A Case Study in Using OpenCL on FPGAs: Creating an Open-Source Accelerator of the AutoDock Molecular Docking Software

Fifth International Workshop on FPGA for Software Programmers
FSP 2018
August 31, 2018, Dublin, Ireland

Leonardo Solis-Vasquez Andreas Koch

Technische Universität Darmstadt

Embedded Systems & Applications

Introduction

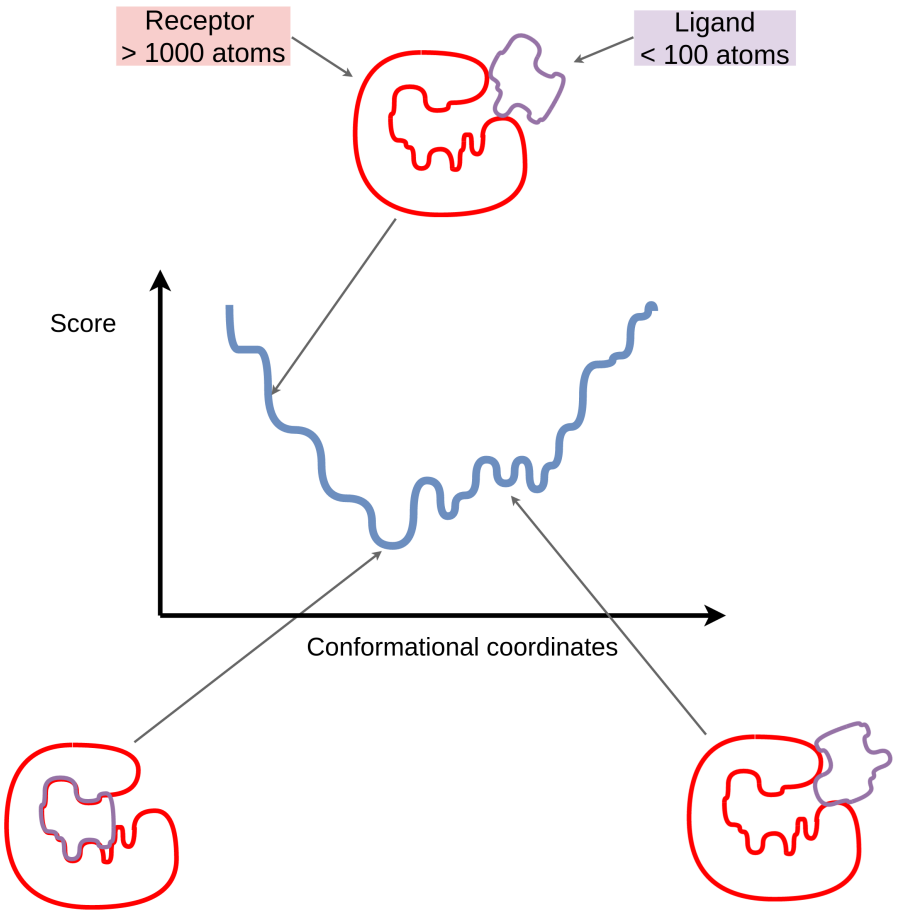
TECHNISCHE UNIVERSITÄT DARMSTADT

- Current status of OpenCL for FPGAs
  - Performance
    - ⋆ In some cases: perf. with OpenCL is drastically lower than with HDLs
  - Application complexity
    - ⋆ Some: code fragments
    - ⋆ Others: fairly regular computation & communication patterns
- A more realistic case study:
  - Molecular Docking (MD): AutoDock
  - Several applications, e.g.: structure-based drug design
  - Performs time-consuming calculations & search methods

Key aspects of molecular docking

TECHNISCHE UNIVERSITÄT DARMSTADT

MD aims to predict the best ways two molecules will interact



- Representation
  - Encoding of docking problem
  - E.g.: translation, rotation, torsion
- Scoring function
  - Energy of a particular pose
  - Lower binding energy is better
- Search methods
  - Finding an optimal pose
  - Which methods should be used?

AutoDock 4.2 (AD4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- AutoDock[1] is one of the most cited[2] molecular docking tools
- Main features
  - Binding encodings: entities of a population
  - Offspring entities are generated through LGA: **Lamarckian Genetic Algorithm**
  - LGA = Global Search (GS) + Local Search (LS)
    - ★ GS: entire population updated through a genetic algorithm: *crossover*, *mutation*, *selection*
    - ★ LS: new entities from population subset (default 6%) are generated using small random arithmetic variations
  - Scoring of binding positions: binding energy

[1] http://autodock.scripps.edu

[2] Sousa et al. [2006]

AD4 scoring function (SF)

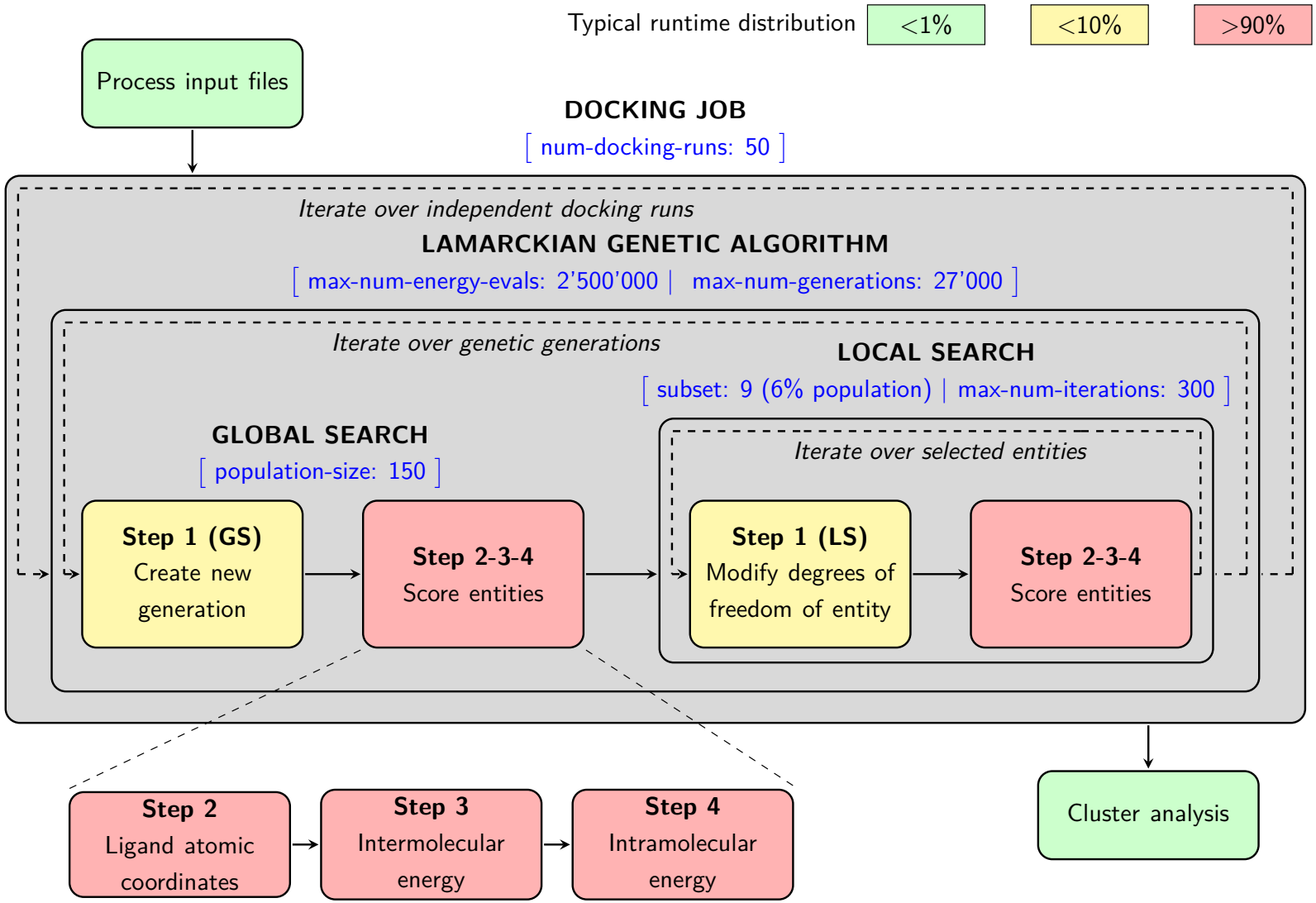
TECHNISCHE UNIVERSITÄT DARMSTADT

- Binding energy (Kcal mol^{-1}) from molecular mechanics
  - Interatomic distance: r_{ij}, between atoms i and j
  - Molecular size (# atoms): receptor >1000, ligand <100

$$SF = \sum_{i,j} \Bigg[\overbrace{W_{vdw}\left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}\right)}^{\textit{Lennard-Jones}} + \overbrace{W_{hb}E(t)\left(\frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}}\right)}^{\textit{Hydrogen bonding}} + \underbrace{W_{el}\left(\frac{q_i q_j}{\epsilon(r_{ij})r_{ij}}\right)}_{\textit{Coulomb's law}} + \underbrace{W_{ds}\left(S_i V_j + S_j V_i\right)e^{\frac{-r_{ij}^2}{2\sigma^2}}}_{\textit{Desolvation}} \Bigg]$$

