IA^3 2020
10th Workshop on Irregular Applications: Architectures and Algorithms
November 11, 2020

Parallelizing Irregular Computations for Molecular Docking

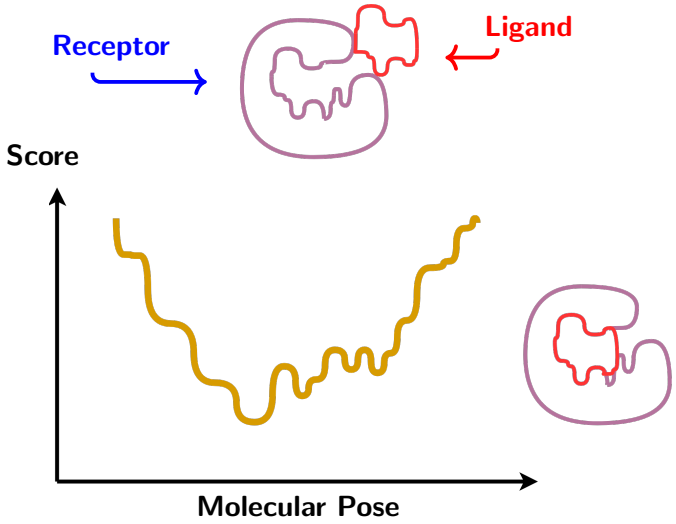
Leonardo Solis-Vasquez [*], Diogo Santos-Martins [+], Andreas F. Tillack [+],

Andreas Koch [*], Jérôme Eberhardt [+], Stefano Forli [+]

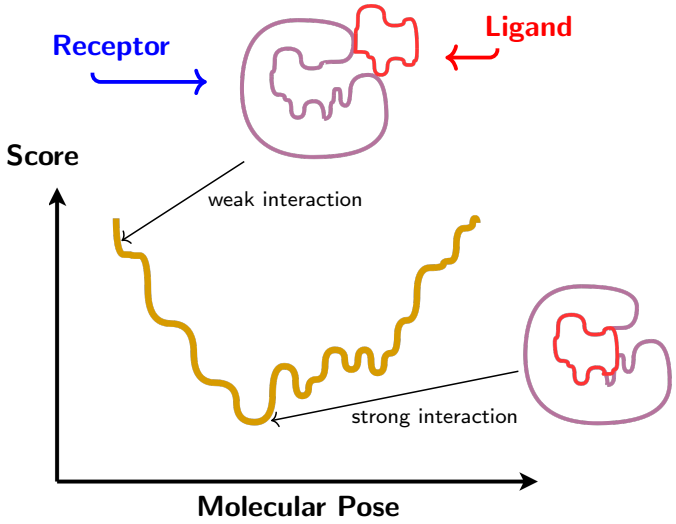
[*] Embedded Systems and Applications Group
Technical University of Darmstadt, Germany

[+] Center for Computational Structural Biology
The Scripps Research Institute, United States

Molecular Docking

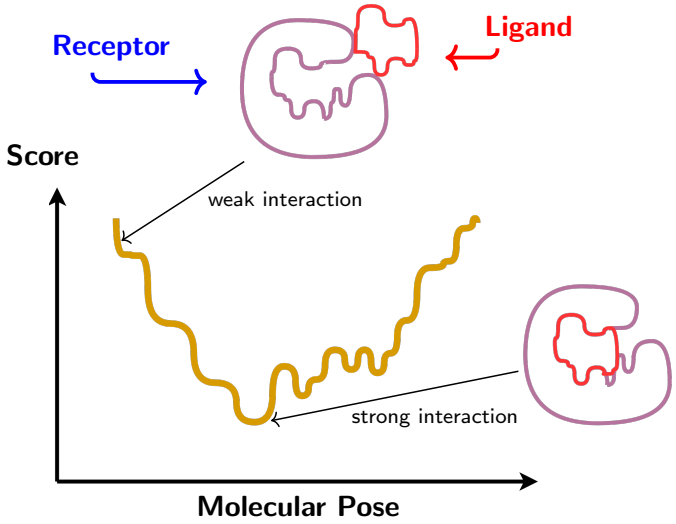


Molecular Docking



- Molecular docking aims to find poses of strong interaction
- Scoring function
  - Measures how strong a pose is

Molecular Docking



- Molecular docking aims to find poses of strong interaction
- Scoring function
  - Measures how strong a pose is
- Representation
  - Encodes a pose in terms of e.g., translation, rotation, torsion
- Search methods
  - Finds an optimal pose

AutoDock

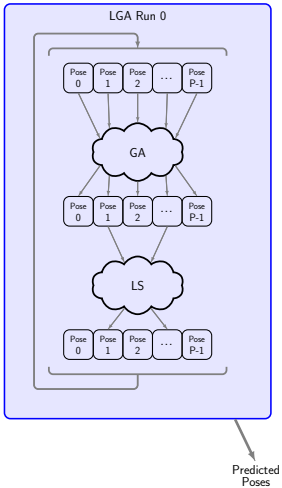
- One of the most cited docking tools
  - E.g., *FightAIDS@Home* project

AutoDock

- One of the most cited docking tools
  - E.g., *FightAIDS@Home* project
- Lamarckian Genetic Algorithm (LGA)
- LGA = GA + LS
  - Genetic Algorithm (GA)
  - Local Search (LS)
  - Both perform score calculations

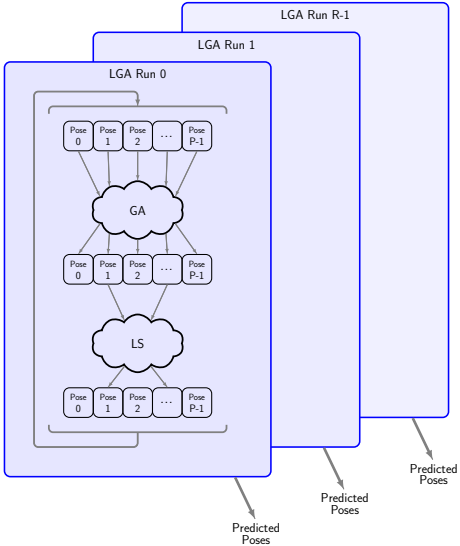
AutoDock

- One of the most cited docking tools
  - E.g., *FightAIDS@Home* project
- Lamarckian Genetic Algorithm (LGA)
- LGA = GA + LS
  - Genetic Algorithm (GA)
  - Local Search (LS)
  - Both perform score calculations



AutoDock

- One of the most cited docking tools
  - E.g., *FightAIDS@Home* project
- Lamarckian Genetic Algorithm (LGA)
- LGA = GA + LS
  - Genetic Algorithm (GA)
  - Local Search (LS)
  - Both perform score calculations



Scoring Function

- Binding energy (Kcal mol^{-1}) from molecular mechanics
  - Molecular size
    - $N_{\text{atom}}^{\text{Receptor}} > 1000$
    - $N_{\text{atom}}^{\text{Ligand}} < 100$
  - Dimensionless coefficients
    - W_{vdw}, W_{hb}, W_{el}, W_{ds}, W_{rot}
  - Look-up tables
    - A, B, C, D, S, V, E, q
  - Interatomic distance r_{ij}
    - Between atoms i and j

