Acceleration of Monte-Carlo Molecular Simulations on Hybrid Computing Architectures

Braun, Claus; Holst, Stefan; Wunderlich, Hans-Joachim; Castillo, Juan Manuel; Gross, Joachim

Proceedings of the 30th IEEE International Conference on Computer Design (ICCD'12)
Montreal, Canada, 30 September-3 October 2012

doi: http://dx.doi.org/10.1109/ICCD.2012.6378642

Abstract: Markov-Chain Monte-Carlo (MCMC) methods are an important class of simulation techniques, which execute a sequence of simulation steps, where each new step depends on the previous ones. Due to this fundamental dependency, MCMC methods are inherently hard to parallelize on any architecture. The upcoming generations of hybrid CPU/GPGPU architectures with their multi-core CPUs and tightly coupled many-core GPGPUs provide new acceleration opportunities especially for MCMC methods, if the new degrees of freedom are exploited correctly. In this paper, the outcomes of an interdisciplinary collaboration are presented, which focused on the parallel mapping of a MCMC molecular simulation from thermodynamics to hybrid CPU/GPGPU computing systems. While the mapping is designed for upcoming hybrid architectures, the implementation of this approach on an NVIDIA Tesla system already leads to a substantial speedup of more than 87x despite the additional communication overheads.

Preprint

Acceleration of Monte-Carlo Molecular Simulations on Hybrid Computing Architectures

Claus Braun, Stefan Holst and Hans-Joachim Wunderlich
Institute of Computer Architecture and
Computer Engineering
University of Stuttgart
Pfaffenwaldring 47, D-70569, Germany
Email: {braun,holst,wu}@informatik.uni-stuttgart.de

Juan Manuel Castillo and Joachim Gross
Institute of Thermodynamics and
Thermal Process Engineering
University of Stuttgart
Pfaffenwaldring 9, D-70569, Germany
Email: {sanchez,gross}@itt.uni-stuttgart.de

***Abstract*—Markov-Chain Monte-Carlo (MCMC) methods are an important class of simulation techniques, which execute a sequence of simulation steps, where each new step depends on the previous ones. Due to this fundamental dependency, MCMC methods are inherently hard to parallelize on any architecture. The upcoming generations of hybrid CPU/GPGPU architectures with their multi-core CPUs and tightly coupled many-core GPGPUs provide new acceleration opportunities especially for MCMC methods, if the new degrees of freedom are exploited correctly.**

In this paper, the outcomes of an interdisciplinary collaboration are presented, which focused on the parallel mapping of a MCMC molecular simulation from thermodynamics to hybrid CPU/GPGPU computing systems. While the mapping is designed for upcoming hybrid architectures, the implementation of this approach on an NVIDIA Tesla system already leads to a substantial speedup of more than 87x despite the additional communication overheads.

***Keywords*-Hybrid Computer Architectures, GPGPU, Markov-Chain Monte-Carlo, Molecular Simulation, Thermodynamics**

I. Introduction

The massive parallelism of contemporary general-purpose graphics processors (GPGPUs) enables simulation applications at workstation-level, which previously required high-performance computing centers. Moreover, the latest developments in computer architecture show that an increasing number of upcoming processors is going to combine traditional general-purpose multi-core with tightly coupled data-parallel many-core architectures on a single chip [1, 2]. Such hybrid architectures will be able to deliver strong computational performance together with high memory bandwidth and reduced communication overhead within an attractive power envelope. However, the high computational potential of these hybrid architectures can only be exploited by thoroughly adapted or re-designed algorithms. This requires algorithmic partitioning which maps serial or coarse-grained parallel tasks to latency-optimized CPUs and fine-grained, data-parallel tasks to the throughput-optimized computation units.