- Energy components
  - Intramolecular: ligand ↔ ligand (computes SF)
  - Intermolecular: receptor ↔ ligand (replaced with grids)

AD4 algorithm



OpenCL programming models

TECHNISCHE UNIVERSITÄT DARMSTADT

OpenCL work-item: an individual processing thread

Data parallel (NDRange)

```
1  kernel void
2  dp_mul(__global const float *a,
3         __global const float *b,
4         __global float *c)
5  {
6    int id = get_global_id(0);
7
8    c[id] = a[id] * b[id];
9
10 } // Executed over "n" work-items
11
12 // Work-items are grouped
13 // into work-groups
```

Task parallel (single work-item)

```
1  kernel void
2  tp_mul(__global const float *a,
3         __global const float *b,
4         __global float *c,
5         int n)
6  {
7    int i;
8
9    // Loop is pipelined
10   // Might require a #pragma
11   for(i=0; i<n; i++)
12     c[i] = a[i] * b[i];
13 }
```

Previous work on OpenCL AD4

TECHNISCHE UNIVERSITÄT DARMSTADT

- Latest OpenCL version for GPUs[3] → data parallel
  - Docking runs are performed in parallel
  - Each entity is processed by a work-group
  - Fine-grained tasks are performed by work-items
- Speed-up of data-parallel OpenCL wrt. baseline
  - Baseline: single-threaded CPU (original AutoDock)
  - GPU: $\sim$ 55.7x max. speed-up ☺
  - FPGA: $\sim$ 1000x slower! ☹
  - GPU outperforms FPGA in all cases
- Could FPGAs benefit from an OpenCL task-parallel approach?

[3] Solis-Vasquez et al. [2017]

Contributions

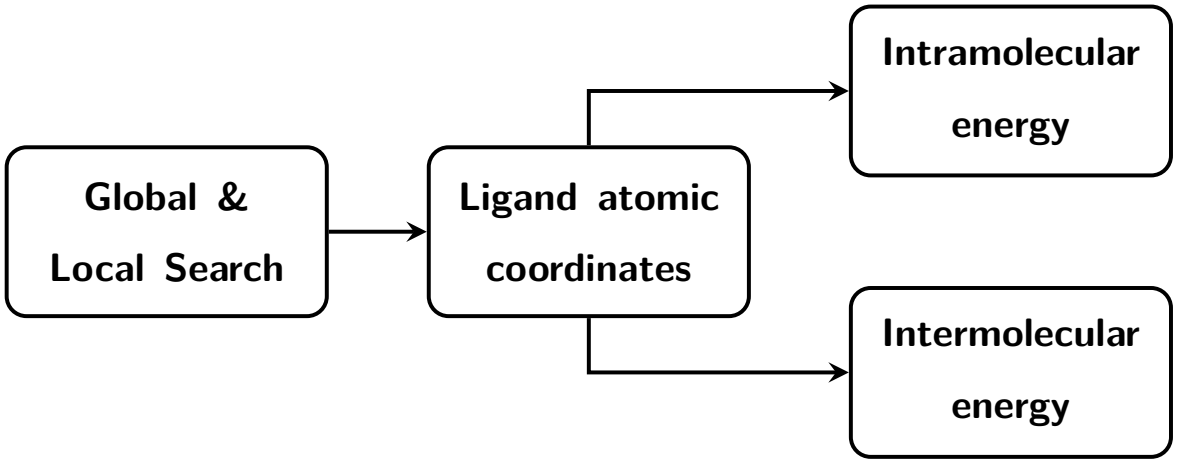
TECHNISCHE UNIVERSITÄT DARMSTADT

1. OpenCL task-parallel implementation of AutoDock
2. Design choices not extensively discussed previously
   - Multiple-producers-to-single consumer datapaths
   - Time-intensive loops with variable runtime
3. The first open-source OpenCL version of AutoDock for FPGAs
   - Performance improvements over single-threaded CPU
   - Maximum speed-up of $\sim$ 2.7x on an Arria-10 FPGA

Starting point

TECHNISCHE UNIVERSITÄT DARMSTADT

Single docking-run architecture[4]



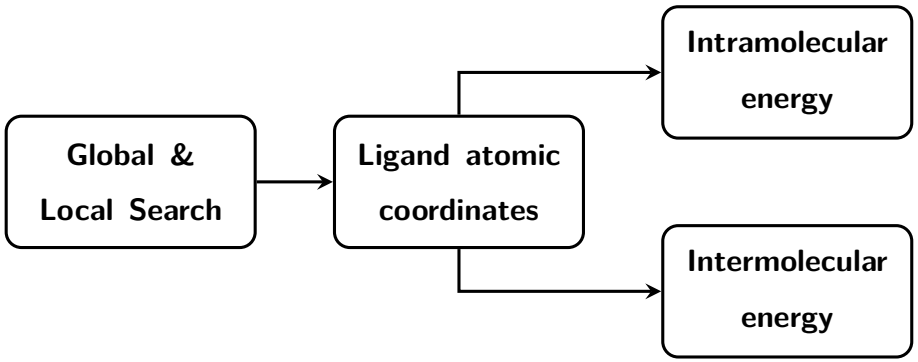
- Existing pipeline architecture for AutoDock
- Docking runs are executed sequentially
  - A new docking run is started after the previous has finished
- Each task is fine-grained pipelined

[4] Pechan et al. [2010]

OpenCL development

TECHNISCHE UNIVERSITÄT DARMSTADT

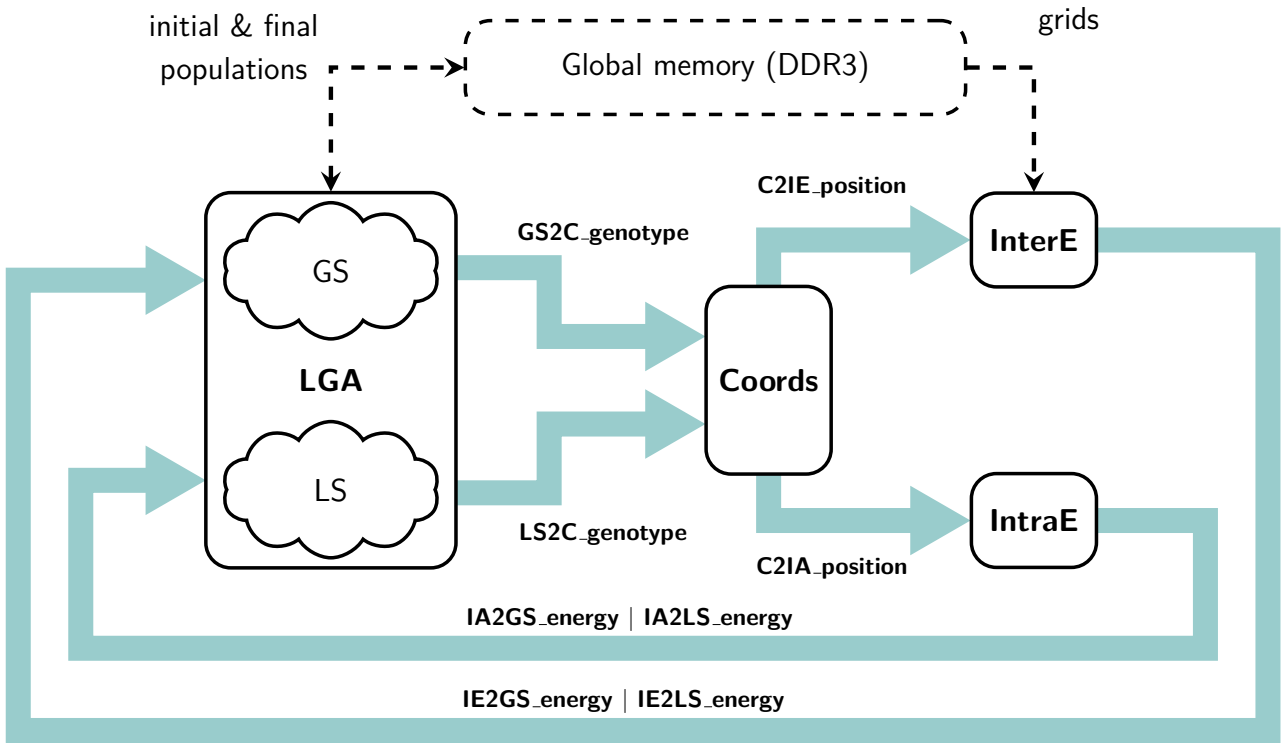
Single docking-run architecture[4]



- Each task coded as a single work-item kernel
- Kernels communicate via OpenCL pipes (= Intel channels)
- General design practices:
  - Pipelining every loop within each kernel
  - Minimizing loops initiation-interval (II)
- Design & optimization steps are organized in phases