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

Lamarckian Genetic Algorithm

- Termination criteria
  - User defined
  - $N^{\mathrm{MAX}}_{\mathrm{score-evals}} = 2\,048\,000$
  - $N^{\mathrm{MAX}}_{\mathrm{gens}} = 27\,000$
- Nested loops
  - With variable upper bounds
  - Time-intensive score evals

```
Function AutoDock
    /* Coarse-Level Parallelism */
1   for each LGA-run do
2       while (N_score-evals < N^MAX_score-evals) and (N_gens < N^MAX_gens)
          do
            /* Medium-Level Parallelism */
3           GA (population)
            /* Medium-Level Parallelism */
4           for individual in random-subset (population) do
5             LS (get-genotype (individual))
```

Local Search: Solis-Wets

- Termination criteria
  - User defined
  - $N_{\mathrm{LS-iters}}^{\mathrm{MAX}} = 300$
  - $\mathrm{step}^{\mathrm{MIN}} = 0{,}01$
- Nested loops
  - With variable upper bounds
  - Time-intensive score evals
- Divergent control
  - Score improves → *success*
  - Score diminishes → *failure*

```
1  /* Fine-Level Parallelism */
2  Function Solis-Wets (genotype)
3      while (N_LS-iters < N_LS-iters^MAX) and (step > step^MIN) do
4          delta = create-delta (step)
5          // new-genotype1
6          for each gene in N_genes do
7              new-gene1 = gene + delta
8          if SF (new-genotype1) < SF (genotype) then
9              genotype = new-genotype1
10             success++; fail = 0
11         else
12             // new-genotype2
13             for each gene in N_genes do
14                 new-gene2 = gene - delta
15             if SF (new-genotype2) < SF (genotype) then
16                 genotype = new-genotype2
17                 success++; fail = 0
18             else
19                 success = 0; fail++
20         step = update-step (success, fail)
```

Our Contribution

- Previous work: OpenCL port of AutoDock
  - AutoDock-GPU
    - ★ Evaluated overall compute performance
    - ★ Focus: molecular prediction quality

Our Contribution

- Previous work: OpenCL port of AutoDock
  - AutoDock-GPU
    - ⋆ Evaluated overall compute performance
    - ⋆ Focus: molecular prediction quality
- Here: parallelization in AutoDock-GPU
  - Focus: development rather than domain-oriented perspective
  - Challenges of dealing with AutoDock irregularity
  - Analysis of impact on execution runtime on GPUs/CPUs
    - ⋆ OpenCL work-groups configuration
    - ⋆ Molecular complexity of different inputs
  - Experiences porting onto FPGAs

Design Considerations for Host Code (1/2)

- AutoDock coded having only functionality in mind
  - I/O and compute tasks intertwine *unnecessarily*
    - Read configuration options
    - Perform computation (search and score calculation)
    - Write partial results (predicted poses)
    - Repeat (until all LGAs are processed)

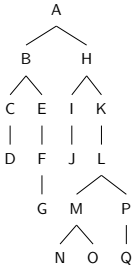
Design Considerations for Host Code (1/2)

- AutoDock coded having only functionality in mind
  - I/O and compute tasks intertwine *unnecessarily*
    - Read configuration options
    - Perform computation (search and score calculation)
    - Write partial results (predicted poses)
    - Repeat (until all LGAs are processed)

- AutoDock-GPU re-structures program
  - Into a parallel-friendly version
  - I/O and compute tasks are decoupled *completely*
  - Exposes the Local Search function
    - As the most runtime consuming
    - Comprising several score evaluations

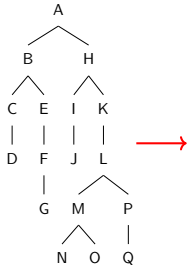
Design Considerations for Host Code (2/2)

- Rotatable bonds (torsions) affect ...
  - Interatomic distances $\rightarrow$ interactions
- AutoDock
  - Builds a tree of torsion-affected atoms
  - Recursively traverses that tree
    - ★ Calculates score at every node



Design Considerations for Host Code (2/2)

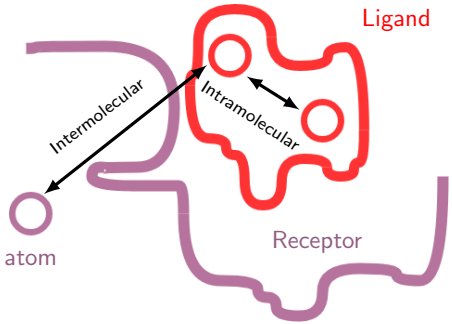
- Rotatable bonds (torsions) affect ...
  - Interatomic distances → interactions
- AutoDock
  - Builds a tree of torsion-affected atoms
  - Recursively traverses that tree
    - ★ Calculates score at every node
- AutoDock-GPU transforms data *on host*
  - Tree → arrays
    - ★ Recursion → iteration
  - More efficient on-device processing



ID	[Atom-to-Rotate | Rotation-Type]
1	Rotation element 1
2	Rotation element 2
3	Rotation element 3
...	...
...	...
...	...
...	...
...	...
N	Rotation element N

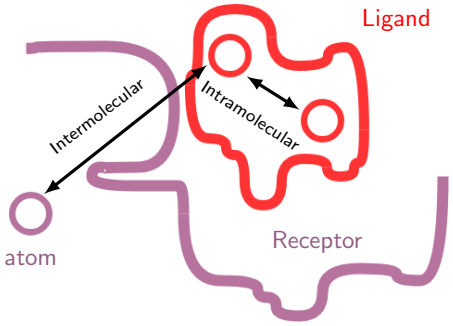
Re-designing Scoring Function (1/2)

- Scoring function has two components
  - Intermolecular
    - ★ receptor atoms ↔ ligand atoms
  - Intramolecular
    - ★ ligand atoms ↔ ligand atoms



Re-designing Scoring Function (1/2)

- Scoring function has two components
  - Intermolecular
    - ★ receptor atoms ↔ ligand atoms
  - Intramolecular
    - ★ ligand atoms ↔ ligand atoms
- AutoDock processes *pre-calculated* interactions
  - Purpose: reducing execution times
  - Pre-calculation takes place before AutoDock execution
  - Loop-up tables are accessed during docking



Re-designing Scoring Function (2/2)

- AutoDock-GPU re-designs scoring
  - Pose calculation → integrated into scoring
  - Intermolecular → pre-calculated (still!)
  - Intramolecular
    - ★ Performs actual computations instead of pre-calculating
    - ★ $N_{\text{atom}}^{\text{Ligand}} < 100$ ($<< N_{\text{atom}}^{\text{Receptor}}$)
    - ★ More accurate than pre-calculation
    - ★ Leverages compute power on e.g., GPUs

```
/* Fine-Level Parallelism */
1 Function SF (genotype)
2     for each rot-item in N_pose−rot do
3         PoseCalculation
4     for each lig-atom in N_atom do
5         InterInteraction
6     for each intra-pair in N_intra−contrib do
7         IntraInteraction
```

