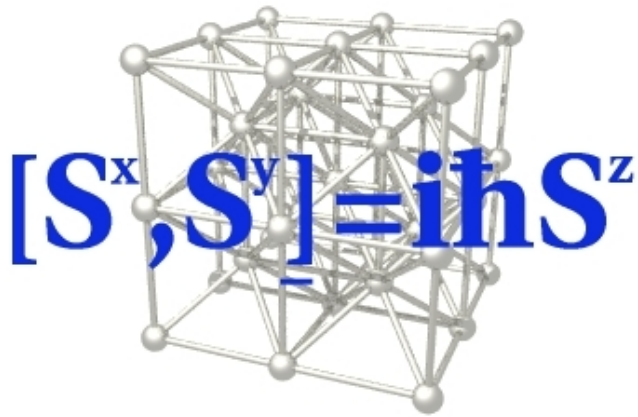
FPGAs for quantum spin systems

Jörg Schulenburg

October 29, 2009

What is SPINPACK?

Lowest eigenvalues and eigenvectors for quantum spin systems on finite lattices
http://www.ovgu.de/jschulen/spin/.



- Heisenberg-, t-J- or Hubbard model
- finite lattices with open or periodic boundary conditions

What is SPINPACK?

- Spin operators ($\hat{S}_i^+ \hat{S}_j^-, \hat{c}_i^+ \hat{c}_j, \hat{n}_i, \ldots$)
- acting on lattice neighbours (ij)
- Hilbert space: $|\uparrow\downarrow\uparrow\uparrow \ldots\rangle \rightarrow$ bit patterns ($0100\ldots$)
- Hilbert matrix H computed (sparse matrix)
- lowest eigenvalues and vectors (Lanczos method)
- H reduced by lattice symmetries

Example: N=6 chain

- Spin operators (12 Ops: $\hat{S}_i^+ \hat{S}_j^-$)
- 12 point symmetries (shift + reflection)
- Hilbert space ($(6!/3!^2 = 20)/12 \approx 3$):
  represented by: 000111, 001011, 010101

$$
\begin{aligned}
c_0: &\quad 000111 + 001110 + 011100 + 111000 + 110001 + 100011 \\
c_1: &\quad 001011 + 010110 + 101100 + 011001 + 110010 + 100101 \\
+ &\quad 001101 + 011010 + 110100 + 101001 + 010011 + 100110 \\
c_2: &\quad 010101 + 101010
\end{aligned}
$$

Example: N=6 chain

- ...
- Hilbert matrix H computed (3x3 sparse matrix)
- $\hat{S}_0^+ \hat{S}_5^- \ |000111\rangle = |100110\rangle$
- symmetrization: $100110 \longrightarrow 010011$ (bit manipulation)
- index search: $010011 \longrightarrow c_1$ (read memory)
- use matrix element: $\langle c_1|H|c_0\rangle$ (FLOPs)

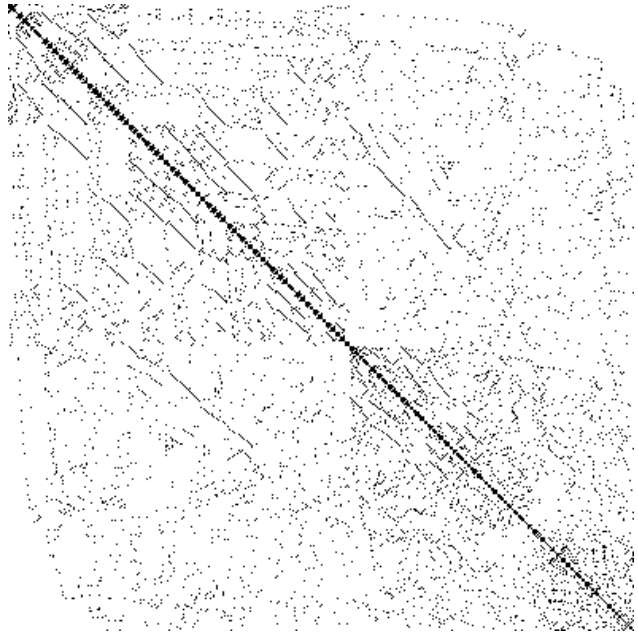
computation effort

- inner loop is matrix vector multiplication: $H \cdot v_0 + v_1$
- H can be stored or must be recomputed
- recomputing: factor 100 slower on single processor (but 1/40th memory)

Parallelization

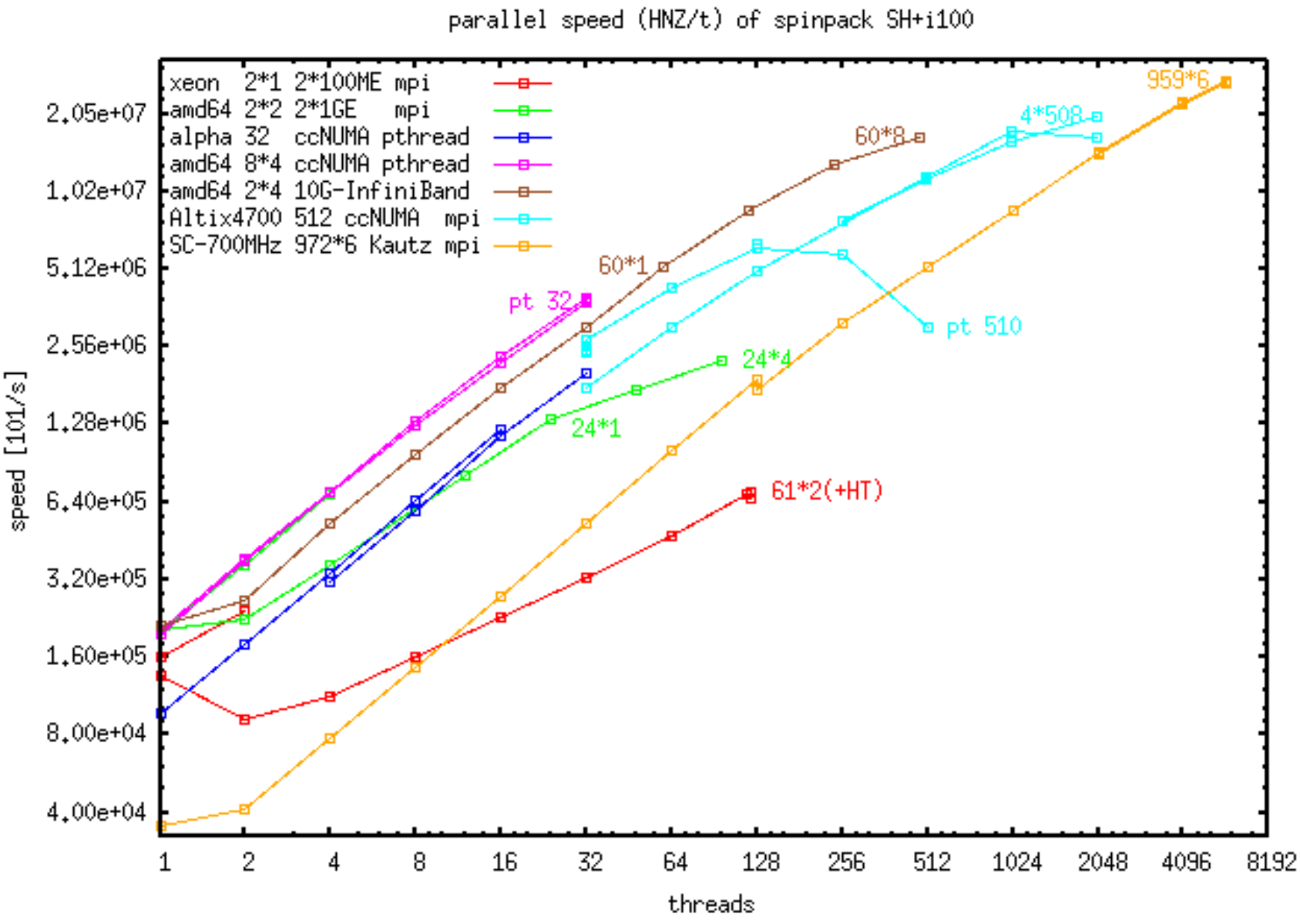
Parallelized sparse matrix problems are communication intensive.

