Porting and Optimizing Molecular Docking Simulations on SX-Aurora Vector Engine

Leonardo Solis-Vasquez (TU Darmstadt)
Erich Focht (NEC Germany)
Andreas Koch (TU Darmstadt)

31st Workshop on Sustained Simulation Performance (virtual), March 16-19, 2021
Computer-Aided Drug Design

- Contributes fighting against diseases
  - AIDS, cancer, COVID
- Molecular docking simulations
  - Key methods in computer-aided drug design
  - Predict molecular interactions at *short distances*
    - Receptor: macromolecule
    - Ligand: small molecule ----> drug candidate
  - Benefits
    - Shorten the task of identifying drug candidates
    - Subsequent lab experiments can be performed on a narrowed list of promising ligands
    - *Reduces the overall need for costly and slow lab experiments*
Molecular Docking

- It aims to find poses of strong interaction

- Scoring function
  - Measures a pose strength
  - Computational expensive
    - $N_{\text{atom, receptor}} > 1000$
    - $N_{\text{atom, ligand}} < 100$
Molecular Docking

- Search methods
  - Finds an optimal pose
    - Strong interaction
  - Usually based on heuristic

- Representation
  - Encodes a pose in terms of e.g., translation, rotation, torsion
AutoDock

- A widely used software for molecular docking simulation
  - Developed by Scripps Research Institute (USA)
- Applicability
  - FightAIDS@Home project
  - OpenPandemics: COVID-19
- Receptor-ligand docking
- Lamarckian Genetic Algorithm
  - Hybridizes search methods
  - Performs compute-intensive score calculations
Encoding Ligand Poses

- Receptor
  - Treated as a rigid body

- Ligand poses are
  - Encoded as a set of variables
    - Translation
    - Orientation
    - Torsion
  - The solutions of the docking problem

\[
\text{Pose}_{\text{Lig}} = \{ x, y, z, \phi, \theta, \alpha, \psi_1, \psi_2, \ldots, \psi_{N_{\text{rot}}} \}
\]
Mapping Docking into Genetic Evolution

- Pose ----> individual
  - An individual is
    - A member of a population
    - Represented by its genome

- Pose variables ----> genes

\[
\text{Pose}_{\text{Lig}} = \{x, y, z, \ldots, \psi_{\text{Nrot}} \} \\
\text{Genome} = \{\text{gene}_1, \text{gene}_2, \text{gene}_3, \ldots, \text{gene}_M, \} 
\]
Genetic Algorithm (GA)

- New individuals are generated through genetic evolution

- Mimicking Darwinian evolution
  - Crossover
    - New individuals inherit genes from either parent
  - Mutation
    - Genes change by a random amount
  - Selection
    - Better-suited individuals reproduce
Lamarckian Genetic Algorithm (LGA)

- Hybrid search
  - LGA = Genetic Algorithm + Local Search

- Local Search
  - Score minimization
  - Genes experience change, which is
    - A constrained random amount
    - Adapted depending on score improvement
  - Self adaptive
    - Variable runtime
Scoring Function

- Binding energy (Kcal/mol)
  - Molecular mechanics
- Coefficients and Look-Up tables
  - $W_{vdw}$, $W_{hb}$, $W_{el}$, $W_{ds}$, $W_{rot}$
  - A, B, C, D, S, V, E, q
- Interatomic distance $r_{ij}$
  - Between atoms i and j
  - Varies with every new pose

$$SF = \sum_{i,j} \left[ W_{vdw} \left( \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right) \right] +$$

Lennard-Jones

$$W_{hb} E(t) \left( \frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}} \right) +$$

Hydrogen bonding

$$W_{el} \left( \frac{q_i q_j}{\epsilon (r_{ij}) r_{ij}} \right) +$$

Coulomb's law

$$W_{ds} \left( S_i V_j + S_j V_i \right) e^{-\frac{r_{ij}^2}{2\sigma^2}}$$

Desolvation
Baseline Project

- **OpenCL Accelerated Molecular Docking on FPGAs**
  - [https://git.esa.informatik.tu-darmstadt.de/docking/ocladock-fpga](https://git.esa.informatik.tu-darmstadt.de/docking/ocladock-fpga)
  - Accelerated AutoDock developed at TU Darmstadt

- Computational-intensive parts already well defined
  - LGA run is offloaded onto an FPGA device

- Optimized for Intel FPGAs
  - Refactored code for leveraging
    - Custom hardware pipelines & memory hierarchies
Baseline: Device Code

- Task-parallel approach
  - Several kernels running simultaneously
  - Kernels communicate with each other using OpenCL pipes