Re-designing Scoring Function (2/2)

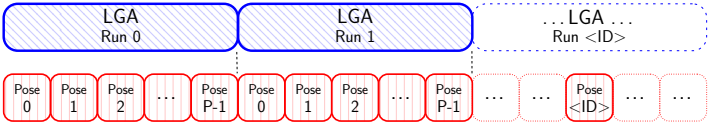
- AutoDock-GPU re-designs scoring
  - Pose calculation → integrated into scoring
  - Intermolecular → pre-calculated (still!)
  - Intramolecular
    - ⋆ Performs actual computations instead of pre-calculating
    - ⋆ $N_{\text{atom}}^{\text{Ligand}} < 100$ ($<< N_{\text{atom}}^{\text{Receptor}}$)
    - ⋆ More accurate than pre-calculation
    - ⋆ Leverages compute power on e.g., GPUs
  - SF leverages fine-level parallelism

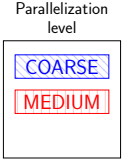
```
/* Fine-Level Parallelism */
1 Function SF (genotype)
2     for each rot-item in N_pose−rot do
3         PoseCalculation
4     for each lig-atom in N_atom do
5         InterInteraction
6     for each intra-pair in N_intra−contrib do
7         IntraInteraction
```

Mapping operations into OpenCL elements

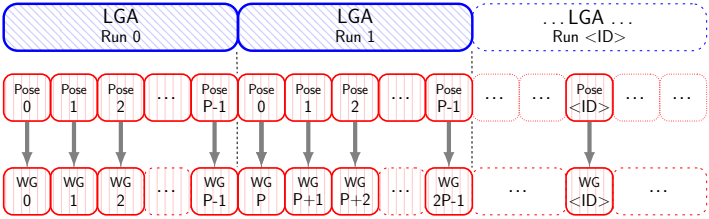
Parallelization level

Mapping operations into OpenCL elements

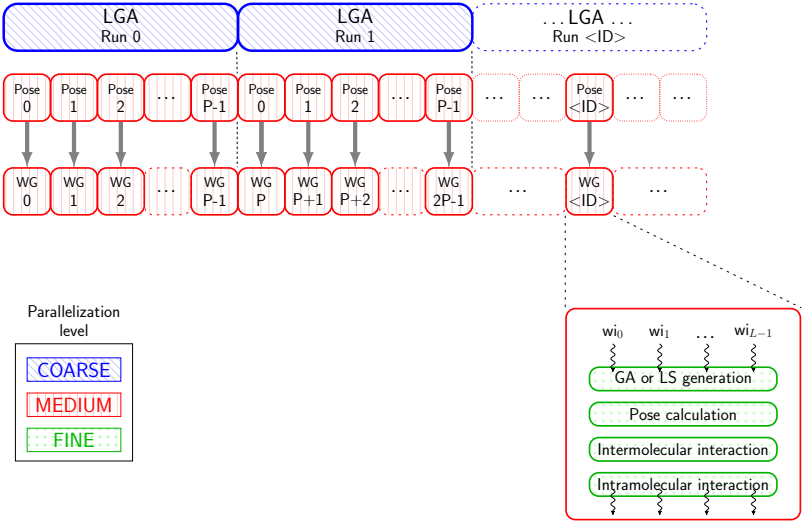




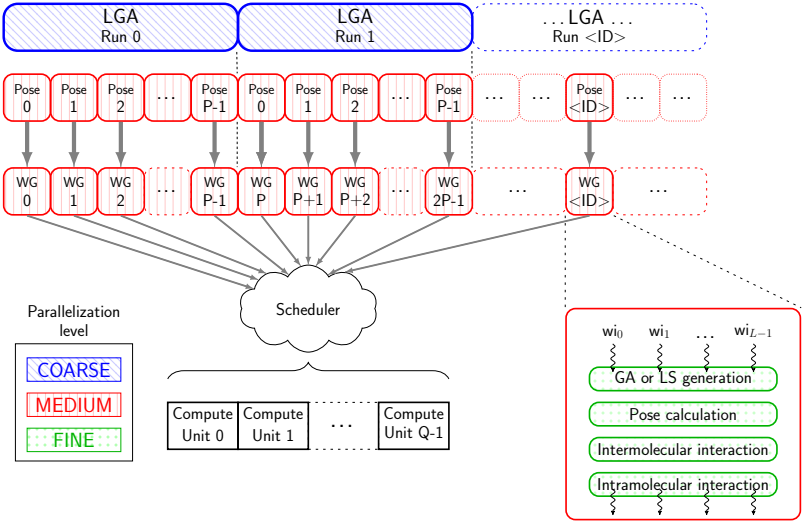
Mapping operations into OpenCL elements



Mapping operations into OpenCL elements

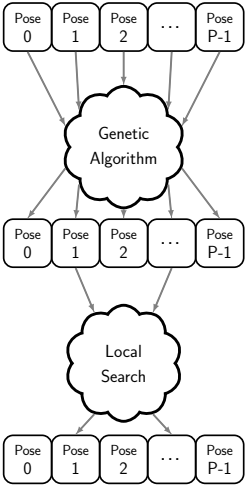


Mapping operations into OpenCL elements



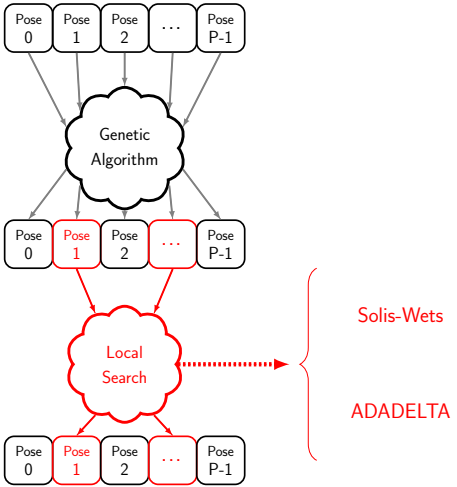
Integrating alternative Local-Search methods

- Local-search refinement
  - Can enhance pose predictions
- More efficient search algorithms
  - May find strong interactions
  - Require more-intensive computations



Integrating alternative Local-Search methods

- Local-search refinement
  - Can enhance pose predictions
- More efficient search algorithms
  - May find strong interactions
  - Require more-intensive computations
- AutoDock-GPU code structure
  - Allows easy exchange between different local-search algorithms
  - Available local-search methods
    - ★ *Solis-Wets* (legacy)
    - ★ *ADADELTA* (newly incorporated)



Local Search: ADADELTA

```
/* Fine-Level Parallelism */
Function GC (genotype)
    /* Gradients in atomic space */
    for each rot-item in N_pose−rot do
        PoseCalculation
    for each lig-atom in N_atom do
        InterGradient
    for each intra-pair in N_intra−contrib do
        IntraGradient
    /* Conversion into genetic space */
    Gtrans // Translational gradients
    Grigidrot // Rigid-body rotation gradients
    Grotbond // Rotatable-bond gradients
```