[4] Pechan et al. [2010]

First phase: development



- Closed loops (channels $\leftrightarrow$ kernels) prevent channel-depth optimization
- Passing through global memory? $\rightarrow$ worked only in emulation
- Temporal populations stored on-chip: local memory
- Population creation & update in LGA $\rightarrow$ data dependencies

Encountered limitations

TECHNISCHE UNIVERSITÄT DARMSTADT

- *Emulator does not run interacting work-items in parallel* [5]
  - Some concurrent exec. behaviors → inconsistent results
- Such concurrency limitation is critical
  - Heuristic-based search in AD4 can mask errors
- Operations might be rescheduled by the compiler
  - E.g.: rescheduling of channels → deadlocks

```
1  channel uint c0 __attribute__((depth(0)));
2  channel uint c1 __attribute__((depth(0)));
3
4  kernel void
5  producer(__global const uint *src,...) {
6    ...
7    write_channel_intel(c0, src[0]);
8    mem_fence(CLK_CHANNEL_MEM_FENCE);
9    write_channel_intel(c1, src[1]);
10   ...
11 } // Enforce order with fences
```

```
1  // Rd & wr channel calls are BLOCKING
2  kernel void
3  consumer(__global const uint *dst,...) {
4    ...
5    dst[0] = read_channel_intel(c0);
6    mem_fence(CLK_CHANNEL_MEM_FENCE);
7    dst[1] = read_channel_intel(c1);
8    ...
9  } // Enforce order with fences
```

[5] Intel FPGA SDK for OpenCL [2017]

First phase: runtime impact

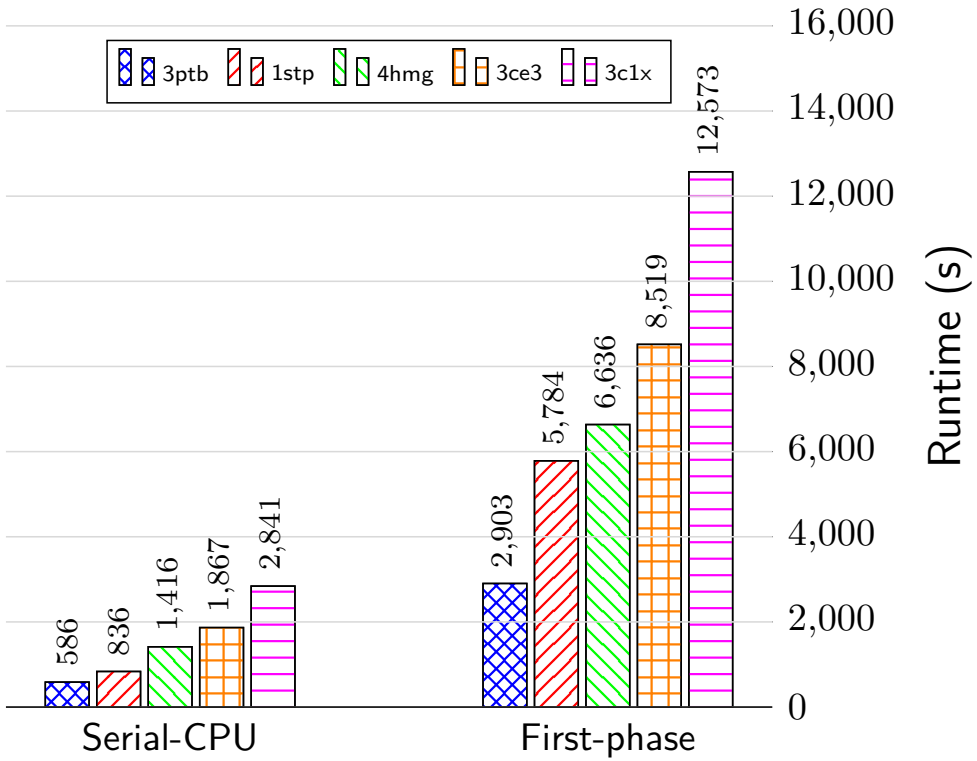
- In all experiments:
  - Default LGA config.[6]
  - 100 docking runs

- Used 5 PDB[7] compounds

Ligand-Receptor					
PDB ID	3ptb	1stp	4hmg	3ce3	3c1x
# Atoms	13	18	27	37	46
# Torsions	2	5	10	5	8

Lower performance ☹

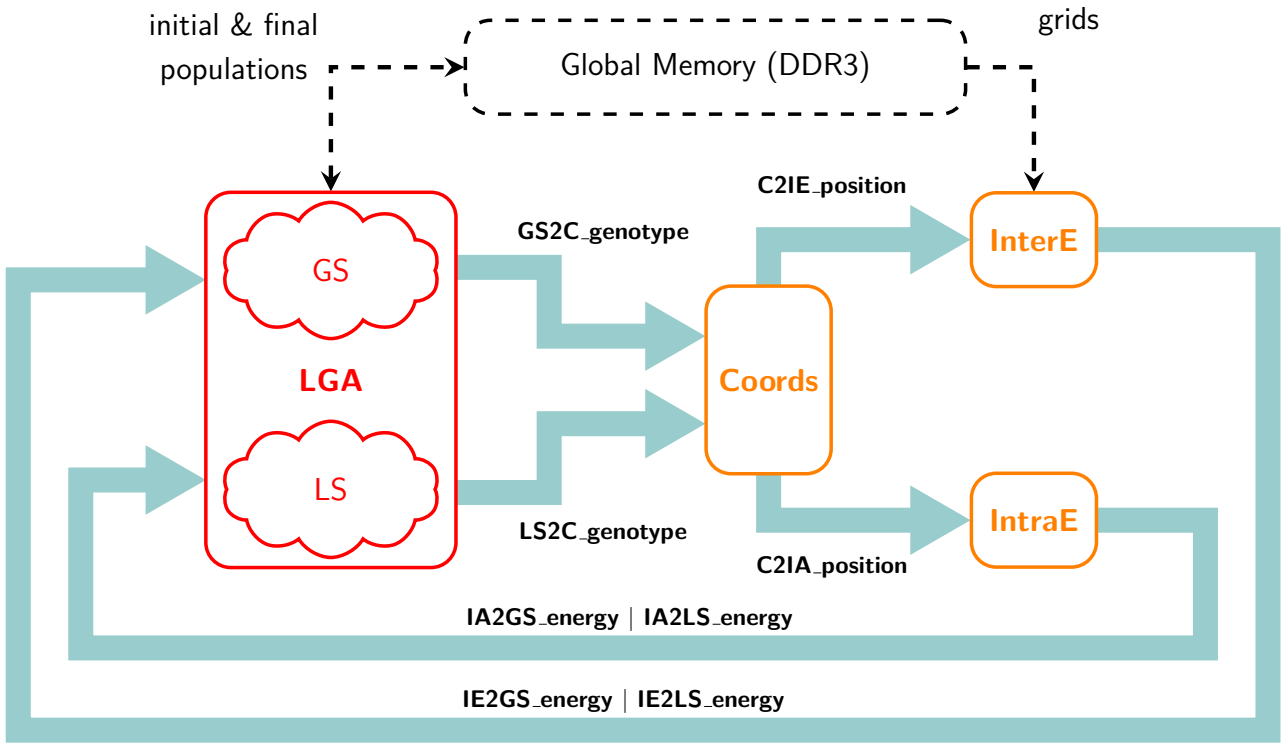
i5-6600K CPU vs. Arria-10 GX 1150 FPGA



[6] AutoDockTools [2012]