- matrix is sparse and randomly distributed
- ca. 1.8 speedup per CPU doubling



- speedup on 5700 cores (700MHz) is only around 700 (matrix stored)
- recomputation of H scales better
- ... larger system size (but much more slow)

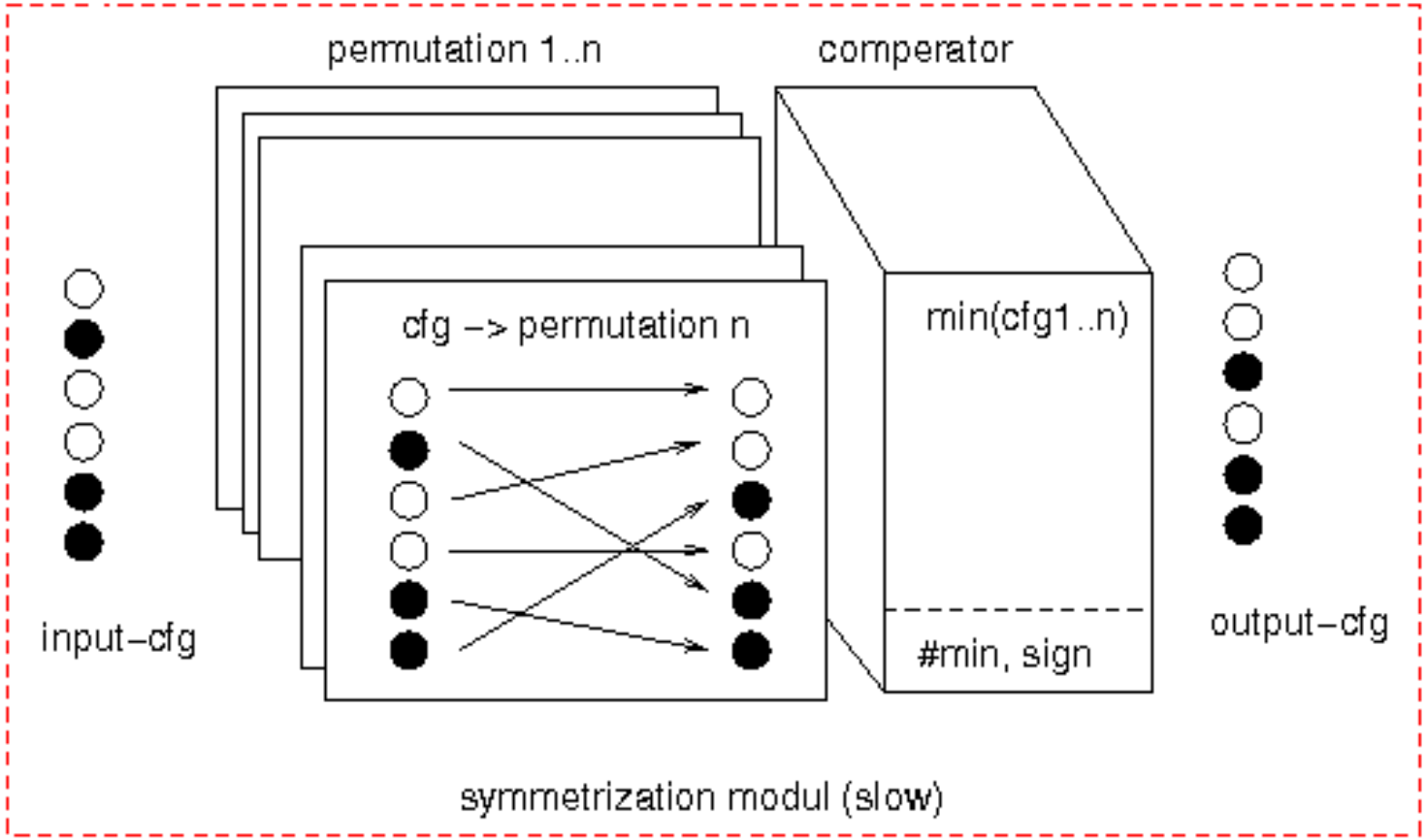
Parallelization MPI scaling



limiting factors

- compute time
- more important is memory size
- cluster solving both, but create new communication bottle neck

most compute intensiv part



Bit permutations

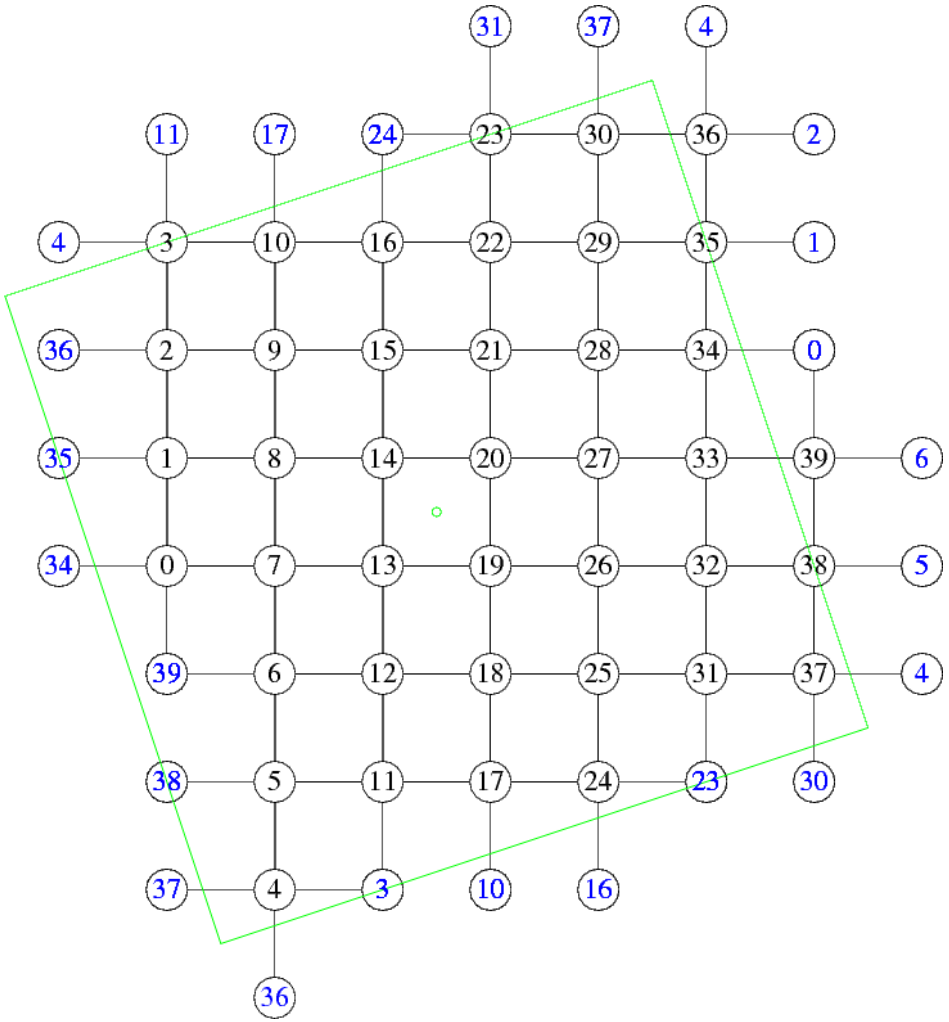
Most compute intense is the symmetrization of Hilbert space.