Local Search: ADADELTA

TECHNISCHE UNIVERSITÄT DARMSTADT Scripps Research

```
/* Fine-Level Parallelism */
Function GC (genotype)
    /* Gradients in atomic space */
    for each rot-item in N_pose−rot do
        PoseCalculation
    for each lig-atom in N_atom do
        InterGradient
    for each intra-pair in N_intra−contrib do
        IntraGradient
    /* Conversion into genetic space */
    Gtrans // Translational gradients
    Grigidrot // Rigid-body rotation gradients
    Grotbond // Rotatable-bond gradients
```

```
/* Fine-Level Parallelism */
Function ADADELTA (genotype)
    gradient = GC (genotype)
    while (N_LS−iters < N^MAX_LS−iters) do
        new-genotype = update-rule (genotype, gradient)
        if SF (new-genotype) < SF (genotype) then
            genotype = new-genotype
        gradient = GC (genotype)
```

OpenCL Work Distribution

- Parameters
  - $\mathrm{NDR}_{\mathrm{size}}$: # work-items per kernel
  - R: # LGA runs
  - P: population size
  - lsrate: local-search rate
  - $\mathrm{WG}_{\mathrm{size}}$: # work-items per work-group

$$\mathrm{NDR}_{\mathrm{size}}^{\mathsf{Krnl\_GA}} = \{R \times P \times \mathrm{WG}_{\mathrm{size}},\ 1,\ 1\}$$

$$\mathrm{NDR}_{\mathrm{size}}^{\mathsf{Krnl\_LS}} = \{R \times P \times \mathsf{lsrate} \times \mathrm{WG}_{\mathrm{size}},\ 1,\ 1\}$$

OpenCL Work Distribution

- Parameters
  - $\mathrm{NDR}_{\mathrm{size}}$: # work-items per kernel
  - R: # LGA runs
  - P: population size
  - lsrate: local-search rate
  - $\mathrm{WG}_{\mathrm{size}}$: # work-items per work-group

$$\mathrm{NDR}_{\mathrm{size}}^{\mathsf{Krnl\_GA}} = \{R \times P \times \mathrm{WG}_{\mathrm{size}},\ 1,\ 1\}$$

$$\mathrm{NDR}_{\mathrm{size}}^{\mathsf{Krnl\_LS}} = \{R \times P \times \mathsf{lsrate} \times \mathrm{WG}_{\mathrm{size}},\ 1,\ 1\}$$

- For all experiments
  - $R = 100$
  - $P = 150$
  - lsrate $= 100\%$
- If $\mathrm{WG}_{\mathrm{size}}^{\mathrm{GPU}} = 64 \longrightarrow \mathrm{NDR}_{\mathrm{size}}^{\mathrm{GPU}} = \{960000,\ 1,\ 1\}$
- If $\mathrm{WG}_{\mathrm{size}}^{\mathrm{CPU}} = 16 \longrightarrow \mathrm{NDR}_{\mathrm{size}}^{\mathrm{CPU}} = \{240000,\ 1,\ 1\}$

Hardware Setup

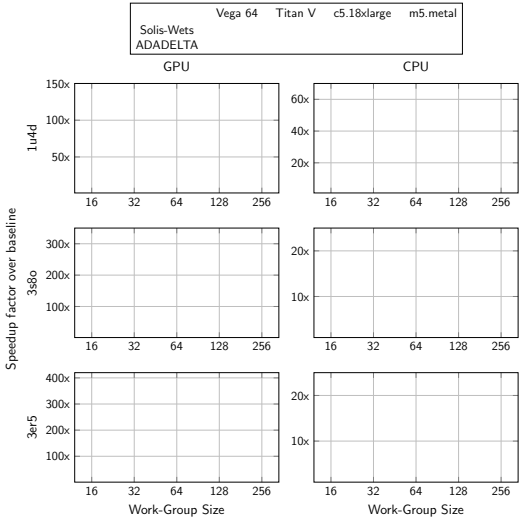
- AutoDock: v4.2.6 (baseline)
  - Implements only Solis-Wets method
  - Does not support multithreading
    - Run on a Xeon Platinum 8124M @3.0 GHz CPU core

Hardware Setup

- AutoDock: v4.2.6 (baseline)
  - Implements only Solis-Wets method
  - Does not support multithreading
    - Run on a Xeon Platinum 8124M @3.0 GHz CPU core
- AutoDock-GPU: v1.2
  - Implements both Solis-Wets and ADADELTA methods
  - Run on different GPU/CPU accelerators
    - Radeon RX Vega 64 GPU (on-premise)
    - Volta Titan V GPU (on-premise)
    - Xeon Platinum 8124M @3.0 GHz 36-core CPU (AWS `c5.18xlarge`)
    - Xeon Platinum 8175M @2.5 GHz 48-core CPU (AWS `m5.metal`)

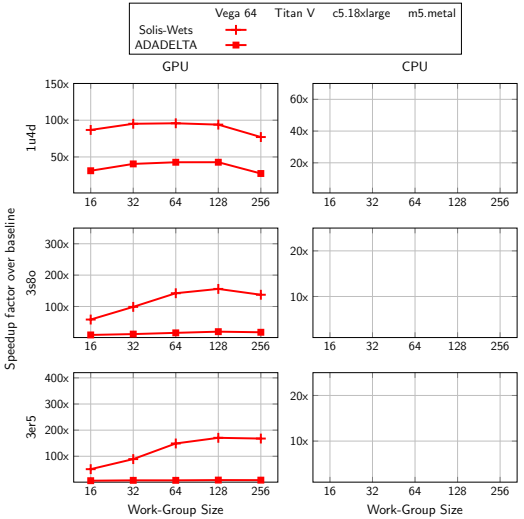
Speedup vs. OpenCL Work-Group Size

- Inputs of different complexity
  - `1u4d` (low), `3s8o` (medium), `3er5` (high)



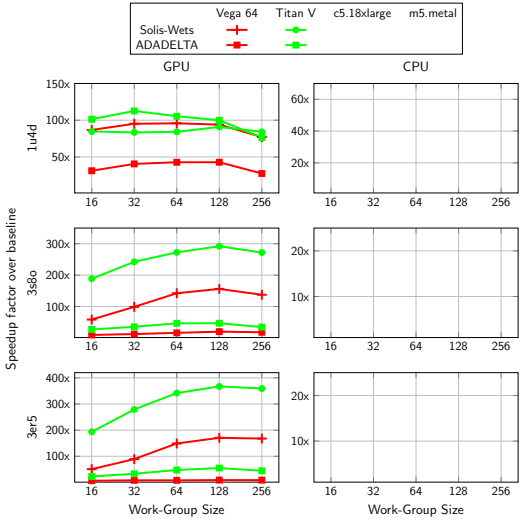
Speedup vs. OpenCL Work-Group Size

- Inputs of different complexity
  - `1u4d` (low), `3s8o` (medium), `3er5` (high)
- GPUs: best $\mathrm{WG}_{\mathrm{size}}$ depends on ...
  - Molecular complexity
  - Accelerator being employed