Stochastic-based simulation methods play an important role since they allow the solution of problems that tend to be very hard to be solved by deterministic algorithms. For search and optimization problems, evolutionary [3–5] and genetic algorithms [6, 7] have been applied. Simulated annealing [8, 9] has been used to localize globally optimal problem solutions. One of the most important classes of such techniques are *Monte Carlo* (MC) [10] methods, which approximate solutions for quantitative problems, with multiple coupled degrees of freedom, by random sampling. The problem targeted in this paper is the parallelization of molecular simulations of the *grand canonical ensemble*, from the field of thermodynamics, on hybrid computing systems. The problem analysis in the next section will show, that these simulations are an instance of a special case of MC methods, the *Markov-Chain Monte-Carlo* (MCMC) simulation. MCMC methods sample from a probability distribution that is based on the construction of a Markov-Chain, which is inherently serial and very difficult to parallelize on contemporary architectures. Being the core of many tasks in thermodynamics, MCMC molecular simulation often forms the major bottleneck, which is typically tackled by coarse-grained parallelization and distribution of simulation instances on clusters or workstation grids. Commonly, this is associated with considerable overhead and costs.

After a review of the related work on parallel MCMC and molecular simulations in section III, an approach to fully utilize the compute power of a hybrid architecture is presented in section IV. The proposed mapping is designed to provide high performance by applying general MCMC parallelization techniques and exploiting some special properties of the target problem at the same time. The experimental results in section V will show a substantial simulation speedup even on commonly available GPU-based accelerators with their well-known communication overheads involved.

II. Problem Definition and Analysis

In many scientific fields, such as material science or chemical engineering, *molecular simulation* is an essential method for the study of molecular structures and processes. A molecular simulation consists in the sampling of a large number of molecular configurations compatible with a particular state of a system in thermodynamic equilibrium. The distribution of these compatible molecular configurations in the phase space, or set of all possible molecular configurations, is called *ensemble*. Depending on which parameters are variable and which of them are held constant, different ensembles can be defined. For example, in the *grand canonical ensemble*, the temperature, volume, and chemical potential of the system are

The authors would like to thank the German Research Foundation (DFG) for financial support of their projects within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

constant, while the energy, pressure, and number of particles are fluctuating properties. By means of statistical mechanic laws, average properties of the ensemble can be related with macroscopic properties of simple gasses and fluids. In order to calculate statistically significant values for the properties of the system, a large number of configurations has to be analyzed, often in the order of billions.

In *Monte-Carlo molecular simulations*, the way to generate new configurations is known as *importance sampling*. Using importance sampling, an initial molecular configuration of the system is modified with what is called a MC move to obtain a new molecular configuration (MC step). The new molecular configuration is accepted or rejected depending on the ensemble probability. If the modification is accepted, the next MC step will be based on the new molecular configuration just generated. If it is rejected, the old configuration is not modified in the current step. This procedure is repeated, so that new configurations are generated in a chain such that the new configuration depends only on the previous configuration (Markov-Chain). In the *canonical ensemble*, new molecular configurations are generated by means of molecule translations and rotations. In the *grand canonical ensemble*, it is also possible to add and delete molecules of the system. In the canonical ensemble, the probability of accepting a new configuration n from a starting configuration o is

$$Pr(o \to n) = \min\{1, e^{-\frac{(E_n - E_o)}{k_B T}}\},$$

where $E_n - E_o$ is the energy difference between the two configurations, k_B the Boltzmann constant and T the temperature of the system. Figure 1 shows the basic principle of the method with a simple example. A known flat area A_{vss} of valid system states is given, and the flat area of interest A_{oi} is to be calculated. The goal is to estimate the relative area $\frac{A_{\text{oi}}}{A_{\text{vss}}}$. The initial system configuration is represented by a point a in the figure. A new configuration is generated by means of a random displacement. If the new point does not belong to A_{vss}, it is rejected immediately. If the point belongs to A_{vss}, the number of times the point falls inside A_{oi} is updated. This step is repeated a large number of times. Finally, the relative area is calculated by

$$\frac{A_{\text{oi}}}{A_{\text{vss}}} = \frac{\text{\# of points that fall in } A_{\text{oi}}}{\text{total \# of points generated in } A_{\text{vss}}}.$$

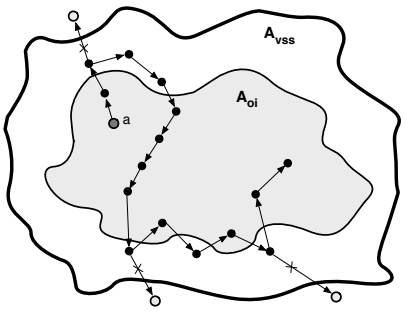

Fig. 1. Sampling of a state space.

