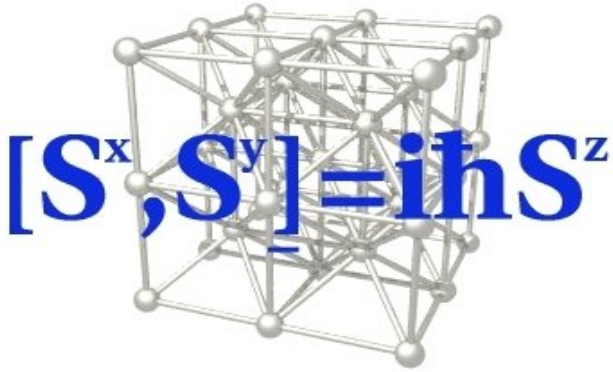
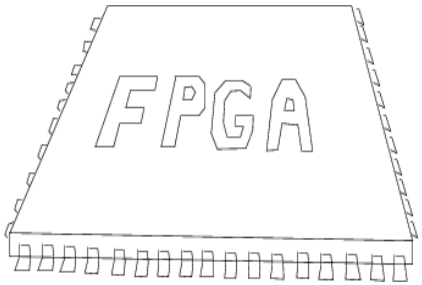
Otto-von-Guericke-Universität Magdeburg

SPINPACK – parallel performance and possible acceleration using FPGAs





Outline ...

- What is SPINPACK?
  - physical models and algorithm (very short)

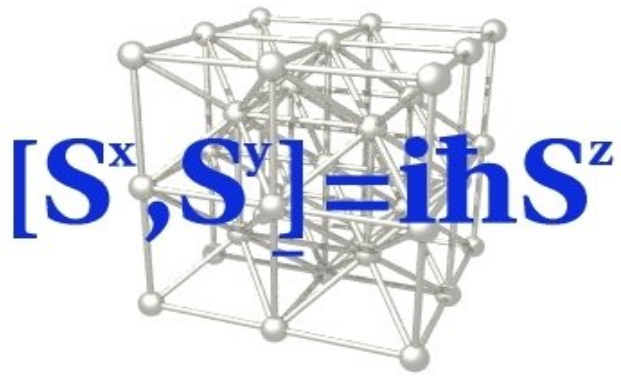
- parallel Performance (short)
  - multithread on SMP machines
  - MPI on DMP machines
  - limits and bottlenecks

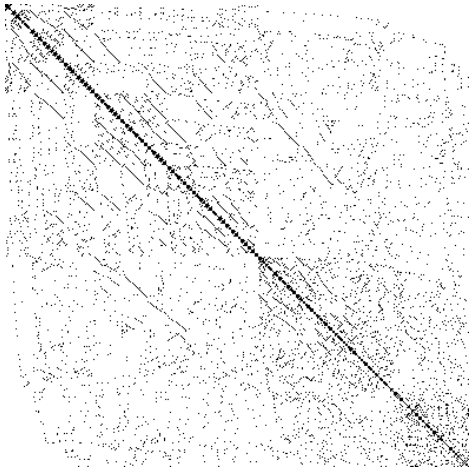
- what are FPGAs? (longer, this is interesting!)
  - expected acceleration by FPGAs

- conclusion

What is SPINPACK?

- Heisenberg-, t-J-, Hubbard model
- finite lattice with periodic boundaries (extrapolation 1/N -> 0)
- Spin operators acting on lattice neighbours
- Hilbert space: |up, down, up, up, ...> = bit patterns (oxoo...)
- Hilbert-Matrix H computed (optional stored) = sparse matrix
- get lowest eigenvalues and vectors by Lanczos-Method
- inner loop is matrix vector multiplication: H * v0 + v1
- H reduced by lattice (and other) symmetries = bit permutations





Bottlenecks:

- **vector (and optional matrix) is distributed over compute nodes**
- **lot of network communication**
- **matrix is sparse and random (random memory access)**

- need network bandwidth (~ hnz · iterations)
(hnz = number of non-zero matrix elements)
- random memory accesses (~ hnz · iterations)