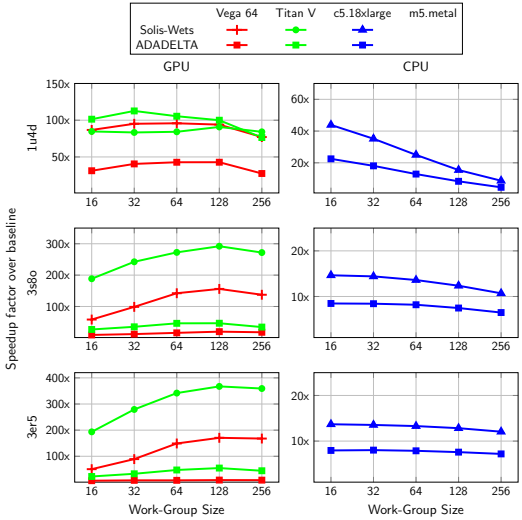
Speedup vs. OpenCL Work-Group Size

- Inputs of different complexity
  - `1u4d` (low), `3s8o` (medium), `3er5` (high)
- GPUs: best $\mathrm{WG}_{\mathrm{size}}$ depends on ...
  - Molecular complexity
  - Accelerator being employed



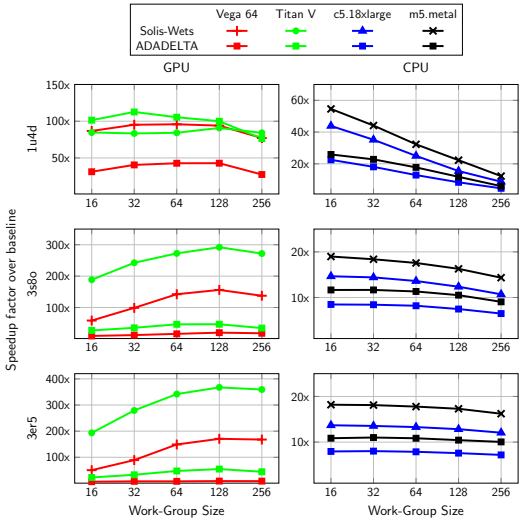
Speedup vs. OpenCL Work-Group Size

- Inputs of different complexity
  - `1u4d` (low), `3s8o` (medium), `3er5` (high)
- GPUs: best $\mathrm{WG}_{\mathrm{size}}$ depends on ...
  - Molecular complexity
  - Accelerator being employed
- CPUs: faster executions when ...
  - Smaller $\mathrm{WG}_{\mathrm{size}}$
  - $\mathrm{WG}_{\mathrm{size}} = 16$



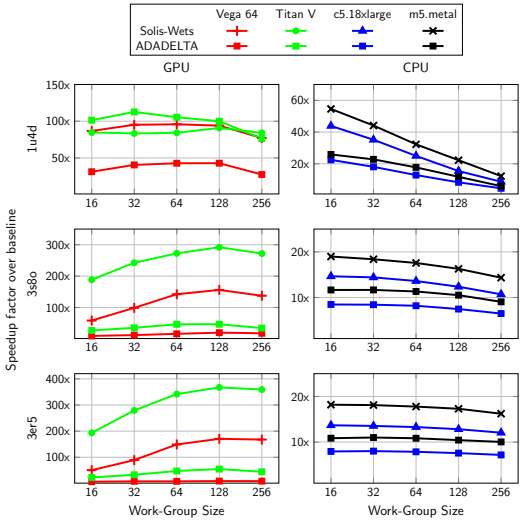
Speedup vs. OpenCL Work-Group Size

- Inputs of different complexity
  - `1u4d` (low), `3s8o` (medium), `3er5` (high)
- GPUs: best $\mathrm{WG}_{\mathrm{size}}$ depends on ...
  - Molecular complexity
  - Accelerator being employed
- CPUs: faster executions when ...
  - Smaller $\mathrm{WG}_{\mathrm{size}}$
  - $\mathrm{WG}_{\mathrm{size}} = 16$

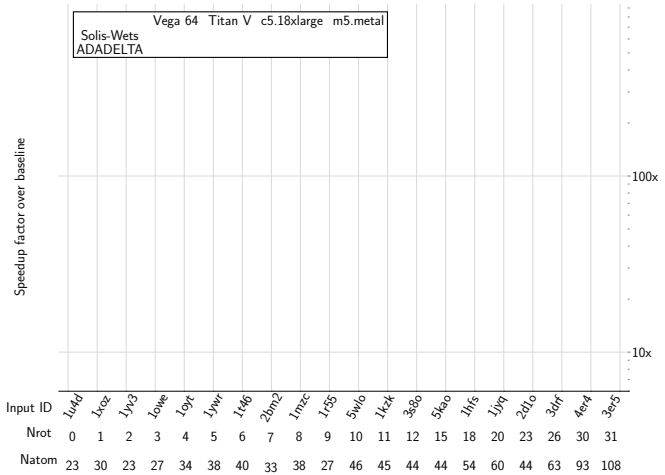


Speedup vs. OpenCL Work-Group Size

- Inputs of different complexity
  - `1u4d` (low), `3s8o` (medium), `3er5` (high)
- GPUs: best $\mathrm{WG}_{\mathrm{size}}$ depends on ...
  - Molecular complexity
  - Accelerator being employed
- CPUs: faster executions when ...
  - Smaller $\mathrm{WG}_{\mathrm{size}}$
  - $\mathrm{WG}_{\mathrm{size}} = 16$
- For next experiments
  - $\mathrm{WG}_{\mathrm{size}}^{\mathrm{CPU}} = 16$
  - $\mathrm{WG}_{\mathrm{size}}^{\mathrm{GPU}} = 64$
    - ★ Min. multiple of a Nvidia *warp* (32) and AMD *wavefront* (64)