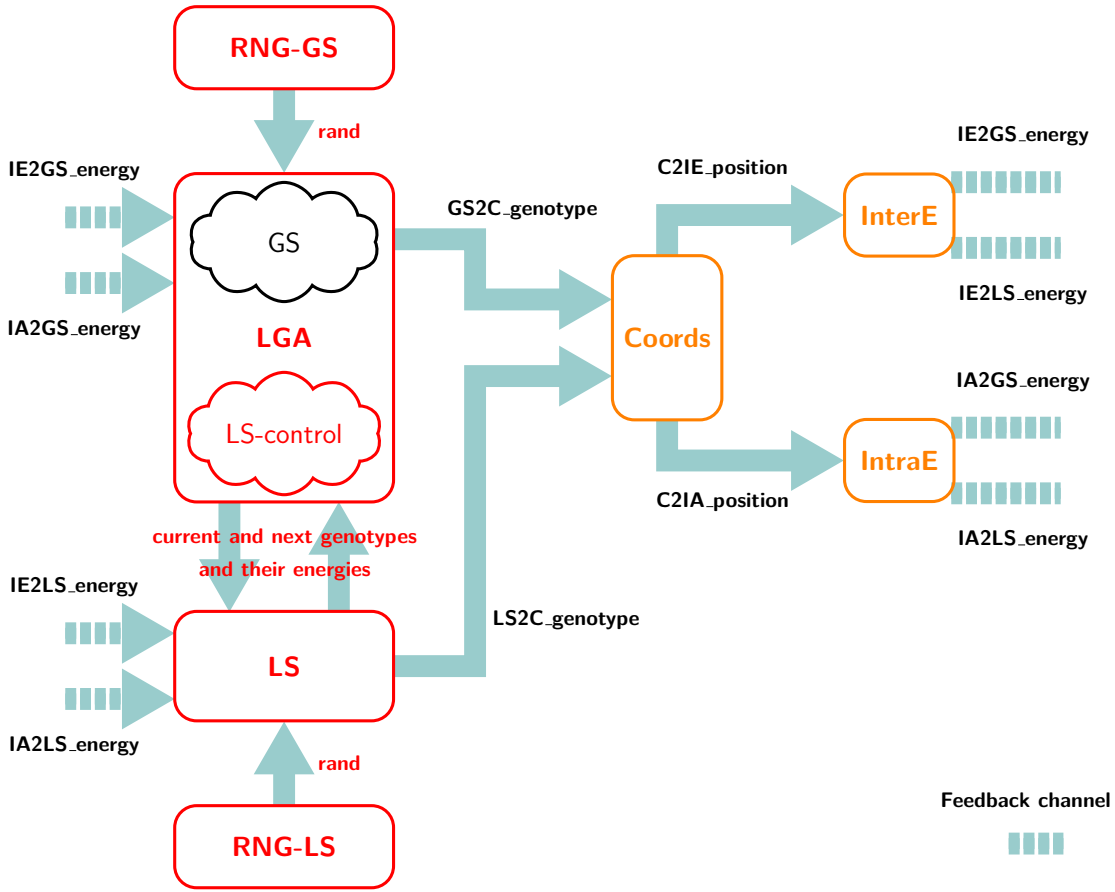
[7] Protein Data Bank archive: https://www.rcsb.org/

First phase: what to optimize next?



- LGA data-dependencies
  - Population creation & update
  - Random numbers in GS and LS
- Bottleneck: energy calc.
  - Coords → InterE → IntraE

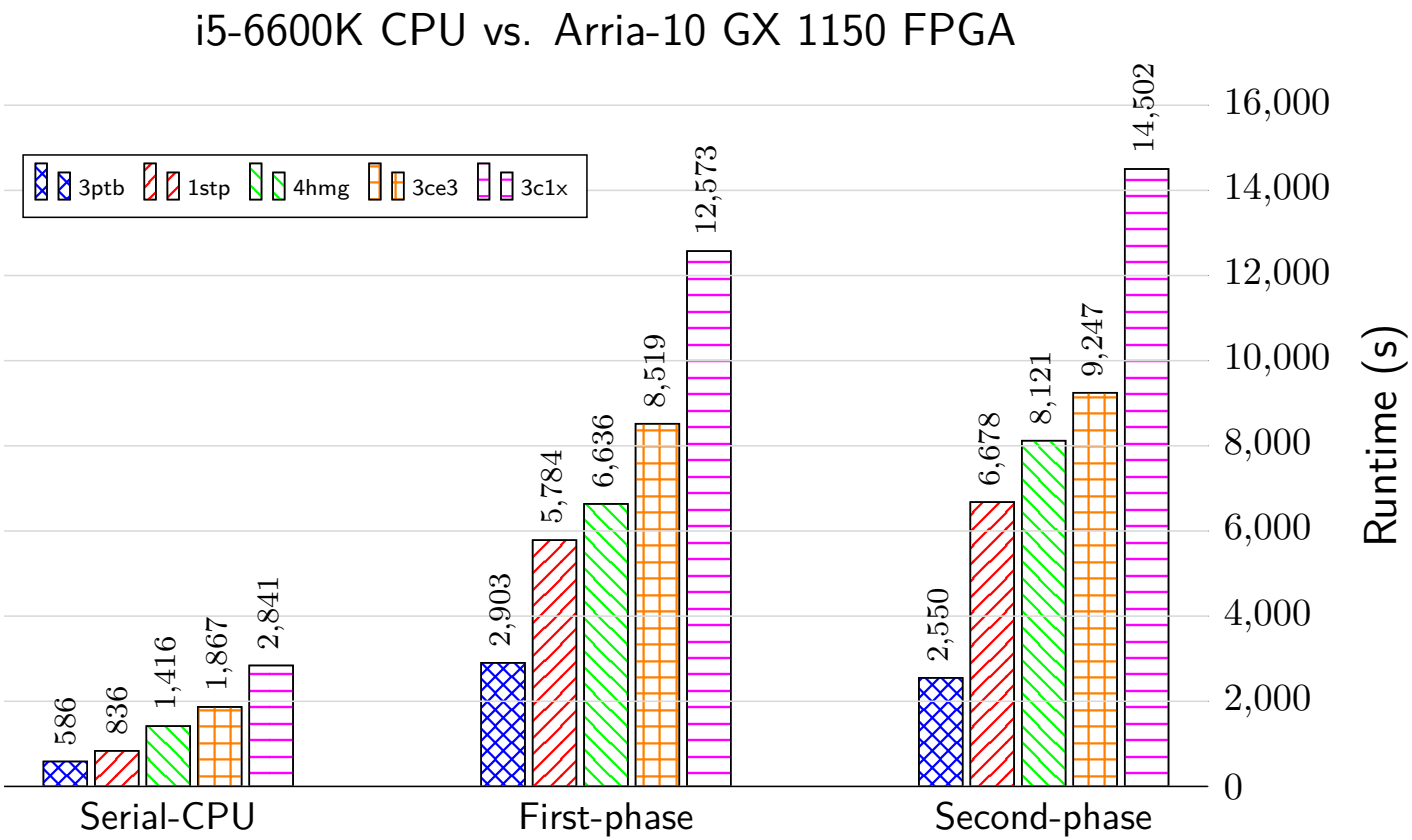
Second phase: development



- LS and RNGs become separate kernels
- Decoupling → increase concurrency

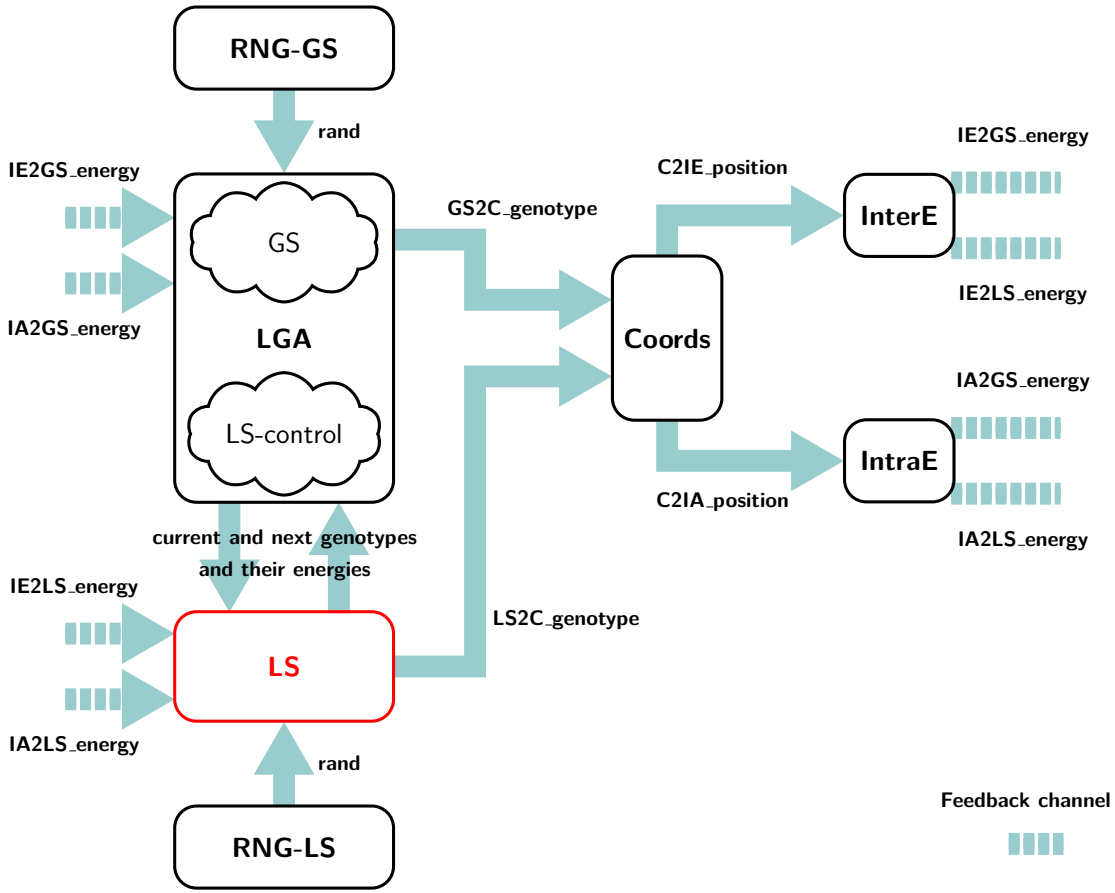
- Coords: data dependency in array of rotated atoms → II = 36
- InterE & IntraE: II = 1