(a linpack machine is not the optimum)

old (16 of 32 Alpha-Cores, 128GB, 10kW, shared memory):

- s=1/2 Heisenberg square N=40 (160 symmetries):
- Memory consumption : 7.8GB (431e6 basis states)
- Non zero matrix elements per line: 41.8 (90GB)
- Matrix in

Memory: 3min/lt (per iteration)

Disk: 240min/lt (6.2MB/s of max=533MB/s)

Computed: 275min/lt = 4.1h/lt

speedup = 1.87 ... 1.90 / CPUdoubling

2*40lt: 4h on 98GB memory OR 14-15 days on 8GB memory

MPI code (since 2008):

new (5832 MIPS-Cores, 3.8TB, 20kW, distributed memory):

- **s=1/2 Heisenberg square N=40 :**
- **Memory consumption: no hard problem, distributed over nodes**
- **Matrix in**

Memory: mind. 83 Nodes (4GB, 6Cores, 8GFPLOPs)

shared disk: --- makes no sence --- (90GB/lt/(.5GB/s)=3m/lt)

Computed: mind. 2 Nodes

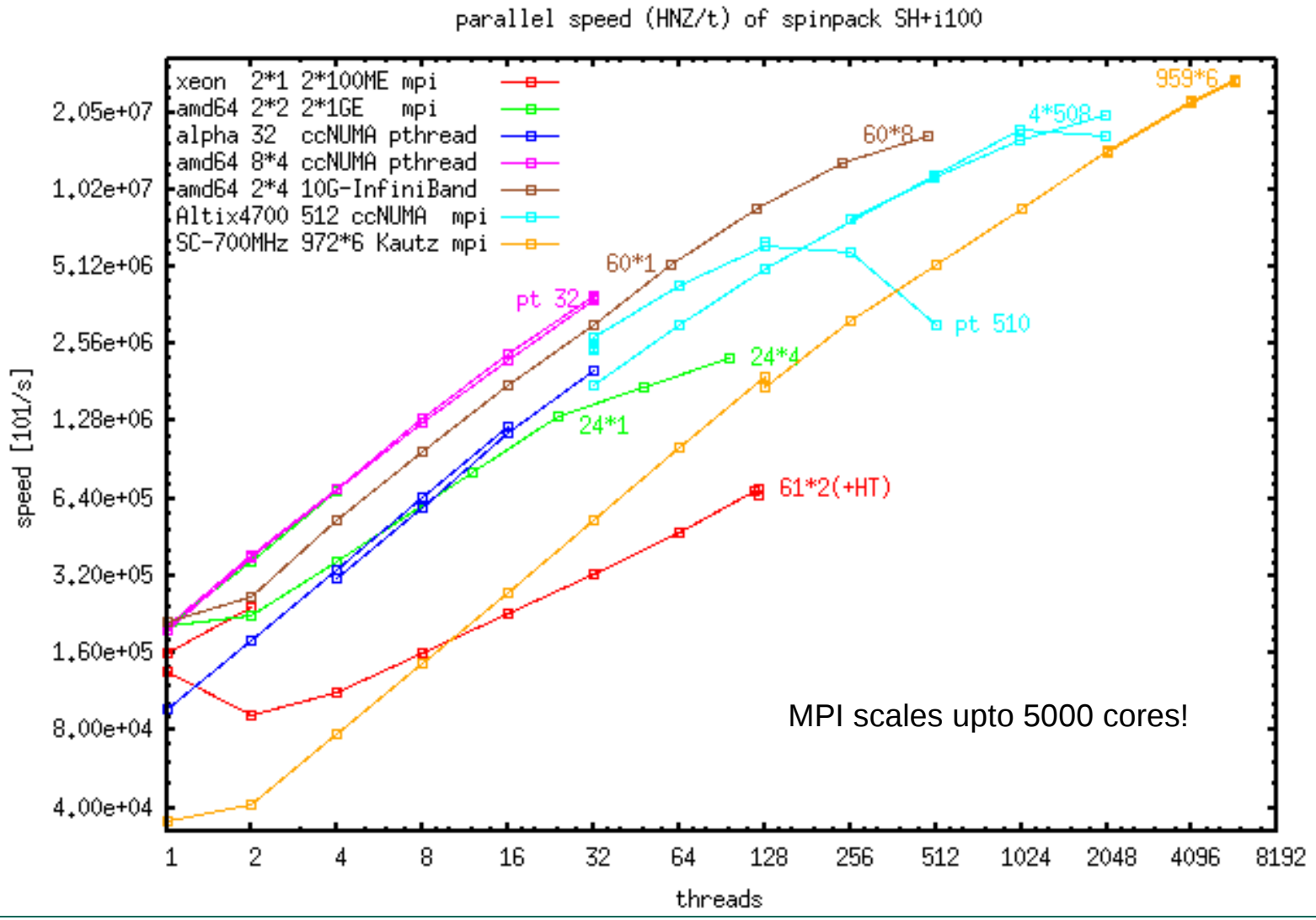
speedup = 1.55 ... 1.60 / CPU-doubling

possible:

s=1/2 kagome N=45 (1TB), square N=48 (2.4TB, 93d, dim=168e9)

overhead: collect remote memory access, transfer blockwise

Parallel performance



parallel performance:

- speedup (only) around 700 for 5700 cores (matrix stored)
- but code scales quite well (5000 cores and more)
- bigger systems (weak scaling) but increasing computation time

- further speed improvements possible?

  - further tuning vs. Flexibility, more complex code

- More interesting than speed is larger system size!

limiting parts for system size:

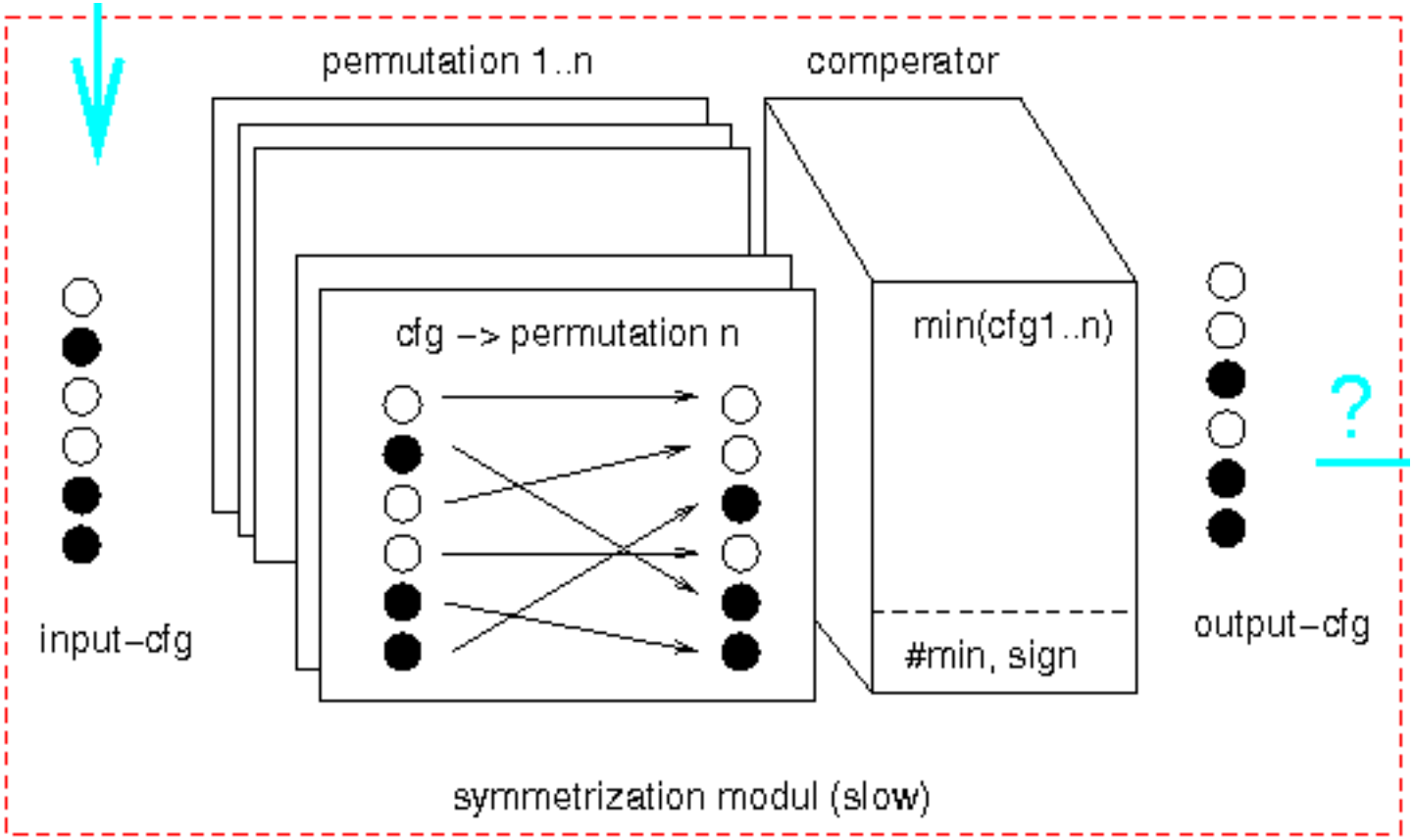
- Hilbert dimension can be 40..100 times larger if H isn't stored
- but generation of matrix is 100 times slower than reading H