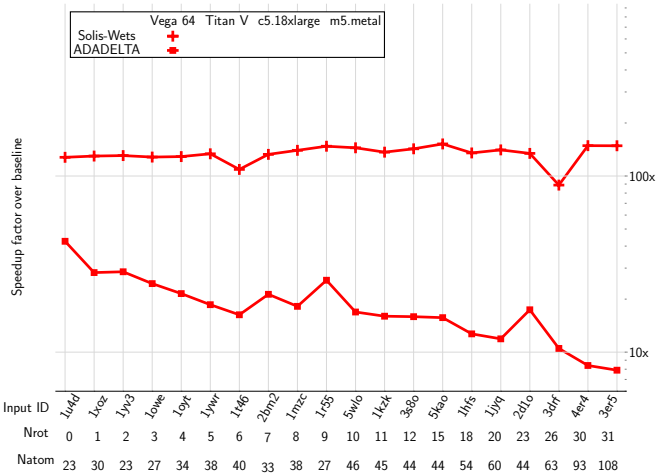


Speedup vs. Molecular Complexity



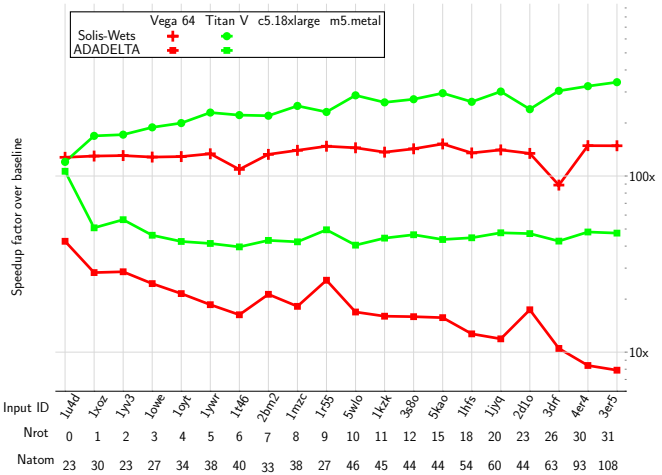
Growing molecular complexity →

Speedup vs. Molecular Complexity



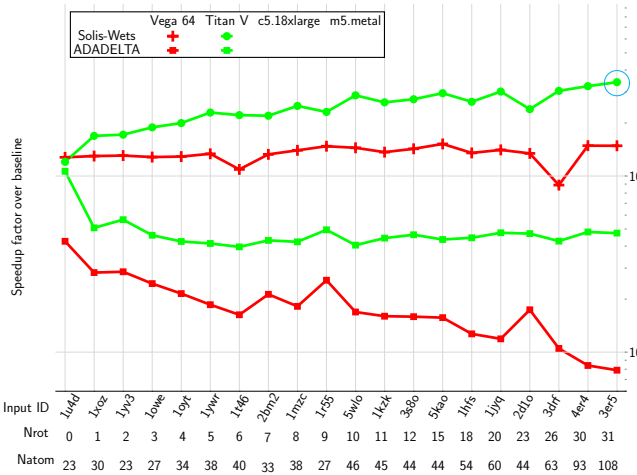
Growing molecular complexity →

Speedup vs. Molecular Complexity



Growing molecular complexity →

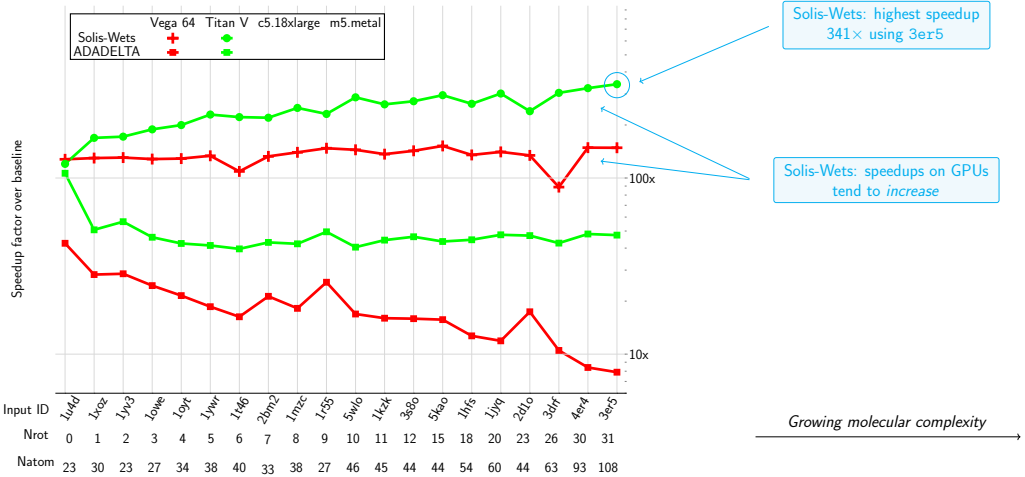
Speedup vs. Molecular Complexity



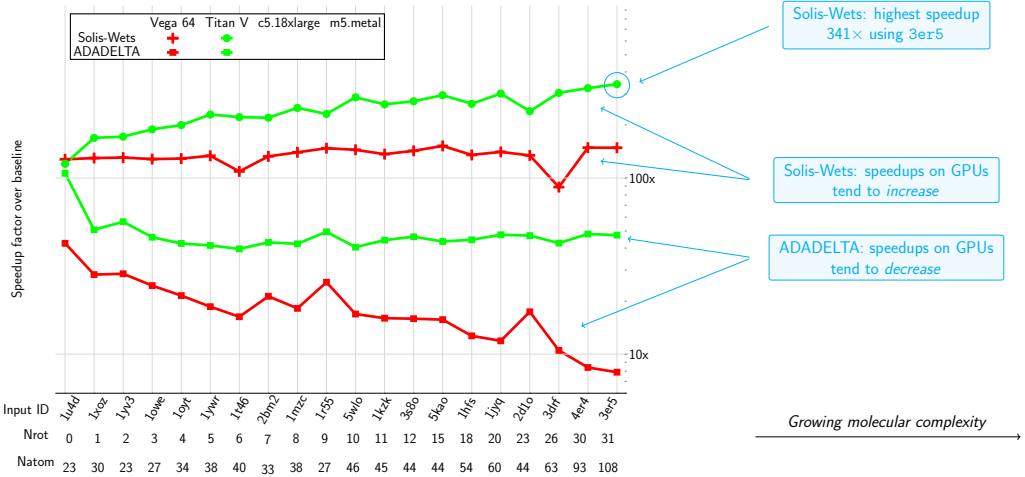
Solis-Wets: highest speedup
341× using 3er5