The acceptance probability Pr depends on the energies of the systems. The total energy E of a system is the sum of two different energy contributions for all the molecular pairs in the system. The first contribution considers the repulsion between the electrons of different atoms at short distance, and their attractive or also called dispersive interaction at larger distances. These interactions are usually described by a Lennard-Jones potential [11], which is only a function of the distance r_{ij} between the particles i and j:

$$\phi_{LJ}(r_{ij}) = 4\varepsilon \left\{ \left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \right\},$$

where ε and σ are constants that depend on the nature of the atoms i and j.

The second contribution is the electrostatic interaction between point particles, and is described by the Coulomb potential:

$$\phi_{\text{Coul}}(r_{ij}) = \frac{1}{4\pi\varepsilon_0 \frac{q_i q_j}{r_{ij}}},$$

where ε_0 is the dielectric constant of vacuum space, and q_i and q_j the charges of the particles i and j.

The Lennard-Jones potential is a short range potential, as the value of the potential decreases rapidly when the distance between the interacting atoms increases. Therefore, it is a common practice to truncate the potential at a large distance called *cut-off*, so that the potential is set equal to zero at distances larger than the cut-off. The difference between a truncated and a non-truncated potential is generally negligible and the size of the simulated system (i.e. the side length of the simulated cube of matter) can be set to two times the cut-off distance. To mimic the properties of an infinitely large system, the simulation cube is virtually replicated to all sides by proper boundary considerations in the calculation of the interactions between particles.

On the other hand, the Coulomb potential is a long range potential, so truncation is not an option. For this potential it is necessary to calculate the pair interactions for atoms that are separated by a long distance, even in different periodic images of the system. Furthermore, the final result depends on the order in which the interactions are calculated and summed. The most widespread method to accurately calculate electrostatic interactions in molecular simulations is the Ewald summation [12]. With this method, the charges in the system are first conveniently screened, and the sum of the pair interactions is calculated in the Fourier space. This method has the advantage that the sum is quickly convergent, and includes all the periodic images of the system. In the following, we will just review the structure of the computational steps relevant for parallelization. The exact formulas and constants can be found in the original publication. In the Ewald summation there are different contributions to the Coulomb potential. The real energy,

$$\phi_{\text{real}}(r_{ij}) = \frac{1}{4\pi\varepsilon_0} \frac{q_i q_j \ \text{erfc}\,(\sqrt{\alpha} r_{ij})}{r_{ij}},$$

where α is an adjustable parameter, and erfc is the complementary error function. For computing the long-range term called Fourier energy F, in a first step C complex coefficients

are calculated as:

$$c_k = \sum_{i=1}^{N} f\,(i,k) \text{ with } 1 \leq k \leq C.$$

f is a complex-valued function, which depends only on the location of the atom i and the coefficient index k. In the next step, the coefficients are used to calculate the final Fourier energy:

$$F = \sum_{k=1}^{C} g[k] \cdot |c_k|^2,$$

with $g[k]$ being a table of constants.

Both the short-range electrostatic interaction and the Lennard-Jones potential are accumulated over all particle pairs to obtain the pair energy:

$$P = \sum_{i,j \in A, i \neq j} \phi_{LJ}(r_{ij}) + \phi_{real}(r_{ij}),$$

and the total energy of a system is just $E = P + F$.

The performance of Monte-Carlo based molecular simulation depends on two factors: The number of accepted MC steps per computing time, and the overall change in the configuration of the system within each step. Large changes in the system configuration are desirable in each step, as the sampling of the ensemble will be faster. The drawback is that the acceptance probability will be very low. Small changes would lead to a large number of accepted steps, but the sampling of the ensemble will be very poor.

