Enhancing Metaheuristic-based Virtual Screening Methods on Massively Parallel and Heterogeneous Systems

Baldomero Imbernón[1], José M. Cecilia [2] and Domingo Giménez [3]

[1−2] Polytechnic School
Catholic University of San Antonio of Murcia (UCAM)
Murcia, Spain

[3] Department of Computing and Systems
University of Murcia
Murcia, Spain

[1]bimbernon@alu.ucam.edu, [2]jmcecilia@ucam.edu, [3]domingo@um.es

March 12, 2016

Table of Contents

Introduction

- Metaheuristic techniques afford optimal approaches for solving optimization problems, combining performance, quality and resource optimization.
- Many of these techniques are used in computing virtual screening processes based on the calculation of a scoring function.
- These screening processes calculate the interaction between a set of chemical compounds (ligands) and a protein (receptor).

Features

- Optimization problem.
- High computational cost.

Introduction

- Metaheuristic techniques afford optimal approaches for solving optimization problems, combining performance, quality and resource optimization.
- Many of these techniques are used in computing virtual screening processes based on the calculation of a scoring function.
- These screening processes calculate the interaction between a set of chemical compounds (ligands) and a protein (receptor).

Features

- Optimization problem.
- High computational cost.

Introduction

- Metaheuristic techniques afford optimal approaches for solving optimization problems, combining performance, quality and resource optimization.
- Many of these techniques are used in computing virtual screening processes based on the calculation of a scoring function.
- These screening processes calculate the interaction between a set of chemical compounds (ligands) and a protein (receptor).

Features

- Optimization problem.
- High computational cost.

Introduction

- Metaheuristic techniques afford optimal approaches for solving optimization problems, combining performance, quality and resource optimization.
- Many of these techniques are used in computing virtual screening processes based on the calculation of a scoring function.
- These screening processes calculate the interaction between a set of chemical compounds (ligands) and a protein (receptor).

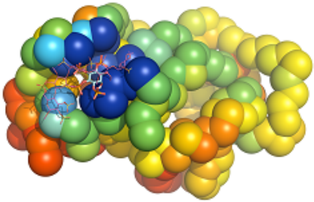
Features

- Optimization problem.
- High computational cost.

Motivation

Problem parallel nature

- Several points in the receptor (called *spots*), where ligands may independently couple.
- A set of bio-inspired metaheuristic techniques that enable parallelization.



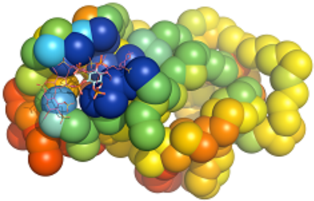
Computational resources

- Heterogeneus computing.
- Application of CUDA-based techniques to accelerate the most expensive parts of the computation.

Motivation

Problem parallel nature

- Several points in the receptor (called *spots*), where ligands may independently couple.
- A set of bio-inspired metaheuristic techniques that enable parallelization.



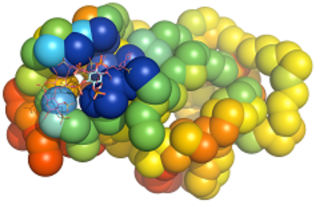
Computational resources

- Heterogeneus computing.
- Application of CUDA-based techniques to accelerate the most expensive parts of the computation.

Motivation

Problem parallel nature

- Several points in the receptor (called *spots*), where ligands may independently couple.
- A set of bio-inspired metaheuristic techniques that enable parallelization.



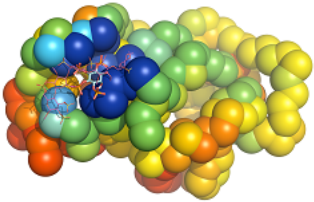
Computational resources

- Heterogeneus computing.
- Application of CUDA-based techniques to accelerate the most expensive parts of the computation.

Motivation

Problem parallel nature

- Several points in the receptor (called *spots*), where ligands may independently couple.
- A set of bio-inspired metaheuristic techniques that enable parallelization.



Computational resources

- Heterogeneus computing.
- Application of CUDA-based techniques to accelerate the most expensive parts of the computation.