Second phase: runtime impact



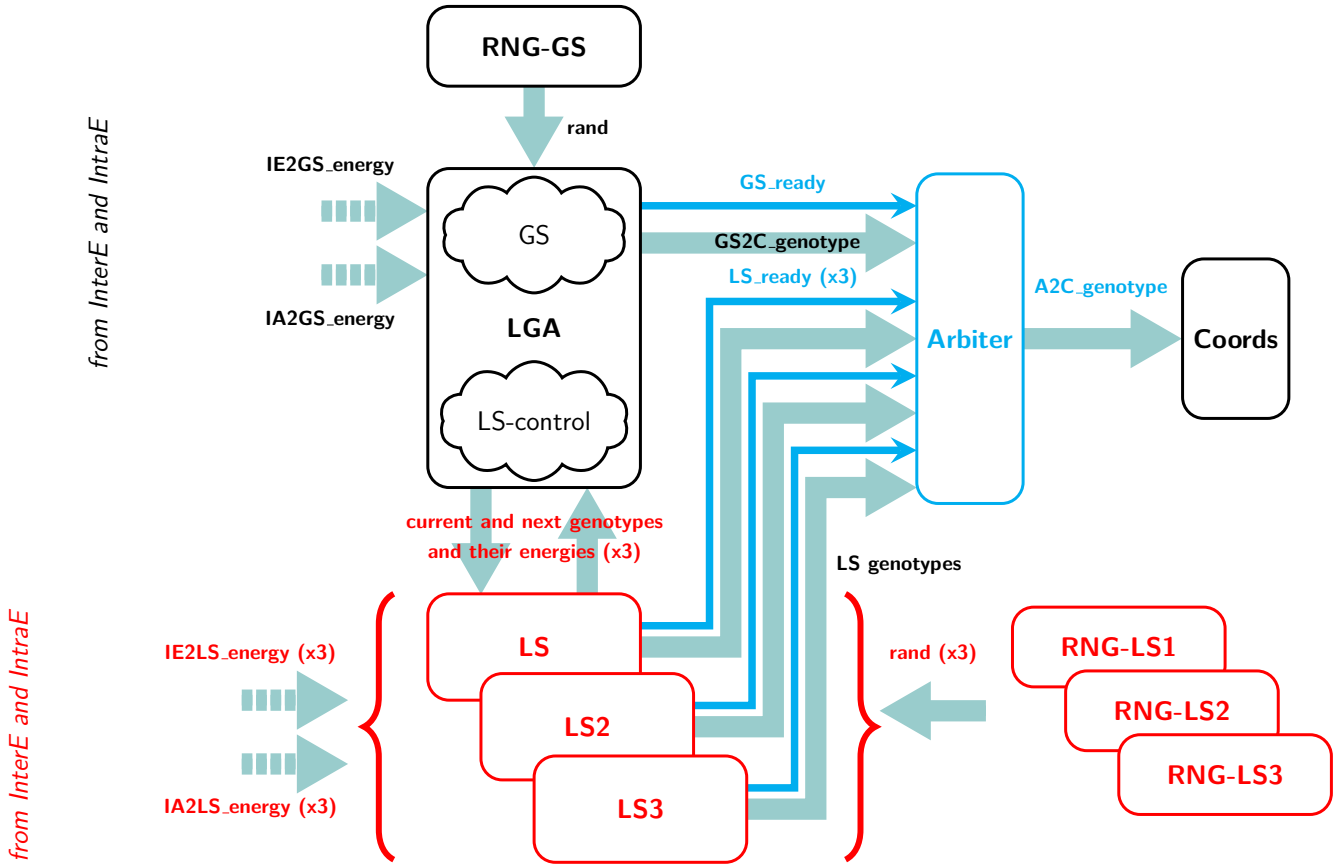
Although bottleneck was optimized, performance became even lower ☹

Second phase: what is next?



- LS: code refactoring for pipelining
- LS-entities are independent $\rightarrow$ exploit LS concurrency!

Third phase: development



- Inner LS loops: most were pipelined
- Outmost LS loop: carried dependency on genotype data → not pipelined
- LS replication → multiple-producers to single-consumer
- Variable # energy evals → ready signals + Arbiter

Allocation of constant data

TECHNISCHE UNIVERSITÄT DARMSTADT

- OpenCL memory-space qualifiers for constant data:
  - `__constant`: on-chip cache, default: 16 KB
  - `__global const`: off-chip, maximum: 16 GB

- If `__constant` args require larger spaces than cache size:
  - Perf. penalties: `__constant` > `__global const`
  - Reason: `__global const` accesses are implemented with extra circuitry → can tolerate longer latencies

- Appropriate allocation in Coords, InterE, IntraE kernels
  - Small arrays: 12 KB → `__constant`
  - Large look-up tables: >270 KB → `__global const`
    - ⋆ E.g.: lists of rotations & intramolecular contributors

Arithmetic representation

TECHNISCHE UNIVERSITÄT DARMSTADT

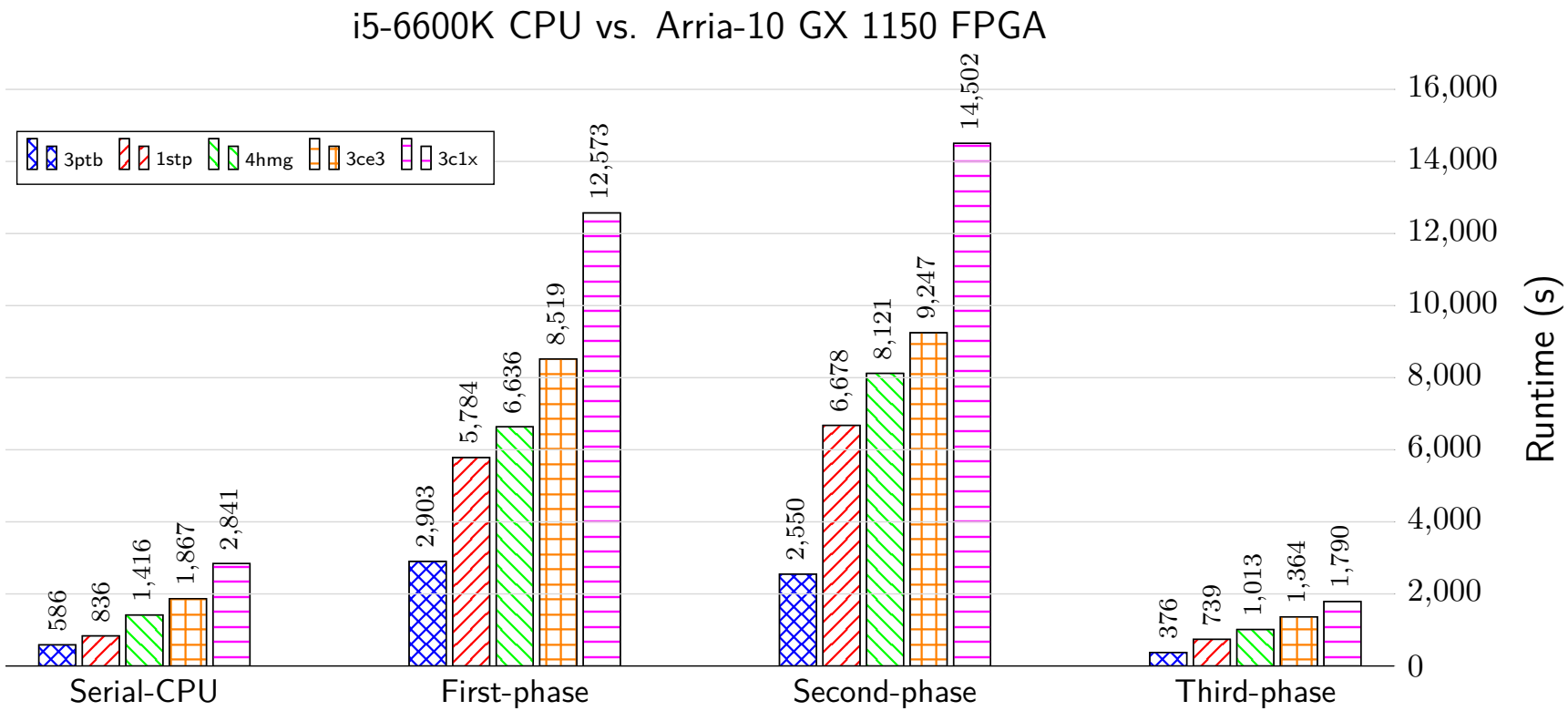
- Arria-10 FPGAs have hardened floating-point DSP units
  - Our design started all in floating-point
- For bottleneck: floating-point or fixed-point?
- Coords kernel
  - Keeps track of rotated atoms → loop-carried dependency
  - Floating-point → pipelined with II=36
  - Fixed-point (16.16) → pipelined with II=10 ☺
- InterE & IntraE kernels
  - Floating-point → pipelined with II=1
  - Fixed-point (32.32) → pipelined with II=1
  - Floating-point results in faster designs 😐