Why? - Lot of bit-permutations must be done via loop.

but could be done via crossed wires in hardware easily

which is not possible using x86-like CPU

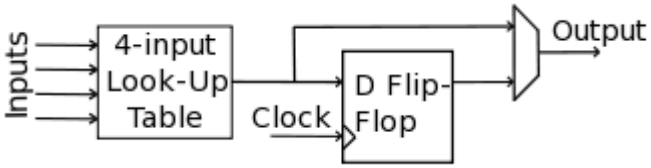
Slowest code part

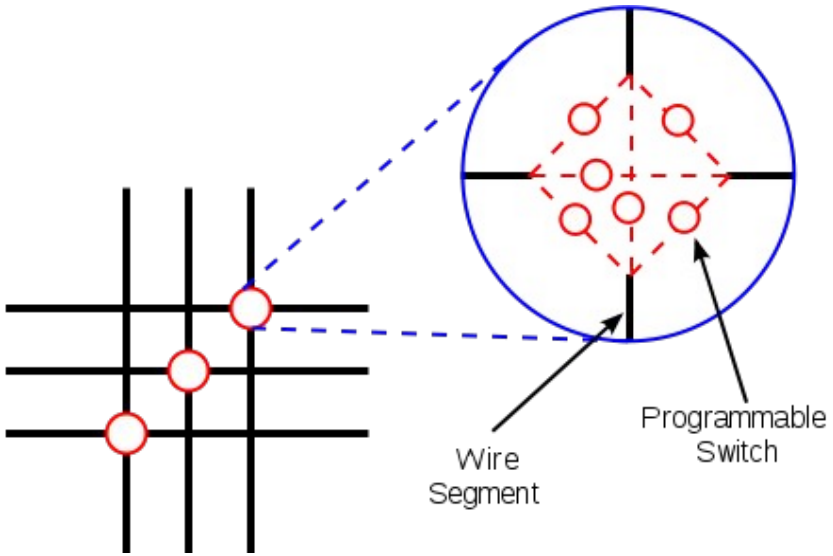


Permutations are just wires in hardware!