Growing molecular complexity →

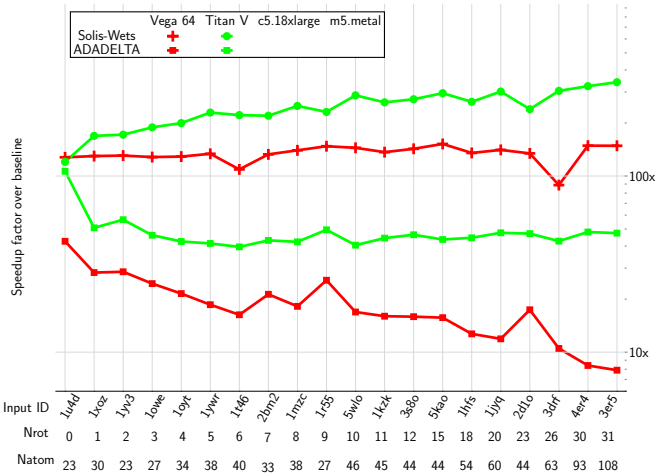
Speedup vs. Molecular Complexity



Speedup vs. Molecular Complexity

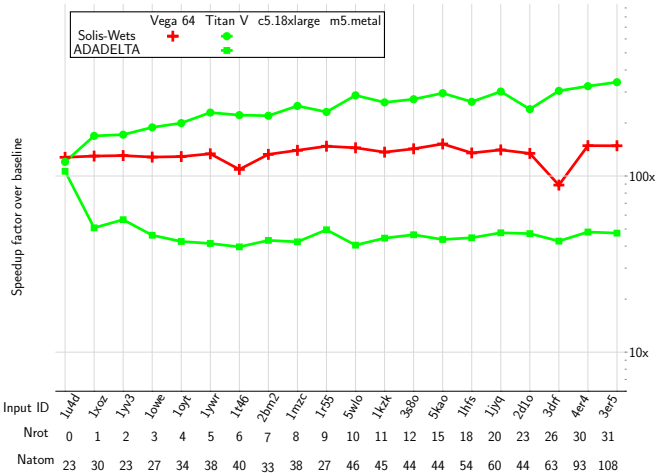


Speedup vs. Molecular Complexity



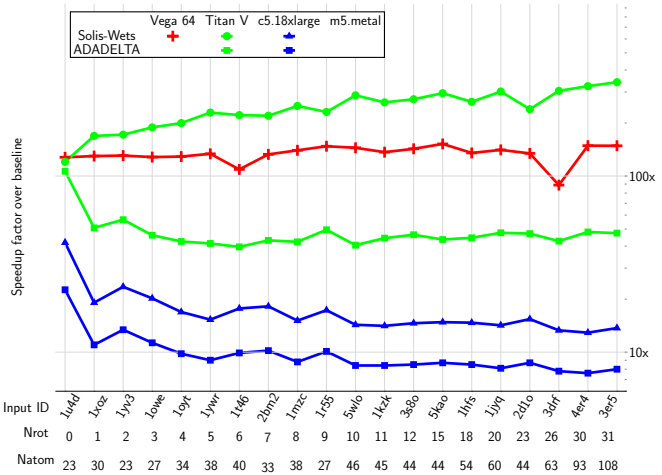
Growing molecular complexity →

Speedup vs. Molecular Complexity



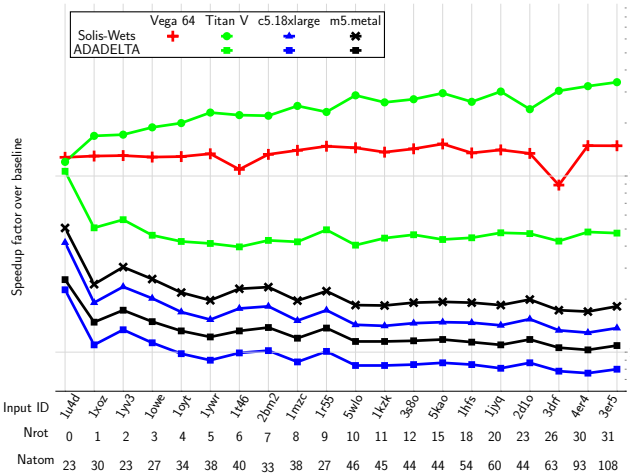
Growing molecular complexity →

Speedup vs. Molecular Complexity



Growing molecular complexity →

Speedup vs. Molecular Complexity



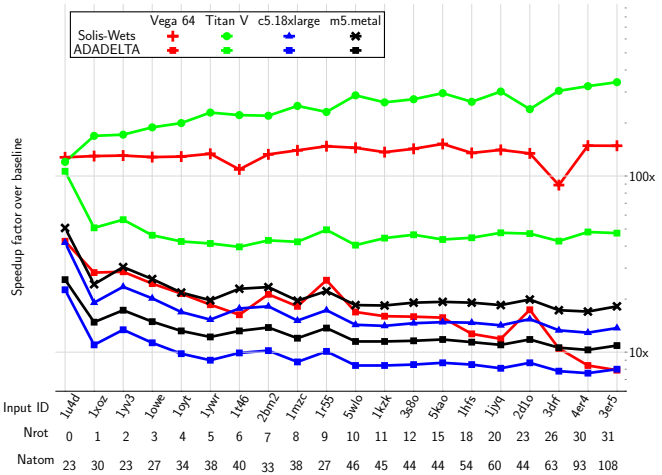
Solis-Wets & ADADELTA:
speedups on CPUs tend to *decrease*

Similar profiles among CPUs
- `m5.metal`: 48 physical cores
- `c5.18xlarge`: 36 physical cores

Speedup ratio: <1.2× - 1.4×>

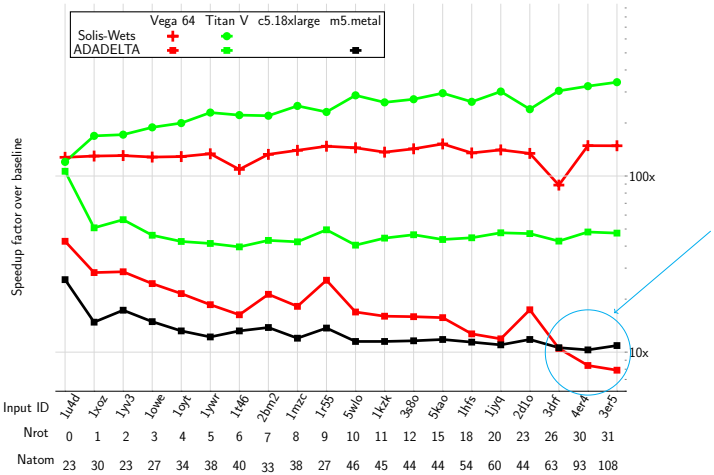
Growing molecular complexity →

Speedup vs. Molecular Complexity



Growing molecular complexity →

Speedup vs. Molecular Complexity

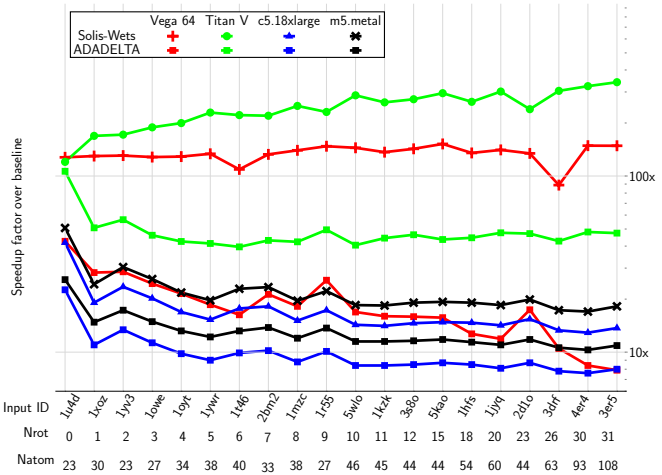


GPUs achieve higher speedups than CPUs (in most cases)

Exception: `3drf`, `4er4`, `3er5`
- Vega 64 GPU: 11×, 8×, 8×
- `m5.metal` CPU: 11×, 10×, 11×

Growing molecular complexity →

Speedup vs. Molecular Complexity

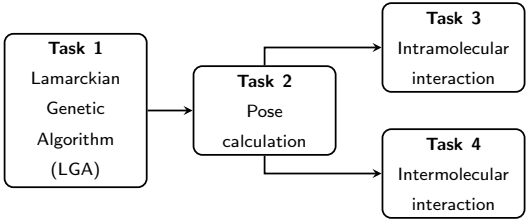


Efficiency:
Solis-Wets > ADADELTA

Growing molecular complexity →

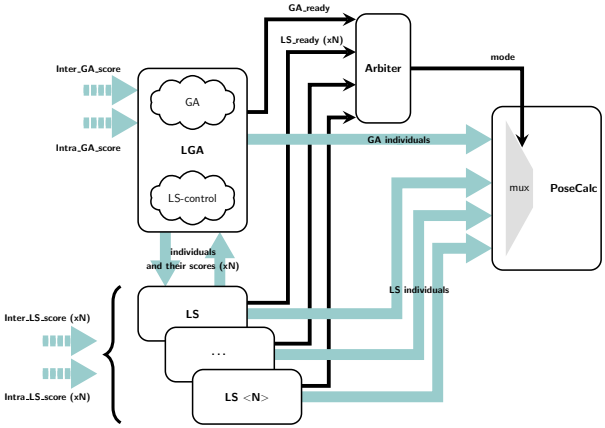
Porting to FPGAs (1/3)