Third phase: runtime impact

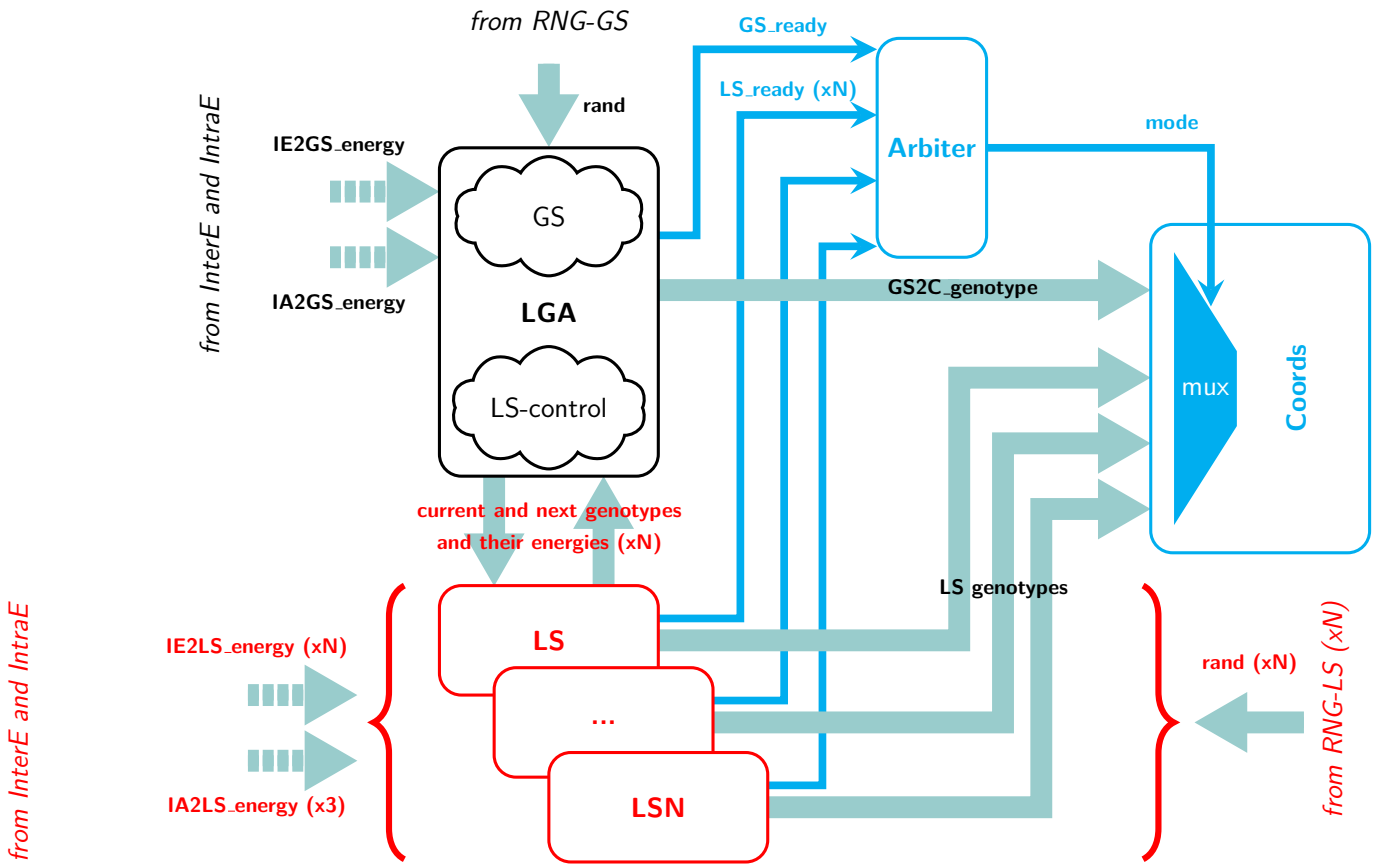
TECHNISCHE UNIVERSITÄT DARMSTADT

Develop. phase	# LS replicas	Arithmetic representation		
		LS	*Coords*	*InterE* *IntraE*
First	1	float	float	float
Second	1	float	float	
Third	3	fixed	fixed	

Speed-up of 1.5x with 3ptb! ☺



Fourth phase: development



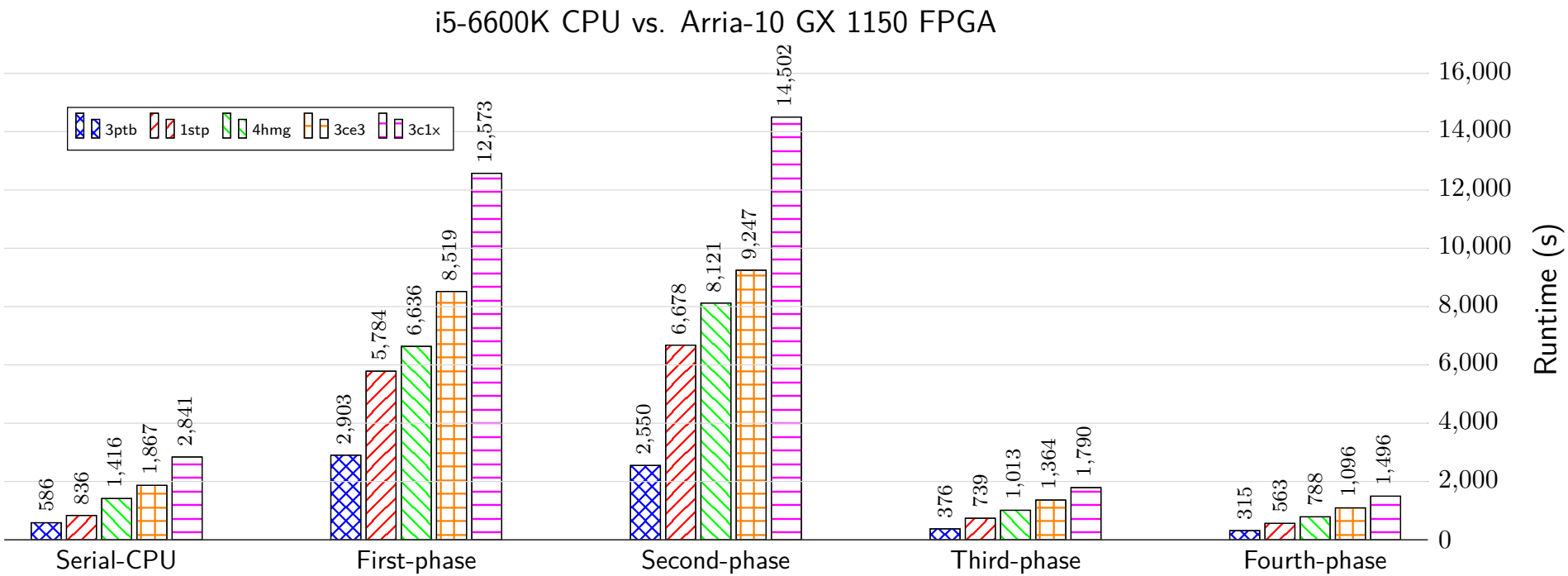
- LS: further replicated
- # LS replicas: based on # LS-entities
- Arbiter: receives only ready signals
- Coords: mux selects genotype

Fourth phase: runtime impact

TECHNISCHE UNIVERSITÄT DARMSTADT

Develop. phase	# LS replicas	Arithmetic representation		
		LS	***Coords***	***InterE IntraE***
First	1	float	float	float
Second	1	float	float	
Third	3	fixed	fixed	
Fourth	5	fixed	fixed	