In an MC step only small random modifications are possible in the configurations. Typically, just one particle is changed at a time to obtain reasonable acceptance rates. The magnitude of change in the system in every MC move is easily adjusted by choosing a maximum translation distance and a maximum rotation angle. These parameters are commonly chosen in a way, that an acceptance rate of 0.5 is achieved. It was suggested [13] that this acceptance rate gives the best overall performance for serial implementations.

While typical MC simulations have only a relatively small number (usually no more than 10^3–10^4) of particles in the system, still a massive amount of Monte-Carlo moves are necessary to sufficiently sample the compatible configurations of the ensemble.

III. State of the Art

From the known parallelization approaches for Monte-Carlo based algorithms [14], the speculative computation of future moves [15] is the most promising for an adaption to the given problem. By creating multiple instances of the molecular system and by evaluating the energies of these systems in parallel, a speedup is achieved without compromising the sequential dependencies of the Markov-Chain. This concept has already been applied to MC based molecular simulation on parallel CPUs [16, 17]. However, for bringing this approach to tightly coupled GPGPU architectures, additional aspects like data-parallelism and scarce memory bandwidth have to be taken into account.

Quite similar to MC molecular simulation are *molecular dynamics* (MD) algorithms. MD algorithms compute for each configuration the forces between all particles and move them according to Newton's equations of motion to obtain a new configuration for the next time step. GPGPU implementations of MD algorithms [18–21], as well as similar simulation algorithms [22, 23] often use the following parallelization concepts:

- *Spatial decomposition*: When the particle interactions are local, particles that are further apart can be simulated in parallel.
- *Data-parallel energy calculation*: The same computations are performed for all the particles or particle-pairs in the system, hence this step is particularly suitable for data-parallel implementation.

For Monte-Carlo simulations of systems with electrostatic interactions, long range forces are dominating, hence the spatial decomposition approach becomes inefficient. Nevertheless, the data-parallel calculation of the energies is applicable.

In [24], speculative computations have been applied on GPGPUs, however, this work only considered the Lennard-Jones part. To the best of our knowledge, no approach is available to leverage the performance of hybrid architectures for MC molecular simulation by combining speculative computation and complete parallel energy evaluation.

IV. MCMC Molecular Simulation on Hybrid Architectures

The presented algorithm efficiently combines and extends the two parallelization concepts of data-parallel energy calculations and speculative computing from [15]. These concepts complement each other and preserve the sequential path of the original algorithm. In the following, we will focus on single-atom particles for sake of simplicity. However, the described discussion is easily extended to molecules.

A. Energy Evaluation

Let A be a set of atoms in the molecular system. Each atom $i \in A$ is described by its location and orientation in the simulation box, and a set of constant model parameters such as mass and charge. In each MC step, the energy difference caused by a change in the simulation box has to be calculated. The system can be changed by removal of an atom $o \in A$ and the addition of a new atom n at a different location and/or orientation (figure 2):

$$A_n = (A_o \setminus \{o\}) \cup \{n\}.$$

We will note this change in the following as $o \triangleright n$. The energy difference $\Delta E_{o \triangleright n}$ for this change is calculated with the expression:

$$\Delta E_{o \triangleright n} = E_n - E_o = P_n - P_o + F_n - F_o.$$

The change in the pair energy can be computed by just considering the interactions of o and n with all other atoms in the system:

$$P_n - P_o = \sum_{i \in A, i \neq o} \phi_{LJ}(r_{in}) - \phi_{LJ}(r_{io}) + \phi_{real}(r_{in}) - \phi_{real}(r_{io}).$$

These $O(|A|)$ terms can all be calculated in parallel.

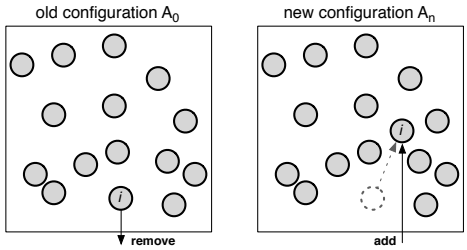

Fig. 2. Translation of atom by removal and addition.

Let U be the set of changes to the molecular system. A kernel launch $kernel\langle A, U\rangle()$ generates $|A| \cdot |U|$ threads on the GPGPU and each of these threads evaluates a different pair of $\langle a, o \triangleright n\rangle$ with $a \in A$ and $o \triangleright n \in U$.