Metaheuristics for Virtual Screening

- A metaheuristic generic template to apply several metaheuristics through six simple functions.

Generic template for metaheuristics

```
Initialize(S)
while not End(S) do
    Select(S,Ssel)
    Combine(Ssel,Scom)
    Improve(Scom)
    Include(Scom,S)
end while
```

- Independent populations at each *spot* $\Rightarrow$ apply metaheuristic techniques to the *spots* in parallel.
- Possible solutions are generated by moving and rotating around each *spot*.

Metaheuristics for Virtual Screening

- A metaheuristic generic template to apply several metaheuristics through six simple functions.

Generic template for metaheuristics

```
Initialize(S)
while not End(S) do
    Select(S,Ssel)
    Combine(Ssel,Scom)
    Improve(Scom)
    Include(Scom,S)
end while
```

- Independent populations at each *spot* $\Rightarrow$ apply metaheuristic techniques to the *spots* in parallel.
- Possible solutions are generated by moving and rotating around each *spot*.

Metaheuristics for Virtual Screening

- A metaheuristic generic template to apply several metaheuristics through six simple functions.

Generic template for metaheuristics

```
Initialize(S)
while not End(S) do
    Select(S,Ssel)
    Combine(Ssel,Scom)
    Improve(Scom)
    Include(Scom,S)
end while
```

- Independent populations at each *spot* $\Rightarrow$ apply metaheuristic techniques to the *spots* in parallel.
- Possible solutions are generated by moving and rotating around each *spot*.

- An OpenMP scheme is used to divide the work among the GPUs available on the node.

Scoring computation on multicore+multiGPU

```
omp_set_num_threads(number_GPUs)
#pragma omp parallel for
for i=1 to number_GPUs do
    Select_device(Devices[i].id)
    Host_To_GPU(S,Stmp)
    Conformations=Devices[i].conformations
    threads=Devices[i].Threadsblock
    Calculate_scoring<Conformations/threads,threads>(Stmp)
    GPU_To_Host(S,Stmp)
end for
```

- Solutions are grouped into 32 GPU threads, similar to the WARP size to optimize the computation.

Parallelization strategy

- An OpenMP scheme is used to divide the work among the GPUs available on the node.

Scoring computation on multicore+multiGPU

```
omp_set_num_threads(number_GPUs)
#pragma omp parallel for
for i=1 to number_GPUs do
    Select_device(Devices[i].id)
    Host_To_GPU(S,Stmp)
    Conformations=Devices[i].conformations
    threads=Devices[i].Threadsblock
    Calculate_scoring<Conformations/threads,threads>(Stmp)
    GPU_To_Host(S,Stmp)
end for
```

- Solutions are grouped into 32 GPU threads, similar to the WARP size to optimize the computation.

Exploiting heterogeneity

- Assign a similar number of possible solutions to each GPU for computation.
- GPUs of a node may belong to different families and have different computation capabilities.

Solution

- Execute a set of calculations in a *Warm Phase* for experimental estimation of the computational capability of the device.
- Divide the work according to the computational capabilities.

$$\text{Percent} = \frac{\text{Ex.time}_{\text{actualGPU}}}{\text{Ex.time}_{\text{slowestGPU}}}$$

Exploiting heterogeneity

- Assign a similar number of possible solutions to each GPU for computation.
- GPUs of a node may belong to different families and have different computation capabilities.

Solution

- Execute a set of calculations in a *Warm Phase* for experimental estimation of the computational capability of the device.
- Divide the work according to the computational capabilities.

$$\text{Percent} = \frac{\text{Ex.time}_{\text{actualGPU}}}{\text{Ex.time}_{\text{slowestGPU}}}$$

Exploiting heterogeneity

- Assign a similar number of possible solutions to each GPU for computation.
- GPUs of a node may belong to different families and have different computation capabilities.

Solution

- Execute a set of calculations in a *Warm Phase* for experimental estimation of the computational capability of the device.
- Divide the work according to the computational capabilities.

$$\mathrm{Percent} = \frac{\mathrm{Ex.time}_{\mathrm{actualGPU}}}{\mathrm{Ex.time}_{\mathrm{slowestGPU}}}$$

Exploiting heterogeneity

- Assign a similar number of possible solutions to each GPU for computation.
- GPUs of a node may belong to different families and have different computation capabilities.

