (or synthesis) of a high-level, platform independent language, to a specific — potentially highly heterogeneous — platform. Currently, (partially) supports commodity multicore CPUs, specific GPGPUs and FPGAs.

Directions of future work

- Seemingly more:
  - heterogeneous
  - automatic/easier to write codes
  - power efficient
  - domain-specific

Questions?


wow talk
much question
very interesting
soo cores
v algorithm