- Each kernel executes a different task
  - GA, LS, energy interactions
  - Some kernels are replicated

- Each kernel is single threaded
  - In contrast to index-based multithreading
Porting from OpenCL to SX-Aurora

- **Host**
  - Replaced OpenCl host APIs with VEO API
    - Some resemble each other
      - clCreateBuffer
      - veo_alloc_mem
      - clEnqueueWriteBuffer
      - veo_write_mem
  - Added custom-made wrapper functions for some VEO APIs
    - For more descriptive/extra error messages

```c
uint64_t wrapper_veo_get_sym (veo_proc_handle* proc, uint64_t libhdl, const char* symname) {
    uint64_t symbol = veo_get_sym (proc, libhdl, symname);
    if (symbol == 0) {
        std::cout << "\veo_get_sym():\tfailed to find symbol.\" << std::endl;
        std::exit (EXIT_FAILURE);
    }
    return symbol;
}
```
Porting from OpenCL to SX-Aurora

- **Device**
  - OpenCL kernels --> C functions
    - Main function: performs a single LGA run
  - OpenCL pipes were removed
    - Inter-kernel communication is performed via *function calls instead*
  - Single-threaded baseline
    - OpenCL C structure was maintained
    - Minor code modifications on loop structures
Initial Optimizations

- Based on NEC tuning guidelines
  - Removing data dependencies
  - Using 4-byte variables for index and loop-control variable
  - Parallelization of LGAs
    - LGA runs are independent
    - ----> distributed among cores using OpenMP

- Fully vectorization time-consuming score/energy functions
  - Ligand-receptor interaction
  - Ligand internal interaction

- Still some room for improvement ...
Performance Tuning: Analysis

First results:

- Loops short
- Loops often not vectorized
- *Single precision computations!*

- Key functions:
  - Calculate ligand atoms positions from “genome”
  - Compute ligand-receptor energy
  - Compute ligand internal energy contributions
- Inner loops: over ligand atoms, atom pairs or ligand torsions

******** Program Information ********

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Time (sec)</td>
<td>114.700255</td>
</tr>
<tr>
<td>User Time (sec)</td>
<td>114.692841</td>
</tr>
<tr>
<td>Vector Time (sec)</td>
<td>10.757880</td>
</tr>
<tr>
<td>Inst. Count</td>
<td>124837499135</td>
</tr>
<tr>
<td>V. Inst. Count</td>
<td>857519799</td>
</tr>
<tr>
<td>V. Element Count</td>
<td>31273502842</td>
</tr>
<tr>
<td>V. Load Element Count</td>
<td>9653868399</td>
</tr>
<tr>
<td>FLOP Count</td>
<td>28193233210</td>
</tr>
<tr>
<td>MOPS</td>
<td>1361.635301</td>
</tr>
<tr>
<td>MOPS (Real)</td>
<td>1361.498080</td>
</tr>
<tr>
<td>MFLOPS</td>
<td>245.823994</td>
</tr>
<tr>
<td>MFLOPS (Real)</td>
<td>245.799221</td>
</tr>
<tr>
<td>A. V. Length</td>
<td>36.469715</td>
</tr>
<tr>
<td>V. Op. Ratio (%)</td>
<td>20.609206</td>
</tr>
<tr>
<td>L1 Cache Miss (sec)</td>
<td>3.521309</td>
</tr>
<tr>
<td>CPU Port Conf. (sec)</td>
<td>0.000000</td>
</tr>
<tr>
<td>V. Arith. Exec. (sec)</td>
<td>1.064510</td>
</tr>
<tr>
<td>V. Load Exec. (sec)</td>
<td>7.402312</td>
</tr>
<tr>
<td>VLD LLC Hit Element Ratio (%)</td>
<td>99.994095</td>
</tr>
<tr>
<td>FMA Element Count</td>
<td>910694400</td>
</tr>
<tr>
<td>Power Throttling (sec)</td>
<td>0.000000</td>
</tr>
<tr>
<td>Thermal Throttling (sec)</td>
<td>0.000000</td>
</tr>
<tr>
<td>Memory Size Used (MB)</td>
<td>302.000000</td>
</tr>
</tbody>
</table>
Code Structure

For LGA run: distribute over VE cores by OpenMP

Lamarckian Genetic Algorithm (LGA) run

While stop condition not reached

For each individual in population

- Selectively apply mutations
- Ligand pose calculations
- Ligand-receptor energy calculation
- Ligand internal energy calculation
Lamarckian Genetic Algorithm (LGA) run