Algorithm 1 Kernel k_pair for pair energy calculation

```
k_pair⟨a, o ▷ n⟩()
begin
  e[tid] = 0
  if (a ≠ o)
    float dx = global_load(a.x) − global_load(n.x)
    float dy = global_load(a.y) − global_load(n.y)
    float dz = global_load(a.z) − global_load(n.z)
    dx = min_image(dx)
    dy = min_image(dy)
    dz = min_image(dz)
    float r = dx² + dy² + dz²
    if (r > 0)
      e[tid] = φ_LJ(r) + φ_real(r)
    fi
  fi
  e = Σ_tid e[tid]
  if (tid == 0)
    global_write(e)
  fi
end
```

Algorithm 1 shows the kernel for the pair energy calculation in pseudo-code. tid is the thread id accessing the common shared memory array $e[\,]$. All other variables are local to each thread. First, the coordinates of the atom are loaded from the GPGPU's global memory. These accesses are organized in a way to provide maximum coalescing, i.e. they are coordinated in a way that a minimum number of memory transactions is performed by the hardware. The function *min_image(d)* adds or subtracts the side length l of the simulation box, so that its return value is $|d'| < \frac{l}{2}$. Then, the distance and the energies are computed. A partial summation is performed and the result is written to the global memory.

The term F contains the rest of the contributions to the energy given by the Ewald summation. In case of the Fourier energy, each of the $f\,(i,k)$ summands can be evaluated in parallel. However, the memory access time required for reading the positions and the model parameters of every atom limits the overall performance. To mitigate this problem, each thread calculates all the summands $f\,(\mathbf{r}_i, k)$ for a given atom, which improves the ratio between memory accesses and arithmetic operations.

Algorithm 2 shows the kernel to generate the partial sums for all coefficients c_k. $er[\,]$, $ei[\,]$ are arrays in the GPGPU's shared memory. In contrast to k_real, this kernel works on atom locations and not on distances between atoms. The locations are loaded either from the common molecular system or from the update data passed to the kernel. After loading the necessary data, the amount of local computation is maximized by calculating all coefficients in a loop and writing the partial sums to the global memory. A further reduction of the data is not possible at this point, since this would require the complete sums of all c_k and not only the partial sums. Hence, a second kernel is necessary.

Algorithm 2 Kernel $k\_fourier$ for Fourier-part energy calculation

```
k_fourier⟨a, o ▷ n⟩()
begin
  er[tid] = 0
  ei[tid] = 0
  if (a == o) then
        x = global_load(n.x)
        y = global_load(n.y)
        z = global_load(n.z)
  else
        x = global_load(a.x)
        y = global_load(a.y)
        z = global_load(a.z)
  fi
  for (k = 0; k < C; k++)
    (er[tid], ei[tid]) = f((x, y, z), k)
    er = Σ_tid er[tid]
    ei = Σ_tid ei[tid]
    if (tid == 0)
      global_write(er, ei)
    fi
  end
end
```

Algorithm 3 Kernel $k\_fourier\_sum$ for Fourier-part summation

```
k_fourier_sum⟨k⟩(psums)
begin
  e[tid] = 0
  er = 0
  ei = 0
  for (s = 0; s < psums; s++)
    (er, ei) = (er, ei) + global_load(c_k[s])
  end
  e[tid] = e[tid] + ffactor[k] · (er² + ei²)
  e = Σ_tid e[tid]
  if (tid == 0)
    global_write(e)
  fi
end
```

The kernel described in algorithm 3 is launched with $k\_fourier\_sum\langle\{0, ..., \#\text{coeff} - 1\}\rangle(\#\text{ partial\_sums})$. After summing up all the partial sums for a coefficient $\mathbf{c}\,(\mathbf{k})$, a partial sum of F is calculated and stored.