- **x86 = von Neumann architecture (fixed hardware + software)**

- **Field-Programmable Gate Array (FPGA, configware)**
- **offer the possibility to create optimal hardware**
- **100000 LUTs (lookup tables, universal, 4 in, AND, OR, XOR, ...)**
- **programmable wires**





Pictures from Wikipedia

acceptable bandwiths between CPUs and FPGA modules

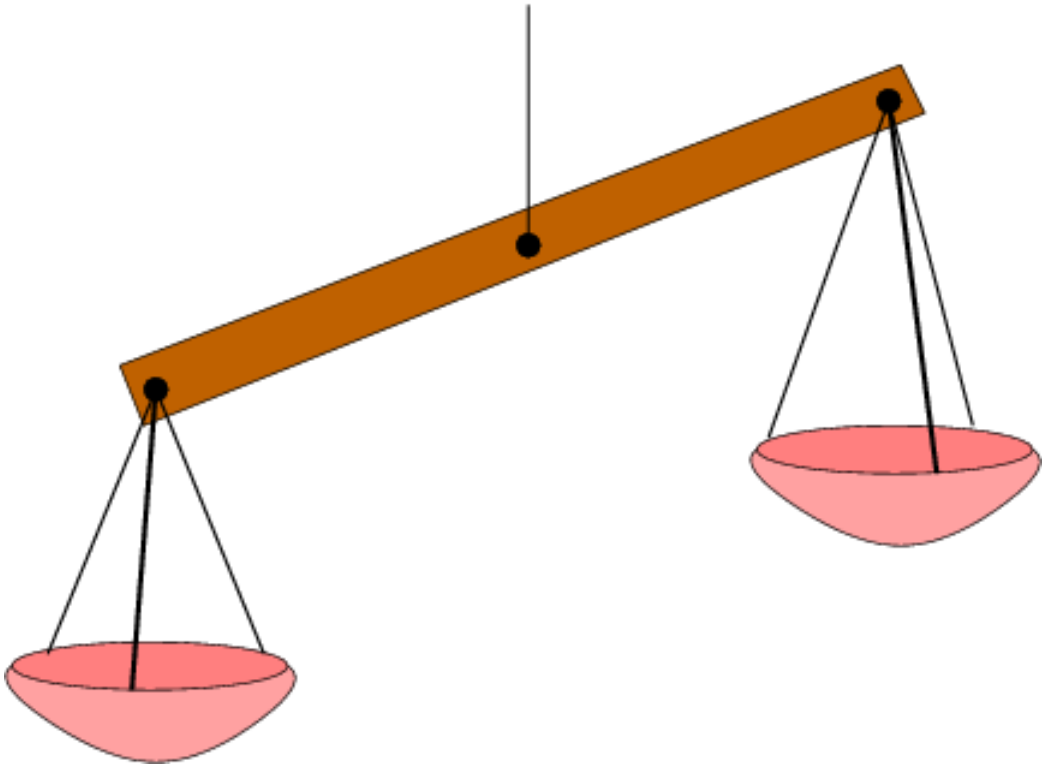
- available as PCI-cards or CPU replacement
- using PCI bus or HT bus
- communication via streams or main memory

Picture from Wikipeda

FPGA advantages vs. disadvantages ...

Pro's & Con's

- **flexibility**
- **performance**
- **low power**
- high level languages
- programming effort
- expensive float
- **low clock (50-500MHz)**
- **missing open standarts**



using Impulse-C + HDL to generate Hilbert space

- generating Ising states using Sz-conservation
  - counting bits until first 01 bit pattern
  - shift bit and add 000...01...111
    like: 000111 -> 001011 -> 001101 -> 001110 -> 010011 -> ...
    x86: 60..70 clocks FPGA: 1 clock

```
while(1) {
#pragma CO FLATTEN // want to have 1clk/loop
#pragma CO SET StageDelay 400 // max. atomic operations/clk
  for (ii=NN-1;ii>=0;ii--){ // about 33 stages
#pragma CO UNROLL // ii must have a fixed range, -33 stages
    if ((co_int2)1==(co_int2)(3&(char)(cfg>>ii)) // 2bit comp.
      { i=ii; cfg2=(cfg>>ii)<<ii; cfg3=((tbase)3)<<ii; }
      // x<<const. = 1 stage
   } // 1 cycle, (2 stages (2bit comp.) - 1 overlap) * NN
....
```