- Data-parallel design on FPGAs
  - Three orders of magnitude *slower*
- Task parallelization
  - Each task coded as a single work-item kernel
  - Kernels communicate via OpenCL pipes
  - General design practices
    - ★ Pipelining loops within each kernel
    - ★ Minimizing loops initiation interval



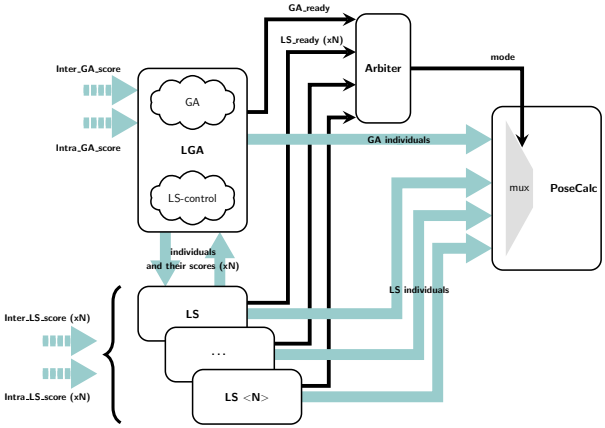
Porting to FPGAs (2/3)

- Final design composed of 27 kernels
  - Additional kernels (and pipes)
    - ★ Local-search kernels (Solis-Wets)
    - ★ Random number generators



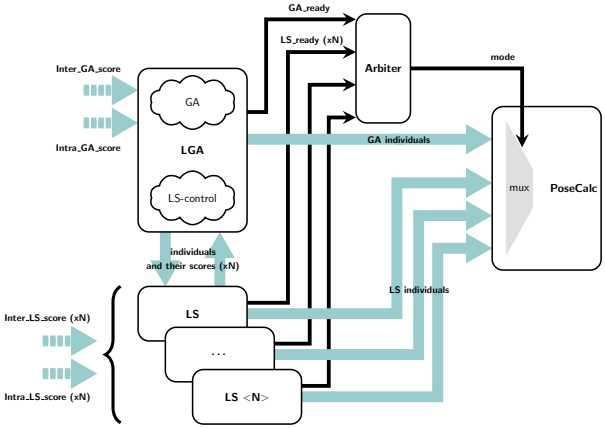
Porting to FPGAs (2/3)

- Final design composed of 27 kernels
  - Additional kernels (and pipes)
    - ★ Local-search kernels (Solis-Wets)
    - ★ Random number generators
  - Complex synchronization
    - ★ All kernels running *simultaneously*
    - ★ Pipes configured as *blocking* and *non-blocking*
    - ★ Non-blocking pipes handle variable kernel communication



Porting to FPGAs (2/3)

- Final design composed of 27 kernels
  - Additional kernels (and pipes)
    - ★ Local-search kernels (Solis-Wets)
    - ★ Random number generators
  - Complex synchronization
    - ★ All kernels running *simultaneously*
    - ★ Pipes configured as *blocking* and *non-blocking*
    - ★ Non-blocking pipes handle variable kernel communication
  - Lower speedups on FPGAs wrt. GPUs
    - ★ Arria 10: 3× faster than serial baseline



Porting to FPGAs (3/3)

- AWS `f1.2xlarge` instance
  - Compilation successful after minor code changes
  - Execution on FPGA fails
    - Non-blocking pipes are *not supported* in Xilinx tools
  - Possible avenue
    - E.g., replacing variable by constant upper-bounds for loops
    - Using only blocking pipes (supported!)
  - Transforming into a *regular* application ?

Final Remarks (1/2)

- Parallelized AutoDock using OpenCL
  - Challenges due to irregularity in AutoDock
    - ⋆ Divergent control performing local search
    - ⋆ Loops with variable upper bounds
    - ⋆ Time-intensive score evaluations
  - Required large-scale code re-structuring
    - ⋆ Trees $\rightarrow$ arrays
    - ⋆ Score function re-structuring

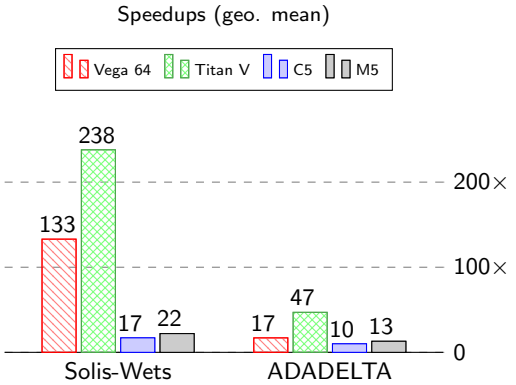
Final Remarks (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT Scripps Research

- Parallelized AutoDock using OpenCL
  - Challenges due to irregularity in AutoDock
    - ⋆ Divergent control performing local search
    - ⋆ Loops with variable upper bounds
    - ⋆ Time-intensive score evaluations
  - Required large-scale code re-structuring
    - ⋆ Trees $\rightarrow$ arrays
    - ⋆ Score function re-structuring
  - OpenCL work-items
    - ⋆ Atomic rotations and score calculations
  - OpenCL work-groups
    - ⋆ Simultaneously processing molecular poses

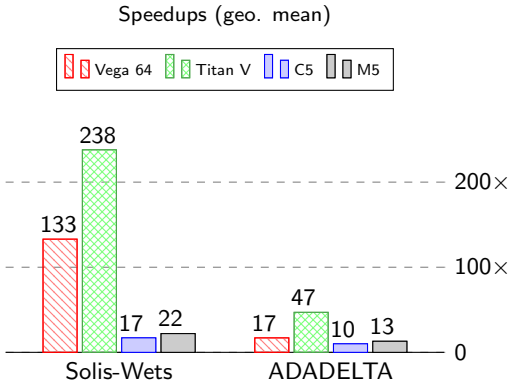
Final Remarks (2/2)

- Evaluated AutoDock-GPU performance
  - Comparing local search
    - ★ Solis-Wets vs. ADADELTA
    - ★ Solis-Wets: higher speedups
    - ★ ADADELTA: better pose predictions
  - Overall performance
    - ★ Depends on the input molecule



Final Remarks (2/2)

- Evaluated AutoDock-GPU performance
  - Comparing local search
    - ★ Solis-Wets vs. ADADELTA
    - ★ Solis-Wets: higher speedups
    - ★ ADADELTA: better pose predictions
  - Overall performance
    - ★ Depends on the input molecule
- Lower speedups achieved on FPGAs
  - Due to irregularity in AutoDock



Parallelizing Irregular Computations for Molecular Docking

https://github.com/ccsb-scripps/AutoDock-GPU

https://www.esa.tu-darmstadt.de https://forlilab.org