All energy contributions consist of an infinitely parallelizable part (the individual parts of the sum) and a reduction operation with $\log_2(n)$ steps. Hence, the possible speedup is in the order of

$$\text{speedup} = \frac{n}{1 + \log_2(n)}.$$

B. Speculative Computation

Given an initial molecular configuration, in every MC step $m \geq 1$ different new configurations, so called mutants, are generated by performing a single MC move. The energies of

all these mutants are calculated in parallel. This procedure has multiple advantages:

- The number of mutants in all steps are the same, and the computing effort for the energy calculations are roughly the same.
- There is limit to the amount of data that is shared among the parallel instances, which reduces memory bandwidth demand significantly.
- MC moves with low acceptance probability and large displacements can be used, improving the sampling of the ensemble.

Let E_o be the energy of the initial molecular configuration and E_j the energy of the mutant j. For every mutant we calculate the factor:

$$w_j = e^{-\frac{(E_j - E_o)}{k_B T}}.$$

According to [17], we select the mutant j as the new configuration of this MC step with a probability given by:

$$Pr(j) = \frac{w_j}{\sum_{i=1}^{m} w_i}.$$

Finally, the molecular configuration of the selected mutant j is accepted or rejected with the probability:

$$Pr = min(1, w_j)$$

C. Alteration of Acceptance Probabilities

In the parallel algorithm, it is convenient to adjust the maximum displacements to obtain a low acceptance rate. The reason is that, the lower the acceptance probability, the higher the quality of an accepted move. In addition, the more mutants are simulated in parallel, the smaller the acceptance rate per mutant can be. Consequently, to reduce the acceptance rate per mutant, the maximum translation distance and rotation angle are increased accordingly.

D. MCMC Simulation System

The overall flow of a Monte-Carlo cycle with two mutants is shown in figure 3. In *mutant preparation*, the CPU generates

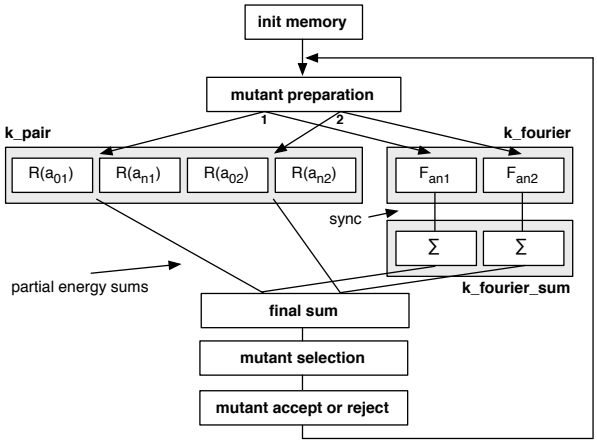

Fig. 3. Flow of a Monte Carlo simulation cycle with two mutants

$m \geq 1$ mutants randomly. Then, the energy differences for each mutant are calculated with the help of the three kernels *k_pair*, *k_fourier*, *k_fourier_sum*, which are executed on the GPGPU. The kernels *k_pair* and *k_fourier_sum* perform an efficient partial summation of the energies in the shared memory of the GPGPU. Only a limited number of p threads can access such a common shared memory, leading to $\lceil n/p \rceil$ partial sums. The CPU performs the final summation and checks the acceptance of the mutants. If a mutant is accepted, the loop proceeds with a new molecular system. To reduce the memory bandwidth required for communication between the CPU and the GPGPU, the mapping transfers only the differences between molecular configurations and not the full information.

V. Experimental Results

To evaluate the speedup of the proposed algorithmic mapping of the Markov-Chain Monte Carlo molecular simulation, a state-of-the-art serial production code written in Fortran has been compared on an Intel Xeon X5680 CPU with 3.33GHz against the hybrid mapping written in C with CUDA extensions. For reference purposes, a version of the code has been mapped to a standard multi-core CPU using MPI.[1] This code showed a purely linear scaling behavior on up to 4 CPU cores with a speedup of at most 3.9x. For the hybrid mapping, the data-parallel parts of the algorithm were executed on an Nvidia Fermi GPGPU whith 448 processing cores running at 1.15 GHz. The memory consumption of all considered test cases was in the order of a few MBytes and is therefore not discussed any further.