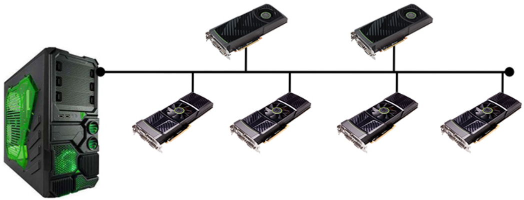
Solution

- Execute a set of calculations in a *Warm Phase* for experimental estimation of the computational capability of the device.
- Divide the work according to the computational capabilities.

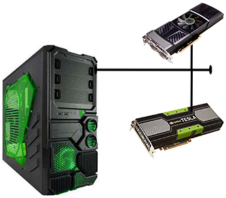
$$\text{Percent} = \frac{\text{Ex.time}_{\text{actualGPU}}}{\text{Ex.time}_{\text{slowestGPU}}}$$

Hardware environment

Jupiter. 12 cores, 32 Gb RAM, 4 GeForce GTX 590 and 2 Tesla C2075.



Hertz. 4 cores, 8 Gb RAM, 1 Tesla K40c and 1 GeForce GTX 580.



Benchmarks and Datasets

Benchmarks

Four metaheuristics considered in the experiments:

- M1. Genetic Algorithm.
- M2. Scatter Search.
- M3. Scatter Search with less intensive local search.
- M4. Neighborhood Search.

metaheuristics M1, M2 and M3 work with a population of 64 individuals for each spot, and M4 with 1024 individuals.

Datasets

Number of atoms of the benchmark compounds from PDB site.

Compounds	Atoms	Compounds	Atoms
2BSM Receptor	3264	2BXG Receptor	8609
2BSM Ligand	45	2BXG Ligand	32

Experimental Results

Execution time (in seconds) obtained with the application to protein PDB:2BXG in Jupiter of the metaheuristics described. Heterogeneous System with 4 GeForce GTX 590 + 2 Tesla C2075.

Metaheuristic	OpenMP	Heterogeneus System			SPEED-UP Heterogeneus Computation vs OpenMP
		Homogeneus Computation	Heterogeneus Computation	percentage reduction	
M1	1402.63	16.96	16.77	1.12	82.70
M2	2272.71	26.57	25.43	4.29	85.53
M3	711.01	8.72	8.46	2.98	81.53
M4	70505.22	764.131	757.32	0.89	92.26

Execution time (in seconds) obtained with the application to protein PDB:2BXG in Hertz of the metaheuristics described. Heterogeneous System with 1 Tesla K40c + 1 GeForce GTX 580.

Metaheuristic	OpenMP	Heterogeneus System			SPEED-UP Heterogeneus Computation vs OpenMP
		Homogeneus Computation	Heterogeneus Computation	percentage reduction	
M1	2327.60	33.92	22.82	32.62	101.96
M2	3908.46	55.56	41.58	25.16	93.98
M3	1336.40	18.13	13.64	24.67	97.96
M4	150958.75	1735.73	1253.64	27.67	120.41

Conclusions

- With the execution of the most expensive parts in GPU the performance obtained is in all the cases superior to 80x.
- The efficient exploitation of heterogeneity allows higher performance in the case study.
- The use of parallel metaheuristics in virtual screening methods facilitates lower execution times and also gets closer to optimal solutions in less time.

Conclusions

- With the execution of the most expensive parts in GPU the performance obtained is in all the cases superior to 80x.
- The efficient exploitation of heterogeneity allows higher performance in the case study.
- The use of parallel metaheuristics in virtual screening methods facilitates lower execution times and also gets closer to optimal solutions in less time.

Conclusions

- With the execution of the most expensive parts in GPU the performance obtained is in all the cases superior to 80x.
- The efficient exploitation of heterogeneity allows higher performance in the case study.
- The use of parallel metaheuristics in virtual screening methods facilitates lower execution times and also gets closer to optimal solutions in less time.

Work in progress

- The parallel nature of the virtual screening problem allows us to parallalize at high level and to extend the calculation to a cluster.
- Use of MPI to assign a set of *spots* to each node in the cluster.