- H does bit flip operations
- lattice symmetry operations are bit permutations
- slow on x86
- Heisenberg: get bit i, put bit j (N times)
- Hubbard: also count bits between i and j (parity)

```
for (cfg2=0, m=0; m<nn; m++, cfg>>=1)
  if (1&cfg) cfg2 |= 1L << sym[m];
```

- ca. 5*nn x86 clocks (10 asm lines)
- additional 1-2 clocks for minimum(cfg) computation

N=40 sample



- $S^z = 2$ (709e6)
  15GB + 150GB
- $S^z = 0$ (431e6)
  8GB + 90GB
- 5400 SC-cores:
  H 7min
  100lt 35min

N=40 sample - estimation

- sample N=40 site square lattice, 160 symmetries,
- x86 clocks: ca. 5 * 40 * 160 = 32000 ($11\mu s$)
- $H_{ij} * v_j + v_i$ needs only one x86 clock!
- about 42 non-zero matrix elements per line (max. 709e6 lines for $S^z = 2$)
- about 88 3GHz-CPU-hours per Matrix
- equivalent to about 1MB/s (reading 238GB matrix elements)

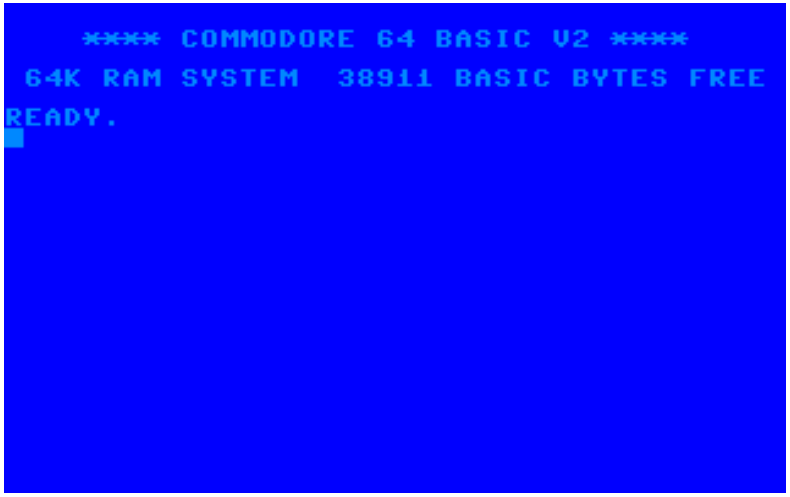
now: FPGAs

First contact ...

- Sept 2005: SGI announced RC100 blades for Altix-Server
- 2 FPGA-Chips (Field-Programmable Gate Arrays) via NUMAlink (6.4GB/s)
- 2 Xilinx Virtex 4 LX200 FPGAs (200000 Logic Cells per FPGA)
- ...
- Q1: What is a FPGA?
- Q2: Could it be usefull for SPINPACK? (2007, 2009)

Fascinating ...

C64 als FPGA hardware emulation (syntiac.com).



- ... discovered searching google for FPGAs
- A realtime Commodore 64 emulator in a FPGA
- 6502/6510 compatible CPU
- VIC-II video chip emulation

Other projets: Minimig (Amiga 500) ...

FPGA manufacturers

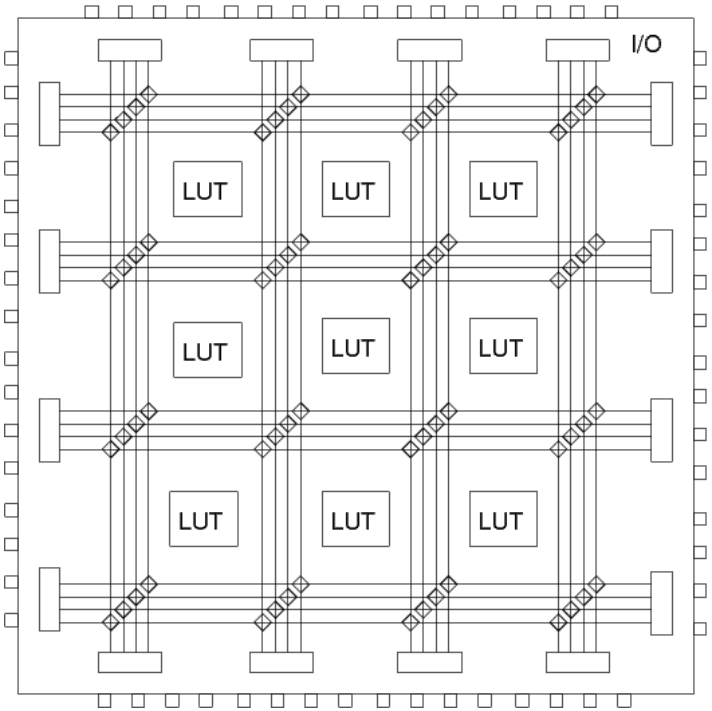
- Chip Manufacturers: Xilinx, Altera, Actel, Atmel
- Xilinx - Spartan family (low-power, high volume)
- Xilinx - Virtex family (HPC)
- Altera - Stratix or Cyclone chip family

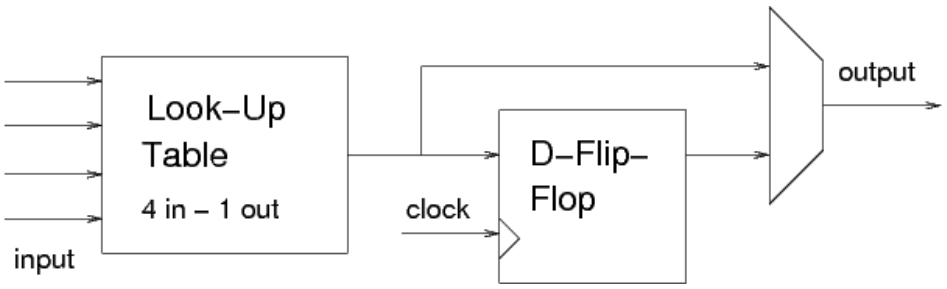
FPGA accelerators

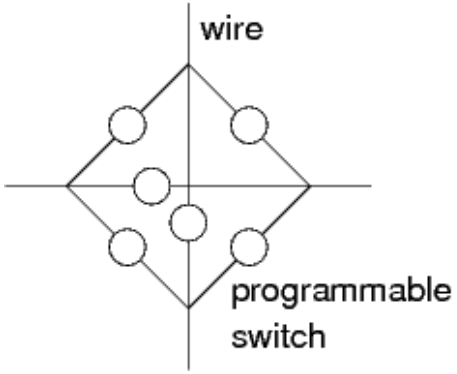
- DRC: Virtex FPGA as AMD coprocessor via HyperTransport
- Nallatech: Virtex FPGA for IBM BladeCenter via PCI-X
- Pico: E-12 PCMCIA/CF cards using Virtex FPGA
- SGI RASC: Virtex FPGA via NumaLink
- SRC: Stratix FPGAs, StandAlone or PCIe

Field-Programmable Gate Array

- $10^5..10^6$ LUTs (lookup tables, 4 in, 1 out)
- programmable wires/switches

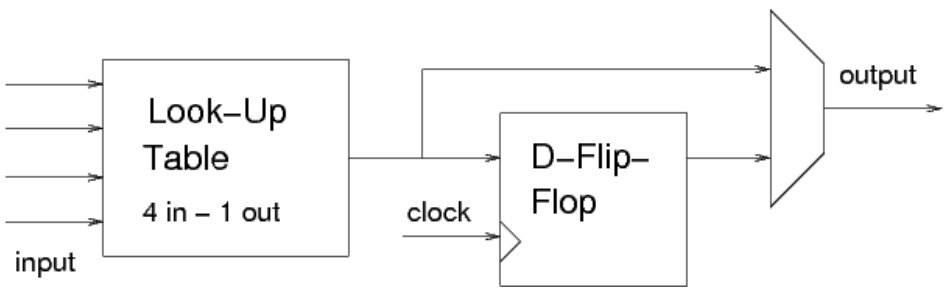






Field-Programmable Gate Array

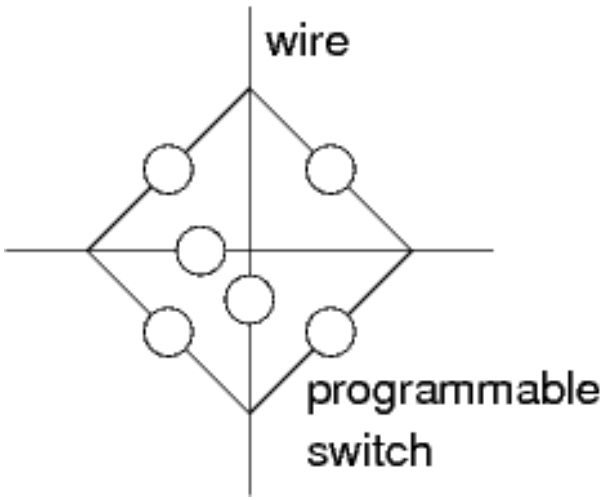
- $10^5..10^6$ Look-Up Tables (4 in, 1 out, NAND, XOR, ...)
- 16 bit per look up table (bitstream)