As a typical example of a molecular simulation, *Single-Point-Charge* (SPC) water [25] was chosen, and both, Lennard-Jones energy and electrostatic interactions were considered. With p being the number of molecules in the system, $n = 3p$ atoms were simulated. During the experiments, random translations and rotations of molecules were evaluated. A profiling of the simulation execution time showed, that the simulation runtime is determined by the energy evaluation after each Monte Carlo move. In detail, 80% to 90% of the execution time were consumed by the GPGPU energy kernels. The extension to different types of MC moves will not have any impact on the overall performance.

Table I shows the overall simulation runtimes for molecular systems of various sizes. Each molecular system was initialized by placing all water molecules on a regular lattice in the simulation box. Then, 10^4 Monte Carlo trials were executed with an acceptance probability of roughly 50% to equilibrate the system. After equilibration, the simulation time was measured while evaluating another 10^4 translation trials. In the hybrid implementation, a single mutant was evaluated per MC cycle. It can be seen from the table, that a substantial speedup is achieved for all molecular systems. The speedup increases with the number of molecules in the system as the massive computing power of the GPGPU becomes better and better exploited. Molecular systems with a small number of molecules can not exploit the full parallelism provided by the GPGPU. For these systems, multiple mutants were generated per MC cycle to improve the acceptance rates in the overall simulation. The effectiveness of this approach from

[1] thanks to A. Kohler, Institute of Computer Architecture and Computer Engineering, University of Stuttgart.

Molecules	Fortran	Hybrid GPU/CPU	Speedup
256	45s	4.9s	**9.2x**
512	91s	5.1s	**18x**
1024	189s	5.7s	**33x**
1536	279s	6.2s	**45x**
2048	374s	6.3s	**59x**
3072	562s	7.9s	**71x**
4096	755s	8.7s	**87x**

TABLE I
SIMULATION RUNTIMES FOR DIFFERENT NUMBER OF WATER MOLECULES IN THE SYSTEM AND 10^4 MONTE CARLO TRIALS.

the performance standpoint can be evaluated by measuring the slowdown in the simulation time while evaluating $m > 1$ mutants in parallel during a MC cycle.

Table II shows the simulation runtimes when multiple mutants were evaluated in each MC cycle. The percentages in paranthesis indicate the amount of simulation speed maintained in spite of the additional workload for the GPGPU. We observed, that for example with 256 molecules, doubling the workload ($m = 2$) for the GPGPU had negligible impact on the overall simulation runtime. With roughly the same simulation time, two molecular configurations were available in each cycle to choose from. The probability to accept a system in a MC cycle was therefore higher and the algorithm progressed faster. The benefit for particular simulation cases can be easily calculated using the runtime values presented here and the acceptance rates in the particular cases. As expected, no benefit was achieved for large systems with more than 4096 molecules. The simulation slowed down according to the number of mutants, which shows that the parallelism of the GPGPU was already fully exploited by parallel energy calculation.

Molecules	$m = 1$	$m = 2$	$m = 3$	$m = 4$
256	4.9s	4.9s (100%)	5.4s (92%)	5.4s (91%)
512	5.0s	5.6s (90%)	6.1s (83%)	6.3s (80%)
1024	5.7s	6.2s (92%)	7.8s (73%)	8.7s (66%)
1536	6.1s	7.9s (78%)	9.2s (66%)	14.0s (44%)
2048	6.3s	8.8s (72%)	14.0s (45%)	16.2s (39%)
3072	7.9s	13.9s (57%)	17.3s (46%)	23.5s (34%)
4096	8.7s	16.2s (54%)	23.7s (37%)	30.6s (29%)

TABLE II
SIMULATION RUNTIMES FOR MULTIPLE MUTANTS IN EACH OF THE 10^4 MONTE CARLO TRIALS.

VI. CONCLUSION

In this paper we presented a new method for the parallel mapping and implementation of Markov-Chain Monte-Carlo molecular simulations on hybrid CPU-GPGPU systems. The mapping is characterized by data-parallel energy caclulations and speculative computations in each Monte-Carlo step. Furthermore, the different simulation tasks are thoroughly partitioned to exploit the high single thread performance of the CPU and the massive data-parallelism of the GPGPU. The computations are arranged in a way that allows maximum data re-use and a substantial reduction of communication overhead. The proposed mapping is able to directly utilize the different architectural characteristics of hybrid computing systems. It was shown that the parallel mapping achieves a speedup of more than 87x. This significant speedup enables MCMC molecular simulations at workstation-level and the investigation of problem sizes, which previously required computing clusters or grid-based systems.