Work modes

- **Static.** A *Warm Phase* to evaluate the computational capacity of each node, and the work is divided accordingly.
- **Dynamic.** Assign a set of *spots* to each node. When a node finishes, it asks for the next group.

Work in progress

- The parallel nature of the virtual screening problem allows us to parallalize at high level and to extend the calculation to a cluster.
- Use of MPI to assign a set of *spots* to each node in the cluster.

Work modes

- **Static.** A *Warm Phase* to evaluate the computational capacity of each node, and the work is divided accordingly.
- **Dynamic.** Assign a set of *spots* to each node. When a node finishes, it asks for the next group.

Work in progress

- The parallel nature of the virtual screening problem allows us to parallalize at high level and to extend the calculation to a cluster.
- Use of MPI to assign a set of *spots* to each node in the cluster.

Work modes

- **Static.** A *Warm Phase* to evaluate the computational capacity of each node, and the work is divided accordingly.
- **Dynamic.** Assign a set of *spots* to each node. When a node finishes, it asks for the next group.

Work in progress

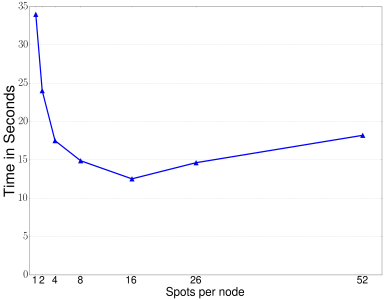
- The parallel nature of the virtual screening problem allows us to parallalize at high level and to extend the calculation to a cluster.
- Use of MPI to assign a set of *spots* to each node in the cluster.

Work modes

- **Static.** A *Warm Phase* to evaluate the computational capacity of each node, and the work is divided accordingly.
- **Dynamic.** Assign a set of *spots* to each node. When a node finishes, it asks for the next group.

Preliminary results

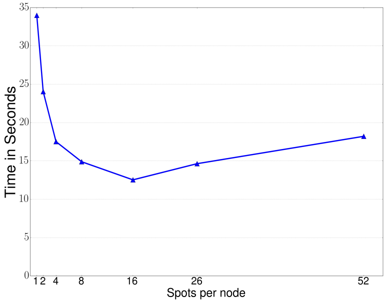
- **Hardware environment.** Four nodes with 2 GeForce GTX 480 and 1 Tesla K20c.
- **Static.** The execution time is *15.24 seconds*.
- **Dynamic.** The best number of *spots* by node is 16, with *12.53 seconds*.



- Metaheuristic M1 with 5 steps of the generic template for metaheuristics.

Preliminary results

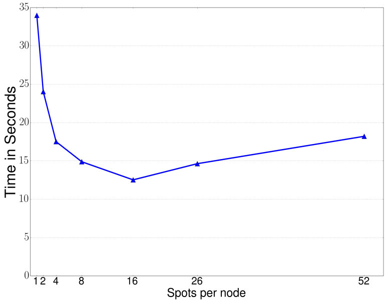
- **Hardware environment.** Four nodes with 2 GeForce GTX 480 and 1 Tesla K20c.
- **Static.** The execution time is *15.24 seconds*.
- **Dynamic.** The best number of *spots* by node is 16, with *12.53 seconds*.



- Metaheuristic M1 with 5 steps of the generic template for metaheuristics.

Preliminary results

- **Hardware environment.** Four nodes with 2 GeForce GTX 480 and 1 Tesla K20c.
- **Static.** The execution time is *15.24 seconds*.
- **Dynamic.** The best number of *spots* by node is 16, with *12.53 seconds*.



- Metaheuristic M1 with 5 steps of the generic template for metaheuristics.

Enhancing Metaheuristic-based Virtual Screening Methods on Massively Parallel and Heterogeneous Systems

Baldomero Imbernón[1], José M. Cecilia [2] and Domingo Giménez [3]

[1−2] Polytechnic School
Catholic University of San Antonio of Murcia (UCAM)
Murcia, Spain

[3] Department of Computing and Systems
University of Murcia
Murcia, Spain

[1]bimbernon@alu.ucam.edu, [2]jmcecilia@ucam.edu, [3]domingo@um.es

March 12, 2016