$i_3 i_2 i_1 i_0$	out
0000	1
0001	1
0010	1
...	...
1111	0

Field-Programmable Gate Array

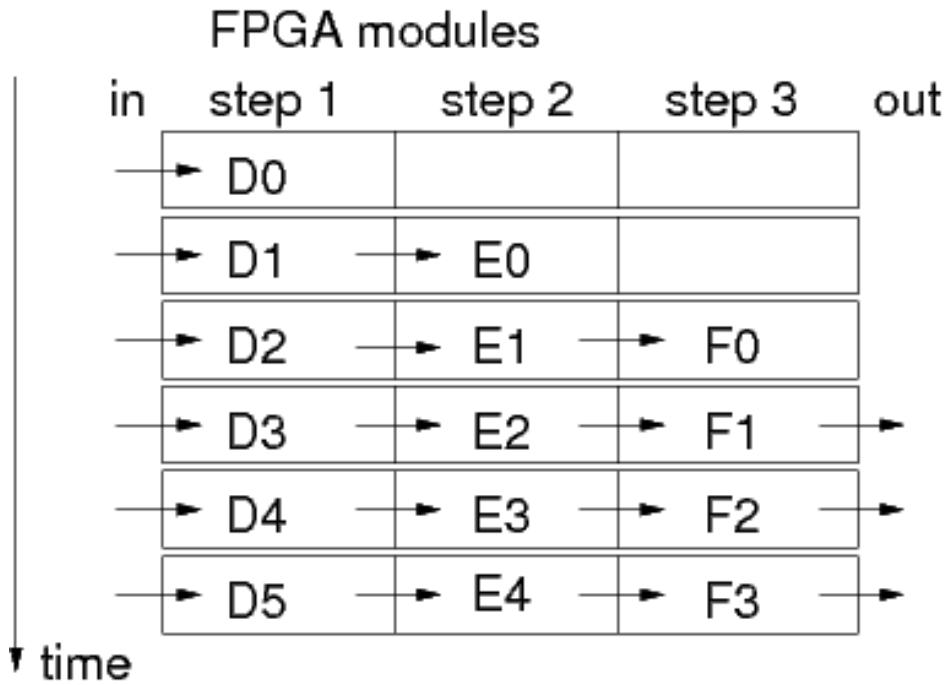
- programmable switches (connecting 4 wires)
- 6 bit per 4-wire switch (bitstream)
- wire network (details are the secrets of manufacturers)
- ... covered in the place & routing software



Field-Programmable Gate Array

- programmable clock units (50..300MHz)
- clock limited by signal propagation through longest path
- pipelining (via D-Flip-Flops)

pipelining example: getting 1 result per clock (after 3 clocks latency)



Field-Programmable Gate Array

... may also contain

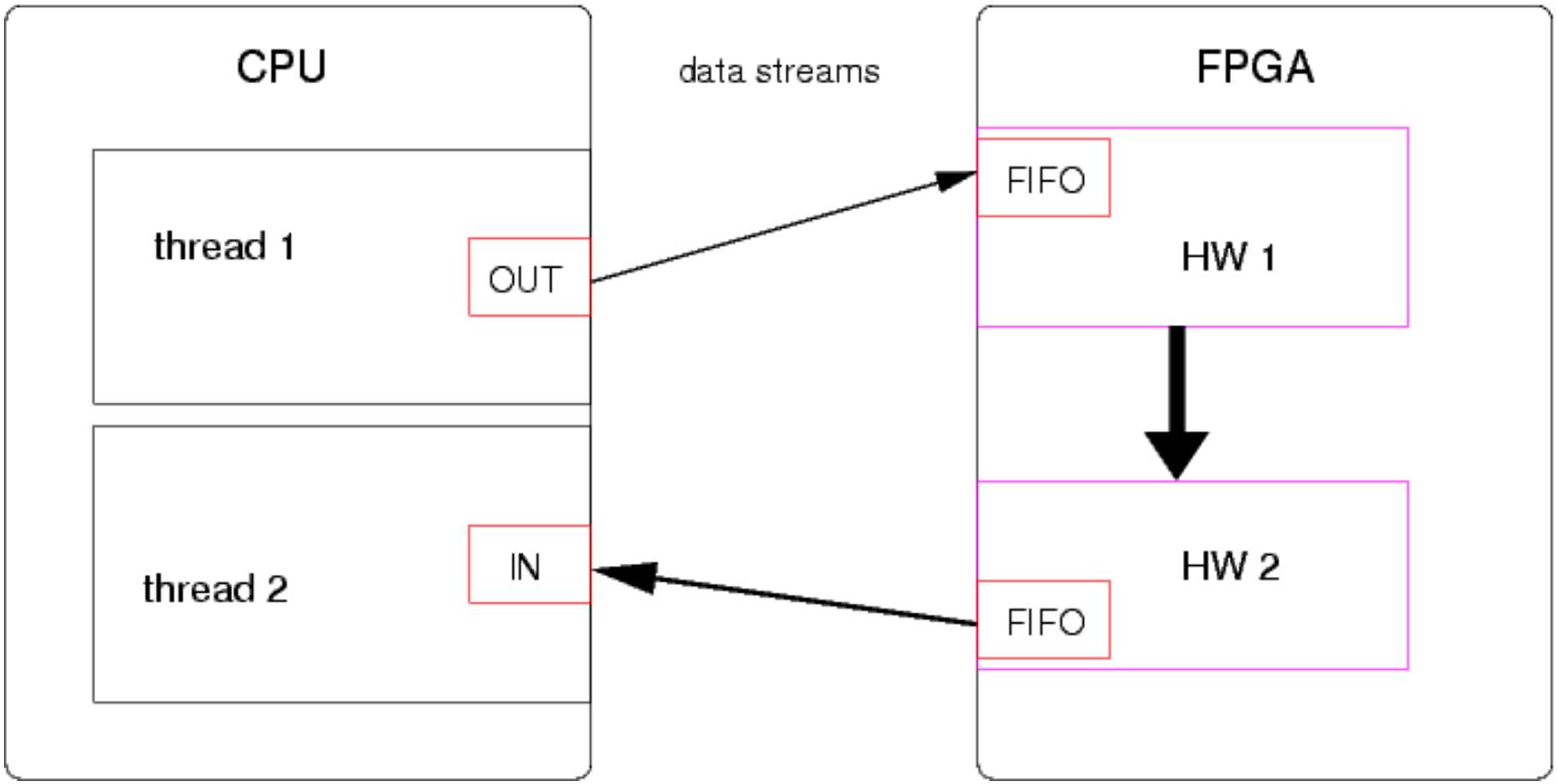
- additional FP-ALUs
- embedded CPU-Cores (PowerPC 405/440)
- DSPs (digital signal processors), integrated memory blocks, ...

I/O-Bandwidth

- acceptable bandwidth to CPUs
- available as PCI-cards or CPU replacement
- ... using PCI or HT bus
- communication via streams or main memory

Project QPACE: QCD on Cell - 100 TFLOP super computer, Cell processor coupled via configurable hypercubus network realized by Virtex-5 FPGAs.