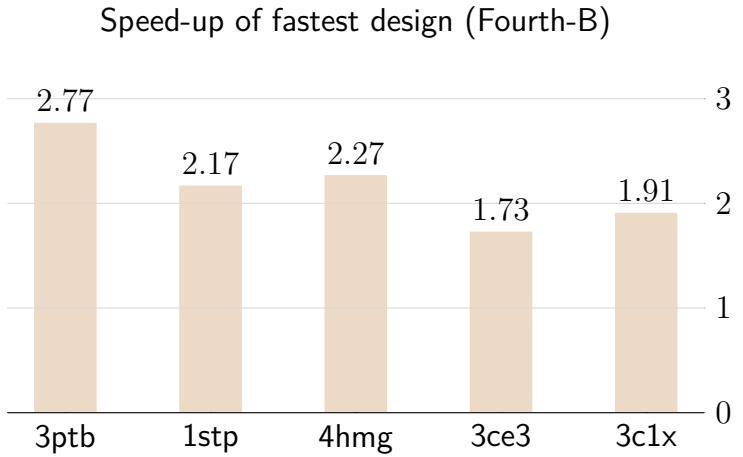
Speed-up of 1.8x with 3ptb! ☺

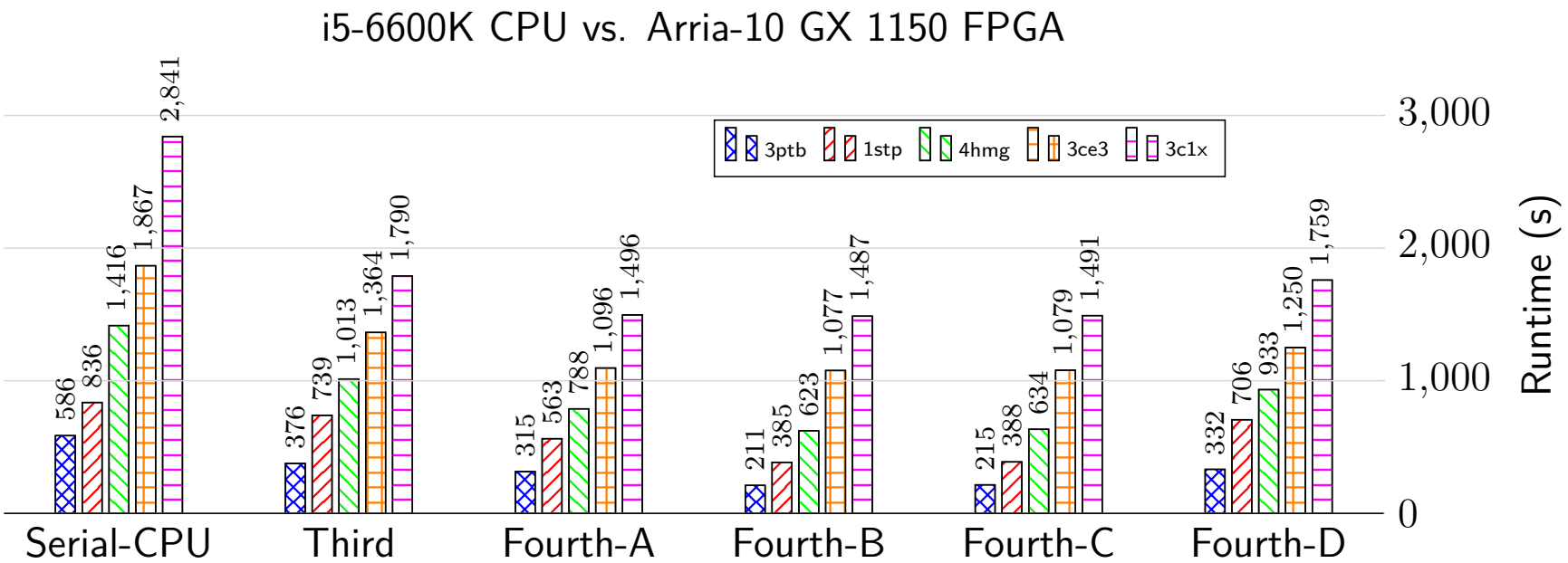


Best performance results

TECHNISCHE UNIVERSITÄT DARMSTADT

Develop. phase	# LS replicas	Arithmetic representation		
		LS	*Coords*	*InterE IntraE*
First	1	float	float	float
Second	1	float	float	
Third	3	fixed	fixed	
Fourth-A	5	fixed	fixed	
Fourth-B	9	fixed	fixed	
Fourth-C	9	float	fixed	
Fourth-D	9	float	float	

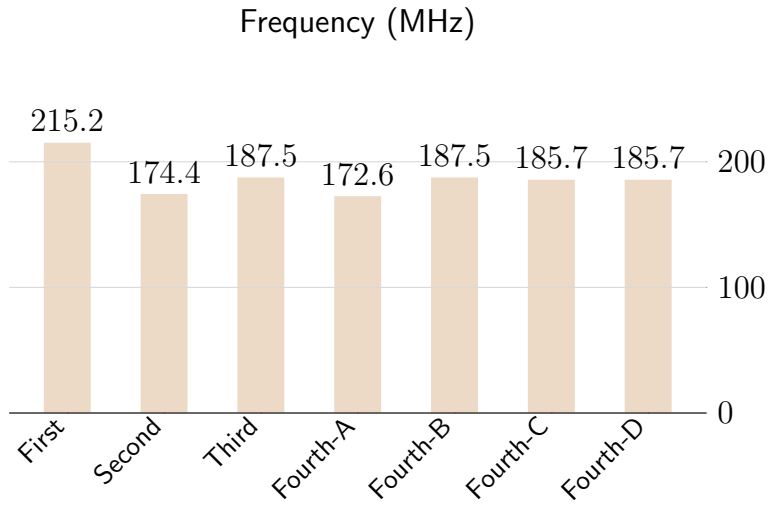


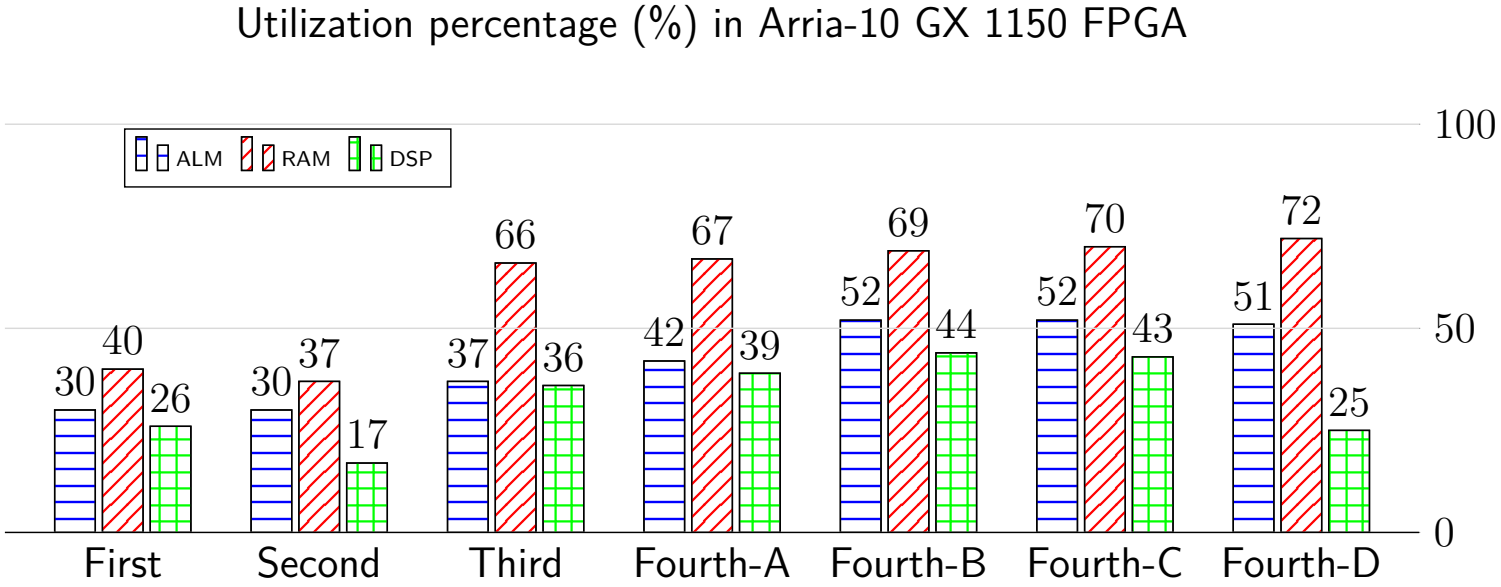


Resource utilization & frequency

TECHNISCHE UNIVERSITÄT DARMSTADT

Develop. phase	# LS replicas	Arithmetic representation		
		LS	***Coords***	***InterE IntraE***
First	1	float	float	float
Second	1	float	float	
Third	3	fixed	fixed	
Fourth-A	5	fixed	fixed	
Fourth-B	9	fixed	fixed	
Fourth-C	9	float	fixed	
Fourth-D	9	float	float	





Challenges for higher performance

TECHNISCHE UNIVERSITÄT DARMSTADT

- Required *awareness* of the underlying hardware
  - For ensuring correctness: explicit synchronization between channel calls using fences (already discussed)
  - For improving II: re-arrangement the local memory layout

```
1  __local int __attribute__((numbanks(4),
2                             bankwidth(4)))
3                             lmem[8][4];
4  #pragma unroll
5  for(int i = 0; i<4; i+=2)
6  {
7    lmem[x][i] = ...;
8  }
```

Recipes for higher performance

TECHNISCHE UNIVERSITÄT DARMSTADT

- AD4 has termination criteria known only at runtime
  - Loop unrolling was possible only in few cases
  - Loop pipelining was much more exploited
- Achieving II=1 is not always possible
  - E.g. : Coords: II = 36
  - Split large sections (LGA) into smaller instances (LS, RNG)
  - Replicate slower data-producer kernels (LS)
- Appropriate allocation of constant data
- Arithmetic representation