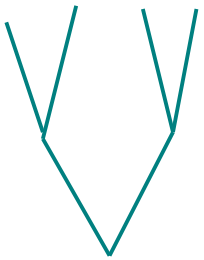
using Impulse-C + HDL code to generate Hilbert space

- filter out symmetric states
  - test/get minimum of 160 spin permutations
    x86: 600 clocks (test minimum) FPGA: 1 clock
    x86: 6000 clocks (get minimum)

```
-- HDL code
entity permute000 is
   port ( signal inp  : in  std_ulogic_vector(63 downto 0);
          signal outp : out std_ulogic_vector(63 downto 0));
end;
architecture test of permute000 is
begin
  outp <= inp[34] & inp[28] & inp[21] & inp[14] & ...;
end test;
...
```

using Impulse-C code to generate Hilbert space

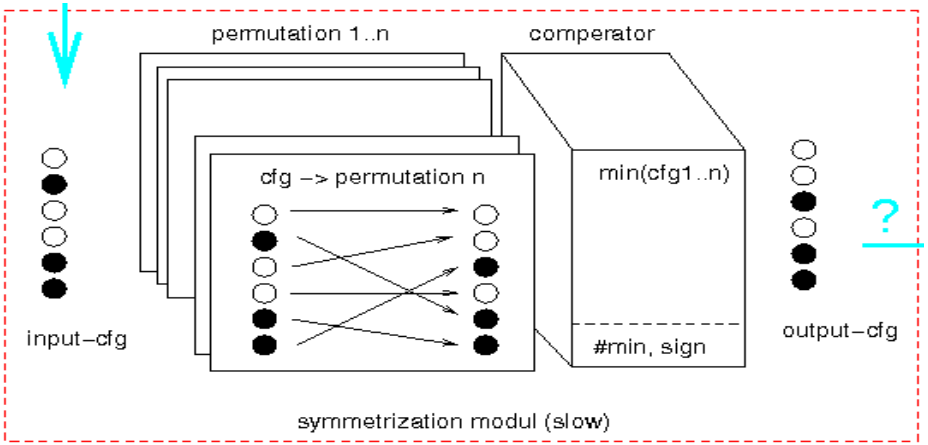
```
// -- Impulse-C code
#pragna CO PIPELINE     // 1 clock, 2 stages
#define MinCfg(a,b) ((a<b)?a:b)
permute000(incfg, &tmp_0_000); // call HDL function
permute001(incfg, &tmp_0_001);
...  // parallel! 0 stages
tmp_1_000 = MinCfg(tmp_0_000, tmp_0_001);
tmp_1_002 = MinCfg(tmp_0_002, tmp_0_003);
...  // parallel!
tmp_2_000 = MinCfg(tmp_1_000, tmp_1_002);
tmp_2_004 = MinCfg(tmp_1_004, tmp_1_006);
...
... // log2(Nsym) levels
...
tmp_8_000 = MinCfg(tmp_7_000, tmp_7_128);
outcfg = tmp_8_000;
```



using Impulse-C to generate Hilbert space

- get symmetric state
= test/get minimum of 160 spin permutations
x86: 6000 clocks FPGA: 1 clock

25% of a FPGA chip does the same task in one clock
assuming 2GHz x86 vs. 100MHz FPGA: speedup=300!



Comparision ...

N = 40 s=1/2 Heisenberg model on square lattice:
vector size = 431e6
matrix size = 41.8 * 431e6

comparision:
- MPI + H stored: 81 x 4GB-Nodes (needs much memory)
- MPI + H not stored: 2 x 4 GB Nodes (needs to much time)
- MPI + H not stored: 2 x 4 GB Nodes incl. 2 FPGAs
cheaper, less power consumption
about same speed like H stored (may be faster)

Conclusion ...

- **FPGAs offer excellent performance improvements SPINPACK**
- **First FPGA experiments provide estimations**
  one FPGA same speedup to 100 x86 !?
- **acceptable programming effort (Mitrion-C, Impulse-C, VHDL)**
- **Hardware becomes more attractive**

may be interesting for other methods
especially where symmetry operations
dominating the computation time