FPGA stream concept



computation costs

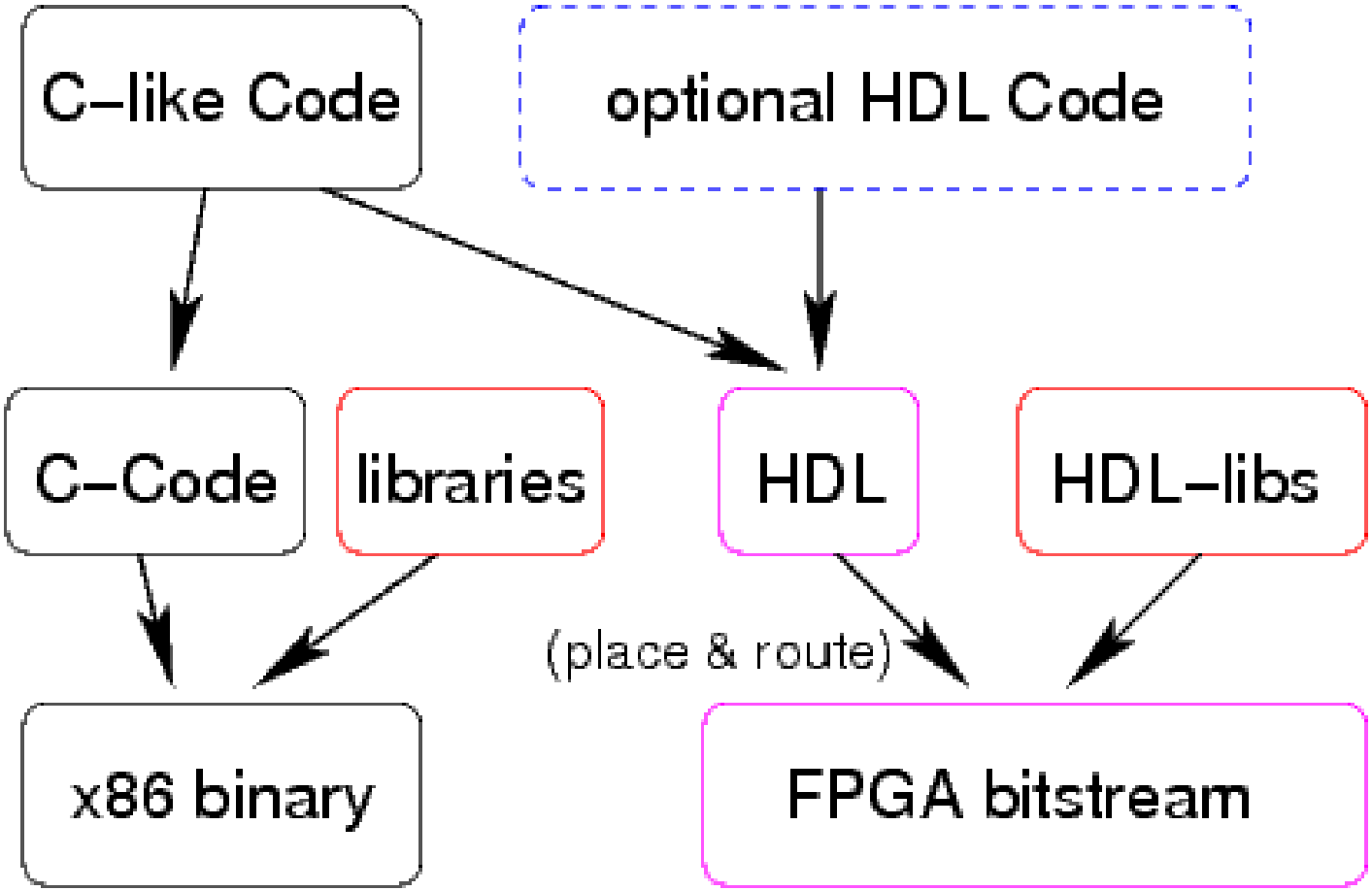
- optimization for time (clocks) or size (LUTs) is user decision
- best estimation: number of LUTs needed for algorithm
- ... 2 LUTs full adder (2 bit + carry in, 1 bit + 1 carry out)
- ... but ...
- part of FPGA is needed for communication to CPU/memory/IO
- losses of LUTs by complex routing
- in case of pure FP-peak comparable to serial processors (Virtex-5 = Quad-Opt-2.5GHz = 40 GFLOPs)
- (much) better in other cases (fix point, integer, bitmani.)

Software

- hardware description language: Verilog or VHDL (asm like)
- high level languages (C to HDL):
- ... Mitrion-C (Extension of C)
- ... Impulse-C (Subset of C + pragmas)
- generating HDL code

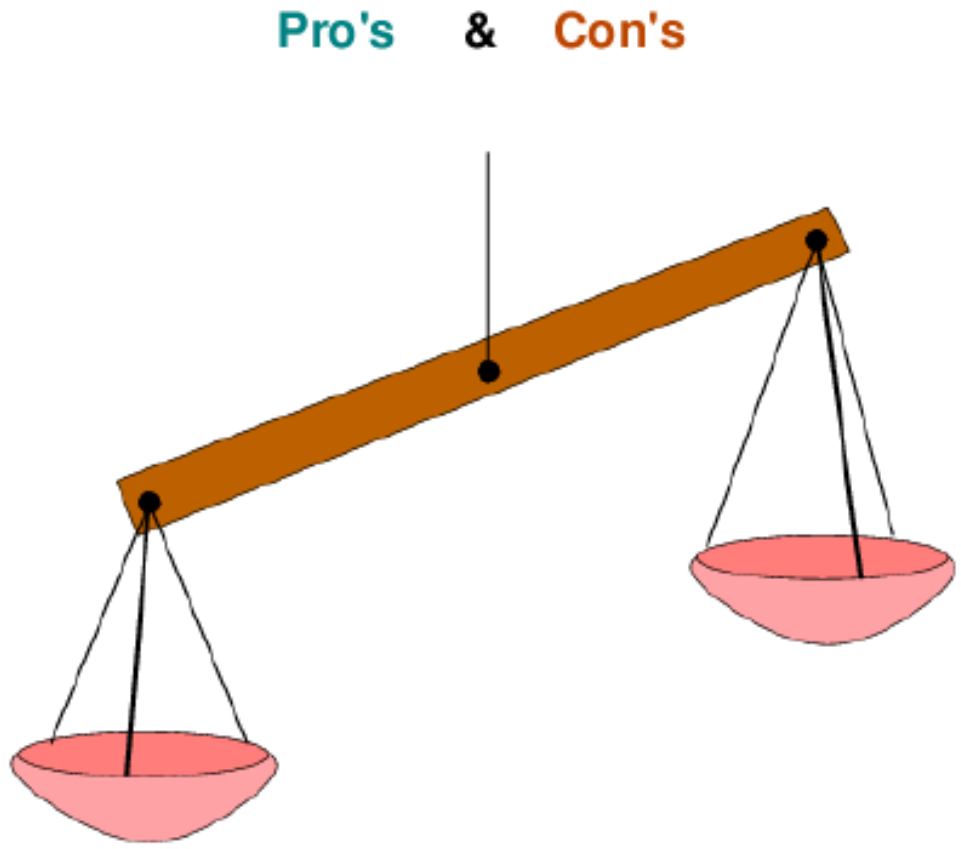
Software flowchart

- computation part + data transfer (libraries)



advantages vs. disadvantages

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



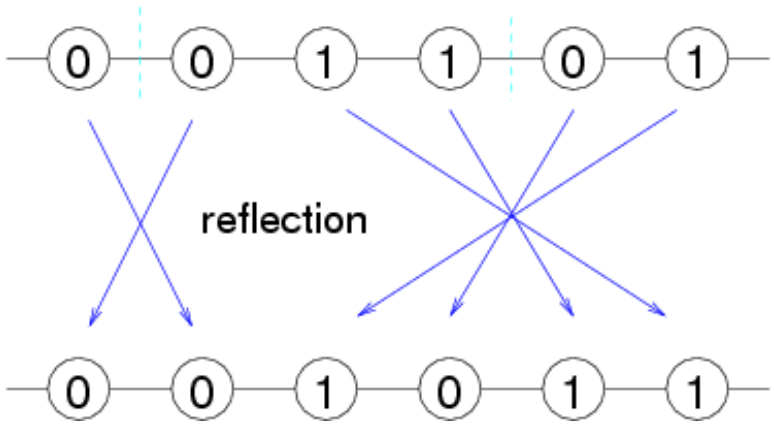
lets try out ... (2009)

- extract core of 10000 source lines (few lines, avout 100)
- porting to FPGA
- starting with Impulse-C (C compatible, easy to understand)
- benchmark $N = 40$