Concluding remarks

TECHNISCHE UNIVERSITÄT DARMSTADT

- Performance of previous work: OpenCL data-parallel AutoDock
  - GPUs: $\sim$ 55x speed-up
  - FPGAs: three orders of magnitude slowdown
- This work: design & optimization methodology
  - Task-based parallelization
  - Detailed development phases
  - FPGA-specific techniques using OpenCL
  - Exploration of different architectural choices
- Overall result of our experiments
  - Maximum of $\sim$ 2.7x speed-up on FPGA

A Case Study in Using OpenCL on FPGAs: Creating an Open-Source Accelerator of the AutoDock Molecular Docking Software

https://git.esa.informatik.tu-darmstadt.de/docking/ocladock-fpga

Leonardo Solis-Vasquez
solis@esa.tu-darmstadt.de

https://www.esa.cs.tu-darmstadt.de

Supplementary slides

LGA method - main calls

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  lamarckian_genetic_algorithm {
2    while lga-stop-condition is false {
3      genetic_generation (population); // global
4
5      for entity in random-subset (population) // local
6        local_search(get_genotype (entity));
7
8      update (population);
9    }
10 }
```

Solis-Wets local-search method

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   local_search (genotype) {
2     while ls-stop-condition is false {
3       delta = create_delta (step);
4       newgenotype1 = add_on_every_gene (genotype, delta);
5
6       if (energy (newgenotype1) < energy (genotype))
7         genotype = newgenotype1;
8         success++; fail = 0;
9       else
10        newgenotype2 = sub_on_every_gene (genotype, delta);
11
12        if (energy (newgenotype2) < energy (genotype))
13          genotype = newgenotype2;
14          success++; fail = 0;
15        else
16          success = 0; fail++;
17
18      step = update_step (success, fail);
19    }
20  }
```

Functional validation

TECHNISCHE UNIVERSITÄT DARMSTADT

Comparison of energy & size of best cluster

PDB ID	# Atoms	# Torsions	Energy of best pose (Kcal mol^{-1})		Size of best cluster (100 docking runs)	
			Serial baseline	***OpenCL FPGA***	***Serial baseline***	***OpenCL FPGA***
3ptb	13	2	−5.55	−5.53	100	66
1stp	18	5	−8.37	−7.76	100	69
4hmg	27	10	−3.68	−4.11	34	25
3ce3	37	5	−11.59	−10.88	94	48
3c1x	46	8	−13.61	−12.61	90	22

- Different selection schemes → discrepancies in the cluster size
  - Serial baseline: *proportional selection*
  - OpenCL FPGA: *binary tournament*
- Binary tournament chosen for better performance leads to ...
  - more diverse populations
  - less dense clusters

Resource utilization

TECHNISCHE UNIVERSITÄT DARMSTADT

FPGA resource utilization & frequency

Design config.	**ALMs** Total: 427 200	**RAMs** Total: 2 713	**DSPs** Total: 1 518	**Freq. (MHz)**
DC1	129 301 (30%)	1 075 (40%)	388 (26%)	215.2
DC2	128 018 (30%)	999 (37%)	262 (17%)	174.4
DC3	158 586 (37%)	1 799 (66%)	548 (36%)	187.5
DC4a	177 509 (42%)	1 826 (67%)	586 (39%)	172.6
DC4b	222 372 (52%)	1 880 (69%)	662 (44%)	187.5
DC4c	220 427 (52%)	1 898 (70%)	659 (43%)	185.7
DC4d	219 359 (51%)	1 944 (72%)	383 (25%)	185.7

- Arria-10 GX 1150 FPGA, Gidel Proc10A card, 16 GB RAM
- Latest compiler supported by the BSP:
  Intel FPGA SDK for OpenCL v16.0

Performance results

TECHNISCHE UNIVERSITÄT DARMSTADT

Execution runtime & best speed-up for 100 docking runs

Execution runtime (seconds)					
Design config.	**Ligand-Receptor PDB ID**				
	3ptb	***1stp***	***4hmg***	***3ce3***	***3c1x***
Serial CPU	586	836	1416	1867	2841
DC1	2903	5784	6636	8519	12573
DC2	2550	6678	8121	9247	14502
DC3	376	739	1013	1364	1790
DC4a	315	563	788	1096	1496
DC4b	211	385	623	1077	1487
DC4c	215	388	634	1079	1491
DC4d	332	706	933	1250	1759
Best speed-up					
DC4b	2.77x	2.17x	2.27x	1.73x	1.91x

- i5-6600K single CPU core, at 3.5 GHz, 16 GB RAM
- Arria-10 GX 1150 FPGA, Gidel Proc10A card, 16 GB RAM

Energy-efficiency results

TECHNISCHE UNIVERSITÄT DARMSTADT

Energy consumption & best energy-efficiency gains for 100 docking runs

Energy consumption (KJ)					
Design config.	**Ligand-Receptor PDB ID**				
	3ptb	***1stp***	***4hmg***	***3ce3***	***3c1x***
Serial CPU	11.80	16.69	28.07	36.27	54.85
DC4b	6.33	11.50	18.69	32.31	44.61
Best energy improvement					
DC4b	1.86x	1.45x	1.50x	1.12x	1.23x

- i5-6600K single CPU core
  - Sampling interval of 50 ms, using power perf. counters
  - Power samples were integrated over time to derive energy
- Arria-10 GX 1150 FPGA
  - Power values from fully placed & routed designs: *quartus_pow*
  - Estimation of $\sim$ 30 W was multiplied by the respective runtime

Comparison with a HDL design

TECHNISCHE UNIVERSITÄT DARMSTADT

- AutoDock accelerator designed in Verilog[8]
  - Similar pipelining architecture
  - No LS-kernel replication
  - Runtimes of a single docking-run were reported
- For smaller PDB molecules: runtimes are similar
  - OpenCL: 3ptb: $\sim$ 2.1 s, 1stp: $\sim$ 3.8 s
  - Verilog: average of 60 PDBs: $\sim$ 3.1 s
- For larger PDB molecules:
  - OpenCL runtimes become longer than the $\sim$ 3.1 s average
  - Verilog does not appear to suffer from this
    - ⋆ (Most likely) due to finer manually-controlled pipelining

[8] Pechan et al. [2010]

Tool support

TECHNISCHE UNIVERSITÄT DARMSTADT

- Design specification & emulation-based verification
  - As easy as with GPUs
  - Optimization reports: to assess impact of code modifications
    - Initiation interval (II)
    - Estimated resource-utilization
- Hardware validation & subsequent optimization cycles
  - Much more involved
  - Hardware profiler: to pinpoint bottlenecks like:
    - Unbalanced communication traffic between producer (GS, LS) & consumer (Arbiter) kernels
    - Inefficient memory accesses

Challenges for higher performance

TECHNISCHE UNIVERSITÄT DARMSTADT

- Required *awareness* of the underlying hardware
  - For improving II: code-refactoring using hardware constructs

```
1  // Unoptimized accumulation
2
3  // Array a[] has N elements
4
5  double temp_sum = 0;
6
7  // II = 11
8  for (int i = 0; i < N; ++i) {
9    temp_sum += arr[i];
10 }
11
12 *result = temp_sum;
```

```
1  // Optimized accumulation using shift register
2
3  // Create shift reg.
4  #define II_CYCLES 12
5  double shift_reg[II_CYCLES+1];
6
7  // Initialize all reg. elements to 0
8  ...
9
10 // II = 1
11 // Load a[i] into end of shift reg.
12 for(int i = 0; i < N; ++i) {
13   // if N > II_CYCLE,
14   // add to shift_reg[0] to preserve values
15   shift_reg[II_CYCLES] = shift_reg[0] + arr[i];
16
17   #pragma unroll
18   // Shift every element of shift reg.
19   for(int j = 0; j < II_CYCLES; ++j) {
20     shift_reg[j] = shift_reg[j + 1];
21   }
22 }
23 ...
```

Mapping of OpenCL constructs into hardware

TECHNISCHE UNIVERSITÄT DARMSTADT

- Fences for local-memory within single work-item kernels
  - Standard says they guard access to data qualified with `__local`
  - Actually they protect the access to data stored in block RAM
  - Either OpenCL private or `__local` arrays can be mapped to block RAM
    - ⋆ As long as their size >= 64 bytes
  - Such fences used in Coords kernel
    - ⋆ For computing correctly atomic positions (loop-carried dependency)

Productivity

TECHNISCHE UNIVERSITÄT DARMSTADT

- Smallest design: first phase $\rightarrow$ 4 kernels
  - LGA, Coords, InterE, & IntraE
- Correctly emulated design for the first phase: $\sim$ 4 weeks
- Largest designs: fourth phase {B, C, D} $\rightarrow$ 27 kernels each
  - LGA, Coords, InterE, IntraE
  - Arbiter, four RNG-GS[9], nine LS, & nine RNG-LS
- Completion of whole development: $\sim$ 5 months
  - Delays caused by emulator limitations
  - Large FPGA synthesis and mapping times
    - $\sim$ 8 hours for each of our largest designs

[9] Four different genetic ops: crossover, mutation, GS & LS selection