REFERENCES

[1] A. Branover, D. Foley, and M. Steinman, "AMD Fusion APU: Llano", *IEEE Micro*, vol. 32, no. 2, pp. 28 –37, march-april 2012.

[2] M. Daga, A. Aji, and W. chun Feng, "On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing", in *2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*, July 2011, pp. 141 –149.

[3] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, L. J. Fogel, A. J. Owens, and M. J. Walsh, Eds. John Wiley & Sons, Inc. New York, USA, 1966.

[4] I. Rechenberg, *Evolutionsstrategien: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Friedrich Frommann Verlag, 1973.

[5] H. P. Schwefel, *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc. New York, USA, 1981.

[6] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[9] V. Černý, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm", *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, 1985.

[10] N. Metropolis, "The Beginning of the Monte Carlo Method", *Los Alamos Science*, vol. 15, no. 584, pp. 125–130, 1987.

[11] J. E. Jones, "On the Determination of Molecular Fields. I. From the Variation of the Viscosity of a Gas with Temperature", *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 106, no. 738, pp. 441–462, 1924.

[12] P. P. Ewald, "Die Berechnung optischer und elektrostatischer Gitterpotentiale", *Annalen der Physik*, vol. 369, no. 3, pp. 253–287, 1921.

[13] G. Heffelfinger and M. Lewitt, "A comparison between two massively parallel algorithms for Monte Carlo computer simulation: An investigation in the grand canonical ensemble", *Journal of Computational Chemistry*, vol. 17, no. 2, pp. 250–265, January 1996.

[14] D. R. Greening, "Parallel simulated annealing techniques", *Physica D: Nonlinear Phenomena*, vol. 42, no. 1-3, pp. 293 – 306, 1990.

[15] E. Witte, R. Chamberlain, and M. Franklin, "Parallel Simulated Annealing Using Speculative Computation", *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 483–494, October 1991.

[16] D. M. Jones and J. M. Goodfellow, "Parallelization Strategies for Molecular Simulation using the Monte Carlo Algorithm", *Journal of Computational Chemistry*, vol. 14, no. 2, pp. 127–137, 1993.

[17] K. Esselink, L. D. J. C. Loyens, and B. Smit, "Parallel Monte Carlo Simulations", *Physical Review E*, vol. 51, no. 2, pp. 1560–1568, February 1995.

[18] J. E. Stone *et al.*, "GPU-accelerated molecular modeling coming of age", *Journal of Molecular Graphics and Modelling*, vol. 29, no. 2, pp. 116–125, 2010.

[19] J. A. van Meel *et al.*, "Harvesting graphics power for MD simulations", *Molecular Simulation*, vol. 34, no. 3, pp. 259–266, 2008.

[20] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units", *Journal of Computational Physics*, vol. 227, no. 10, pp. 5342 – 5359, 2008.

[21] J. Yang, Y. Wang, and Y. Chen, "Gpu accelerated molecular dynamics simulation of thermal conductivities", *Journal of Computational Physics*, vol. 221, no. 2, pp. 799–804, 2007.

[22] E. Elsen *et al.*, "N-Body Simulations on GPUs", Stanford University, Tech. Rep., 2007.

[23] Y. Frishman and A. Tal, "Multi-Level Graph Layout on the GPU", *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1310 –1319, Nov.-Dec. 2007.

[24] J. Kim *et al.*, "Molecular Monte Carlo Simulations Using Graphics Processing Units: To Waste Recycle or Not?" *Journal of Chemical Theory and Computation*, vol. 7, no. 10, pp. 3208–3222, October 2011.

[25] H. J. C. Berendsen *et al.*, *Intermolecular Forces*, 1981, ch. Interation Models for Water in Relation To Protein Hydration, pp. 331–342.