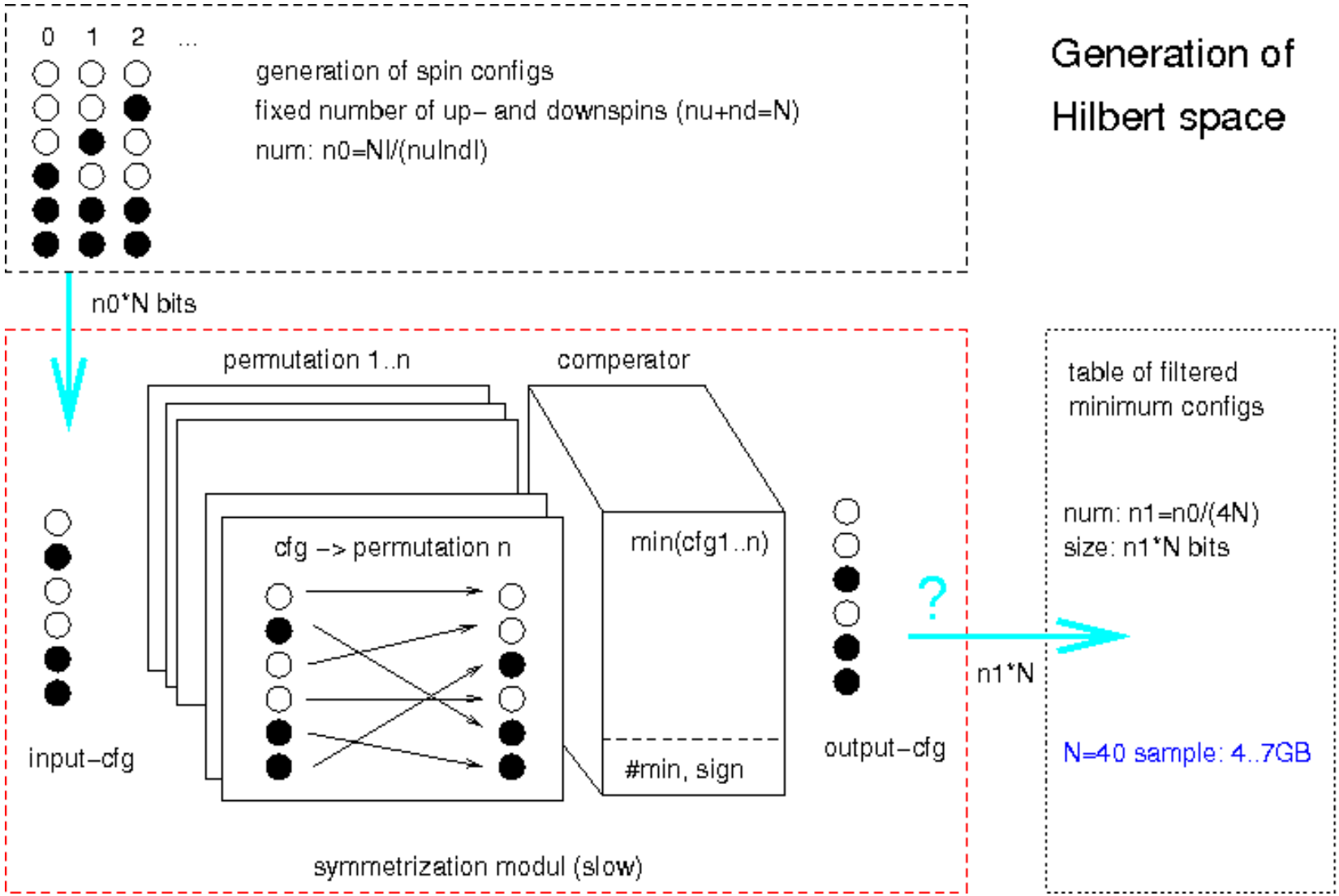
Example: Hilbert space generation

- part 1: generate configurations (fixed number of 1 bits)
  $000111 \rightarrow 001011 \rightarrow 001101 \rightarrow 001110 \rightarrow 010011 \rightarrow \ldots$
- part 2: filter out symmetric configurations

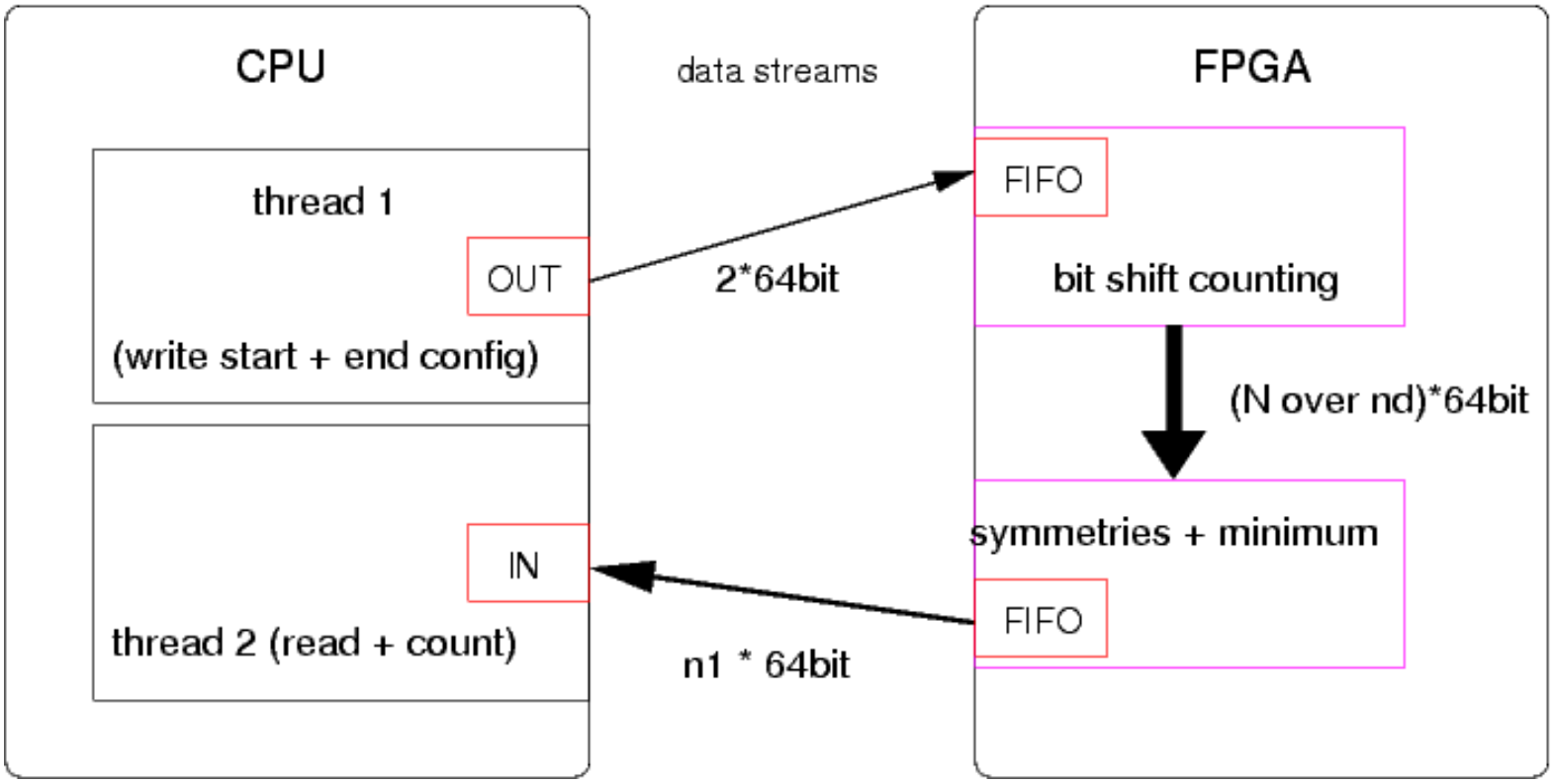
input	mincfg?	
000111	min	
001011	min	
001101	= 001011	←
001110	= 000111	
010011	= 001011	
...	...	



Example: Hilbert space generation



Example: stream concept



Part 1: conventional C for x86

Using Impulse-C to generate Hilbert space using S^z-conservation.