For each individual in population

While stop condition not reached

Selectively apply mutations
Ligand pose calculations
Ligand-receptor energy calculation
Ligand internal energy calculation

For LGA run: distribute over VE cores by OpenMP

Short loops
Lamarckian Genetic Algorithm (LGA) run

For each individual in population

- Selectively apply mutations
- Ligand pose calculations
- Ligand-receptor energy calculation
- Ligand internal energy calculation

While stop condition not reached

For LGA run: distribute over VE cores by OpenMP

Not large, initially ~128

Short loops
Loop Pushing

- Loop over individuals (genomes)
  - Increase population: 2048
  - Move loop inside functions, make it innermost (for ncc)
  - Tricky for genetic algorithm part

- Replaced random generator by ASL function call
  - Pull many random numbers at once
Loop Pushing: First Results

Case: 1t46, 40 atoms, 111 rotations
100 LGA runs, population size: 2048

Much better!
Loop Pushing: First Results

Case: **1t46**, 40 atoms, 111 rotations
100 LGA runs, population size: 2048

Much better!

---

### Program Information

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Time (sec)</td>
<td>45.922564</td>
</tr>
<tr>
<td>User Time (sec)</td>
<td>354.048237</td>
</tr>
<tr>
<td>Vector Time (sec)</td>
<td>313.792794</td>
</tr>
<tr>
<td>Inst. Count</td>
<td>352157100866</td>
</tr>
<tr>
<td>V. Inst. Count</td>
<td>108264054847</td>
</tr>
<tr>
<td>V. Element Count</td>
<td>238713987151364</td>
</tr>
<tr>
<td>V. Load Element Count</td>
<td>2412434798072</td>
</tr>
<tr>
<td>FLOP Count</td>
<td>17850210880912</td>
</tr>
<tr>
<td>MOPS</td>
<td>85464.630971</td>
</tr>
<tr>
<td>MOPS (Real)</td>
<td>654920.857127</td>
</tr>
<tr>
<td>MFLOPS</td>
<td>50724.155528</td>
</tr>
<tr>
<td>MFLOPS (Real)</td>
<td>388702.402827</td>
</tr>
<tr>
<td>A. V. Length</td>
<td>220.589096</td>
</tr>
<tr>
<td>V. Op. Ratio (%)</td>
<td>99.188735</td>
</tr>
<tr>
<td>L1 Cache Miss (sec)</td>
<td>8.237327</td>
</tr>
<tr>
<td>CPU Port Conf. (sec)</td>
<td>12.740749</td>
</tr>
<tr>
<td>V. Arith. Exec. (sec)</td>
<td>197.959744</td>
</tr>
<tr>
<td>V. Load Exec. (sec)</td>
<td>105.976551</td>
</tr>
</tbody>
</table>

### FTRACE

<table>
<thead>
<tr>
<th>TIME[sec][%]</th>
<th>EXCLUSIVE</th>
<th>AVER.TIME</th>
<th>MOPS</th>
<th>MFLOPS</th>
<th>V. OP</th>
<th>AVER.</th>
<th>VECTOR</th>
<th>CPU</th>
<th>PORT</th>
<th>VLD</th>
<th>LLC</th>
<th>HIT</th>
<th>E.%</th>
</tr>
</thead>
<tbody>
<tr>
<td>262319</td>
<td>187.783(55.4)</td>
<td>0.716</td>
<td>124156.5</td>
<td>76670.0</td>
<td>99.24</td>
<td>222.2</td>
<td>182.542</td>
<td>0.766</td>
<td>0.000</td>
<td>97.34</td>
<td>energy_ia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>262319</td>
<td>59.600(17.6)</td>
<td>0.257</td>
<td>49133.4</td>
<td>14087.6</td>
<td>99.66</td>
<td>220.8</td>
<td>58.935</td>
<td>0.293</td>
<td>12.676</td>
<td>92.05</td>
<td>energy_ie</td>
<td></td>
<td></td>
</tr>
<tr>
<td>262319</td>
<td>51.431(15.2)</td>
<td>0.196</td>
<td>64558.4</td>
<td>49827.2</td>
<td>99.47</td>
<td>216.9</td>
<td>49.268</td>
<td>0.838</td>
<td>0.000</td>
<td>89.96</td>
<td>calc_pc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(cumulated times, 8 OpenMP threads)
Loop Pushing: First Results

Case: 1t46, 40 atoms, 111 rotations
100 LGA runs, population size: 2048

******** Program Information ********
Real Time (sec) : 45.922564

VE 10B, 8 core, 1.4GHz
UMA mode

Comparison with other systems:

<table>
<thead>
<tr>
<th>Testcase</th>
<th>Runtimes [s]: popsize = 2048</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NVIDIA V100</td>
</tr>
<tr>
<td>1t46</td>
<td>6.56</td>
</tr>
</tbody>
</table>

V100 vs. VE10
3x SP peak
but 7x better
Packed Single Precision Struggles

- VE reaches $\frac{1}{3}$ of v100 single precision performance only in packed mode!
  - 2 x 32 bit floats in 64 bit word
  - Vector length effectively up to 512 elements, must be even

- Code vs. Compiler (ncc)
  - SP functions: `floorf()`, `ceilf()`, `sinf()`, `expf()` must be used, not `floor()`, `ceil()`, `sin()`, `exp()`.
  - Issues that hinder packed vectorization
    - IF blocks inside loop (waiting for fix)
    - `unsigned int` loop index (” “ “ )
    - Gather/Scatter (won’t fix)
  - GPU code uses `native_exp()`, `native_sin()`, etc…
    - Defined in OpenCL, can be implemented with reduced accuracy

- => unpacked vectorized code, with proper SP functions:
  - From ~43s down to 34.3s (factor 5.2 to v100)
LLVM-VE and RegionVectorizer

- Simon Moll (NEC HPCE)
- RV is an outer loop vectorizer, can deal excellently with predication (IF blocks)
  - Whole function vectorization of SLEEF math library functions
- + recent work on backend support for packed vectorization
- See Simon’s talk on Friday, 8:35

- Compiling only `energy_ia.c` with LLVM-VE-RV: packed vectorization!
Further Tunings

- **Multi-VE Processes for NUMA**
  - In NUMA mode an ncc OpenMP process can only get 4 threads
    - Actually could have used LLVM-VE...
  - Using AVEO multi-VE capability to:
    - Support multiple processes on one VE => NUMA with OpenMP is possible
    - Support multiple VEs

- **Reduced accuracy implementations of** \textit{sqrtf()} \textit{and} \textit{expf()}

- **No packed mode possible for** \texttt{energy_ie.c}
  - Dominated by indirect loads of data / vector gather operations
Packed Single Precision: VE ISA Deficits

Vector Gather / Scatter are impossible in packed mode

```
VGT %v10, %v11, %s12, %s13 [, %vm14]
```

target
64 bit

addresses
64 bit

overlap
hints

“Packed” loading 512 x 32 bit elements requires 2 x VGT, VSHF, VMRG

For arrays smaller than $2^{32}$ elements the address VREG could be instead interpreted as a packed array of 4-byte aligned offsets, requiring just one such packed gather/scatter. Example:

```
PVGTO %v10, %v11, %s12
```

target
32 bit

offsets
32 bit

base
address
64 bit

mem

%v10

1
2
3
4

256 x 64 bit

%v11

1
2
3
4

256 x 64 bit
## Finally: Evaluation Results

<table>
<thead>
<tr>
<th>Testcase</th>
<th>NVIDIA V100</th>
<th>NVIDIA RTX2070</th>
<th>NVIDIA RTX3070</th>
<th>NVIDIA A100</th>
<th>AMD EPYC 7502 2x32-core</th>
<th>AMD EPYC 7742 2x64-core CPU</th>
<th>Intel SKL Gold 6126 12-core</th>
<th>NEC SX-Aurora VE20B (NUMA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1t46</td>
<td>6.56</td>
<td>14.89</td>
<td>10.81</td>
<td>2.62</td>
<td>108.04</td>
<td>61.59</td>
<td>425.64</td>
<td>12.28</td>
</tr>
</tbody>
</table>

### Geometric mean over 31 validation tests
- Runtimes normalized to VE20B results
- (Includes many small cases)
- popsize=2048, nrun=100
- AutoDock-GPU v1.1
Conclusions

● Porting OpenCL-AutoDock onto SX-Aurora
  ○ Relatively smooth, main code structure was maintained
  ○ Leveraged VEO/AVEO

● Optimization for SX-Aurora was much more involved
  ○ Increased the population size (2048) from default (150) for vector length
  ○ Loop pushing improved performance significantly
    ■ Local-Search, score/energy calculations
  ○ Packed mode is mandatory for full single precision performance
    ■ LLVM-VE is currently the most advanced

● Competitive performance
  ○ > 3.7x times faster than modern 128 core servers, at lower power consumption
  ○ Better than expected performance compared to GPUs
Outlook

- Working with Scripps Research on joining the Open Pandemics - COVID19 effort with spare SX-Aurora cycles
  - Virtual screening of COVID19 drug candidates