- constant number or 0 and 1 bits
- counting 1 bits after lowest 01 bit pattern
- shift bit and sort right bits $\longrightarrow$ *1000...01...11
- 000111 $\longrightarrow$ 001011 $\longrightarrow$ 001101 $\longrightarrow$ 001110 $\longrightarrow$ 010011 $\longrightarrow$ ...

```
cfg >>= i;  // 001011 -> 0010 (i=dd=2)
for(;;){ // dd - right side 1 bits
  if (cfg&1) dd++; else if (dd) break; // 001(0)
  i++; cfg>>=1; }
cfg = ((cfg|1)<<i) | (~((~0)<<(--dd))); // -> 001(1)01
i=dd;  // i=dd=1
```

Part 1: Impulse-C

- 000111 → 001011 → 001101 → 001110 → 010011 → ...
- x86: 60..70 clocks, FPGA: 1 clock

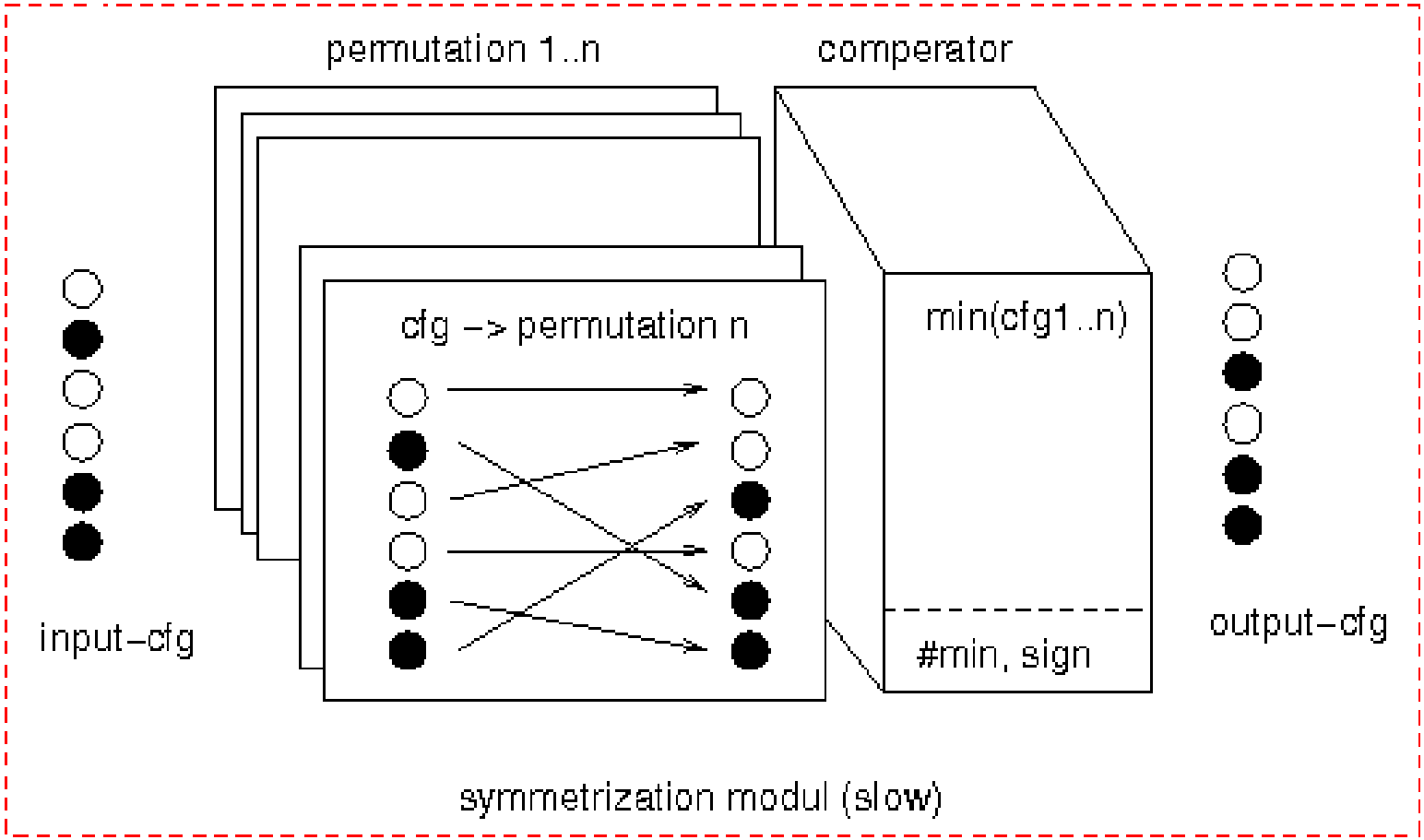
```
#pragma CO FLATTEN // want to have 1clk/loop
#pragma CO SET StageDelay 400 // max. atomic ops/clk
for (ii=NN-1;ii>=0;ii--) { // about 33 stages
#pragma CO UNROLL // range ii must be fixed, -33 stages
  if ((co_int2)1==(co_int2)(3&(cfg>>ii)) // 2-bit-op
    { cfg2=(cfg>>ii)<<ii; cfg3=3L<<ii; }
} // shift by constant costs 1 stage
cfg2=cfg^cfg2; cfg1=0; // delay=1 only (get lower bits)
for (ii=0;ii<NN;ii++){  // ii++: delay=32+1
#pragma CO UNROLL  // delay 41, (4bit blocks delay=11)
  if ((co_uint1)(1&(cfg2>>ii))) cfg1=(cfg1<<1)|1; }
cfg = ((cfg^cfg3)^cfg2) | cfg1; // 1 stage
```

Part 1: Impulse-C unrolled

- 000111 → 001011 → 001101 → 001110 → 010011 → ...
- x86: 60..70 clocks, FPGA: 1 clock

```
// unrolled + simplified (no type conversions shown)
if (1==(3&(cfg>>39)) {cfg2=(cfg>>39)<<39; cfg3=3L<<39;}
if (1==(3&(cfg>>38)) {cfg2=(cfg>>38)<<38; cfg3=3L<<38;}
...
if (1==(3&(cfg>>0)) {cfg2=(cfg>>0)<<0; cfg3=3L<<0;}
cfg2=cfg^cfg2; cfg1=0;
if ((1&(cfg2>>0))) cfg1=(cfg1<<1)|1;
if ((1&(cfg2>>1))) cfg1=(cfg1<<1)|1;
...
if ((1&(cfg2>>39))) cfg1=(cfg1<<1)|1;
cfg = ((cfg^cfg3)^cfg2) | cfg1;
```

Part 2: state symmetrization

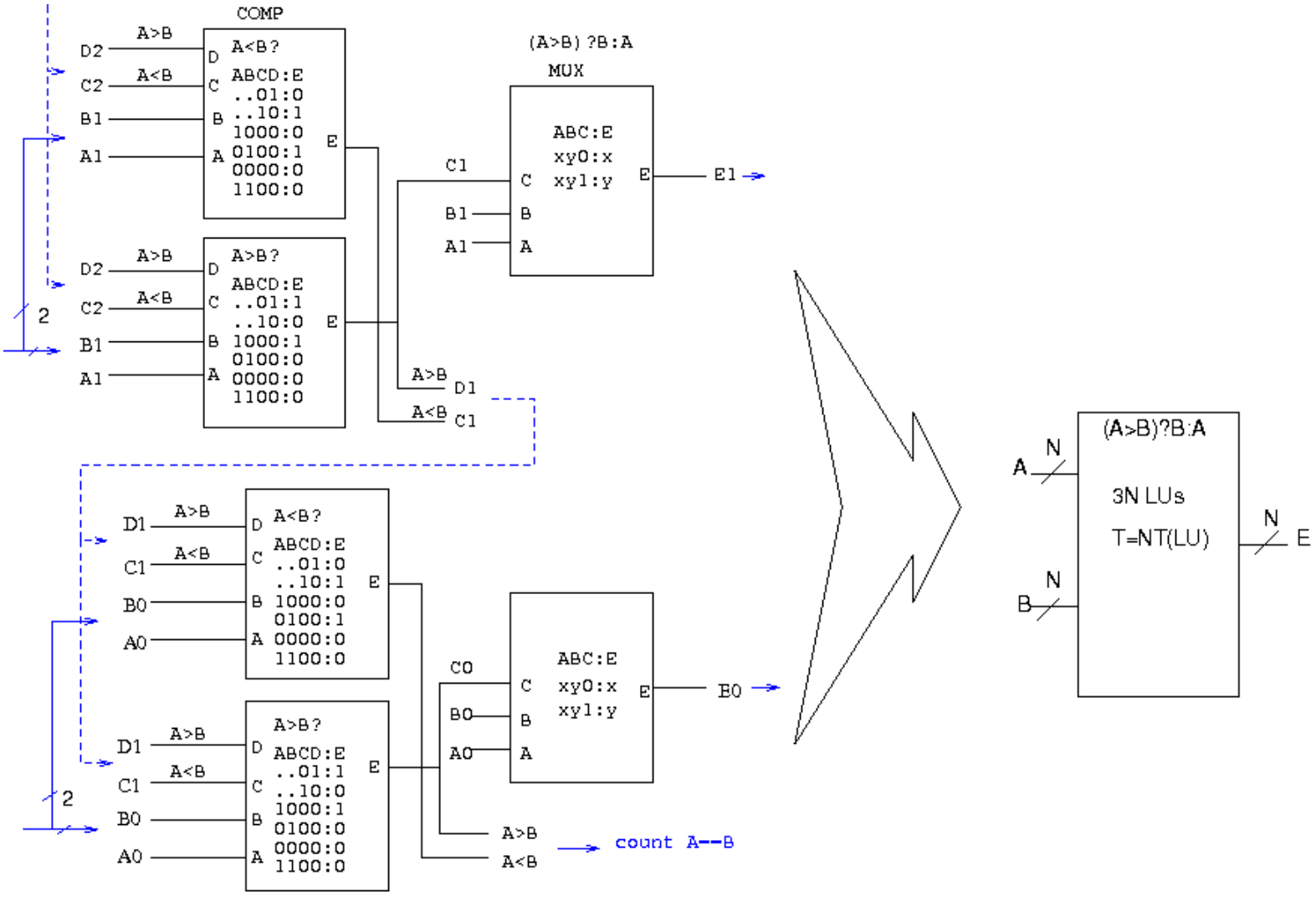


Part 2: state symmetrization - permutations

```
for (cfg2=0, m=0; m<nn; m++, cfg>>=1)
  if (1&cfg) cfg2 |= 1L << sym[m];
```

- x86: min search allows optimizations (breaking m loop)
- bit permutations are just wires ($160 \cdot 40$)
- ... but no C command to get/set bits
- ... adding external VHDL code
- dramatic speedup here!

Part 2: state symmetrization - comperator



3LUs per Bit for a 2-word comparator

Part 2: state symmetrization

using Impulse-C + HDL code to generate Hilbert space/matrix

- test/get minimum of 160 spin permutations
- x86: 600 clocks (test minimum), FPGA: 1 clock
- x86: 6000 clocks (get minimum)

```
-- HDL code (permute000 ... 159, auto generated)
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;
```

Part 2: state symmetrization

using Impulse-C + HDL code to generate Hilbert space/matrix

```
// -- Impulse-C code (auto generated)
#pragma 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_0_000, tmp_0_002);
tmp_2_004 = MinCfg(tmp_0_004, tmp_0_006);
...
... // log2(Nsym) levels
...
tmp_8_000 = MinCfg(tmp_7_000, tmp_7_128);
outcfg = tmp_8_000;
```

Part 2: state symmetrization

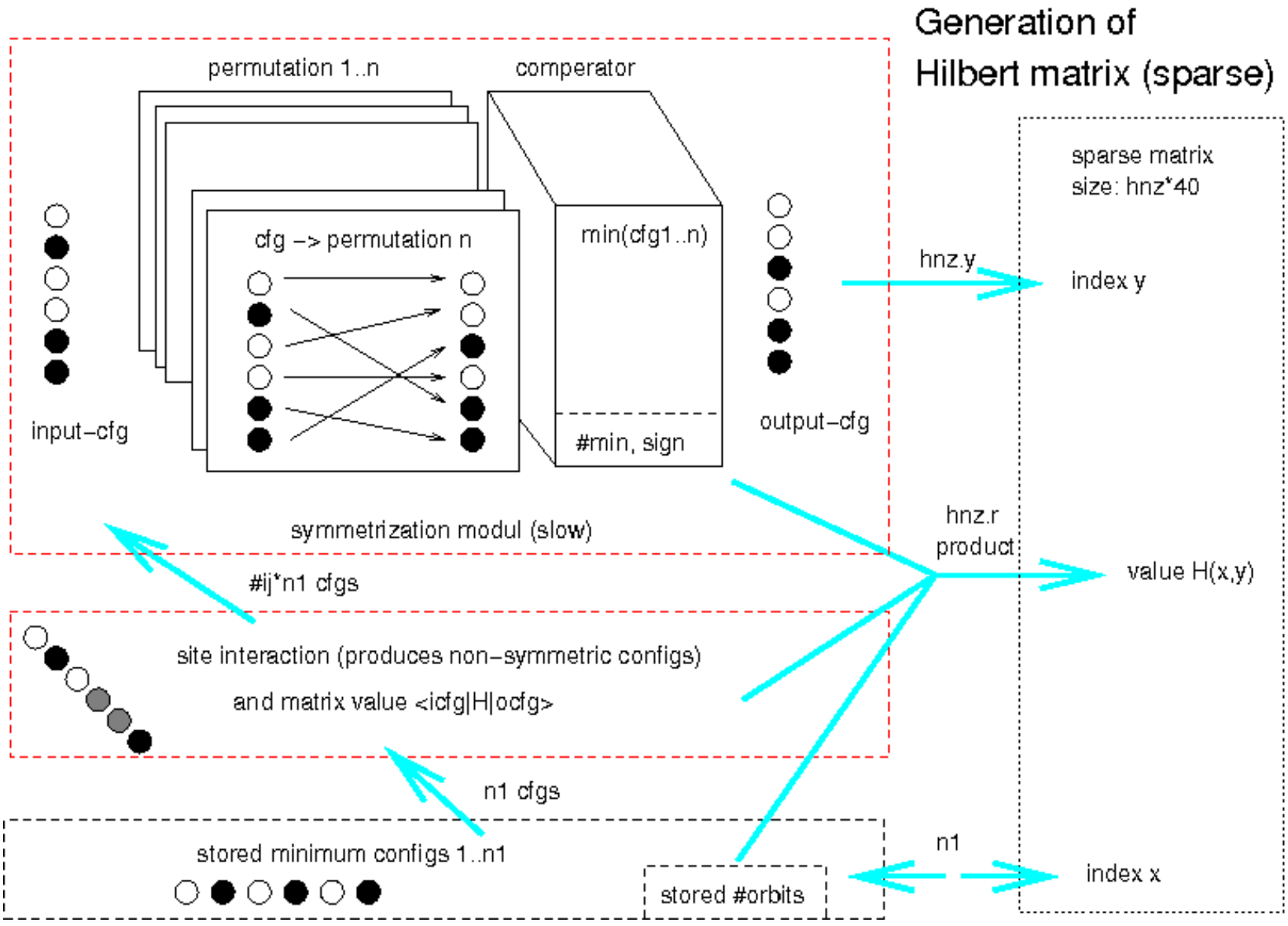
using Impulse-C + HDL code to generate Hilbert space/matrix

- get minimum of 160 spin permutations
- x86: 6000 clocks, FPGA: 1 clock
- 25% of a FPGA chip does this 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 (4..9GB)
- matrix size = 41.8 * 431e6 (160GB)
- FPGA: no need to store matrix, but about same speed
- ... may replace a cluster (less power consumption)

ToDo: Hilbert matrix generation



Conclusion

- excellent performance improvements (especially for bit operations)
- acceptable programming effort (C like, VHDL, needs extensions)
- more possibilities (any accuracy, fault tolerance, ...)
- hardware becomes more attractive
- next step: put FPGA support to SPINPACK, benchmarks
- possible steps: implement operators on FPGA