Università
della
Svizzera
italiana

**Facoltà
di scienze
informatiche**

Particle Simulations with OpenACC: Speedup and Scaling

Overview of mathematical models, simulation used, and OpenACC

Samuel A. Cruz Alegría, Alessandra M. de Felice, Hrishikesh R. Gupta

(University of Lugano)

March 15, 2018

Motivation & Goals

- Particle Simulations are essential for visualization of behaviour of physical systems
  - Molecular Dynamics
  - Celestial body simulations
  - Fluid dynamics
- Realistic computations have massive particle numbers and therefore immense computational costs
- Multi-core simulation tools are essential for runtime optimization
- Therefore our **goal** is to optimize particle simulation runtime using OpenACC parallelization and to observe scaling with an increasing number of particles.

Mathematical Models

Molecular dynamics *simulations* consist of the *numerical* step-by-step solution of the classical equations of motion. For a simple atomic system, they may be written as follows:

$$\begin{gathered} m_i \cdot \ddot{r}_i = f_i, i = 1, \ldots, N, \\ f_i = -\frac{\partial E_p}{\partial r_i}, \end{gathered} \tag{1}$$

where

- N is the number of atoms.
- $E_p(r^N)$ represents the potential energy.
- $r^N = (r_1, r_2, \ldots, r_n)$ represent the $3 \cdot n$ *atomic coordinates*.

Force Fields

In biochemical systems, commonly-used force fields model the *potential energy* function as the sum of bonded, van der Waals, and electrostatic (Coulomb) energy:

$$E_p = E_{\text{bonded}} + E_{\text{non-bonded}}. \qquad (2)$$

The potential is a function of the positions of *all* the atoms in the simulation. The *force* on an atom is the negative gradient of this potential at the position of the atom:

$$f = -\nabla E_p. \qquad (3)$$

Total Energy

We can view equation the constituents of equation (??) as follows:

$$E_{\text{bonded}} = E_{\text{str}} + E_{\text{bend}} + E_{\text{tor}}. \tag{4}$$

$$E_{\text{non-bonded}} = E_{\text{Coul}} + E_{\text{vdW}}. \tag{5}$$

The bonded interactions are named as such because the atoms involved must be directly bonded or bonded to a common atom.

Bonded Interactions

We can model the bonded energy as follows:

$$E_{\text{bonded}} = E_{\text{str}} + E_{\text{bend}} + E_{\text{tor}}, \tag{6}$$

where

- E_{str} is the energy required to *stretch* or compress a given bond.
- E_{bend} is the energy required to *bend* a bond from its equilibrium angle, θ_{eq}.
- E_{tor} is the energy of *torsion* needed to rotate about bonds. These are most relevant in single bonds because double and triple bonds are too rigid to permit rotation.

Stretch/Compression Energy

A bond can be thought of as a spring having its own equilibrium length, r_{eq}, and the energy required to stretch or compress it can be approximated by the Hookian potential for an ideal spring:

$$E_{\text{str}} = \frac{1}{2} \sum_{\text{bonds}} k^s_{ij} \left(r_{ij} - r_{\text{eq}} \right)^2 , \tag{7}$$

where

- k^s_{ij} is the stretching force constant for the bond.
- r_{ij} is the distance between atoms i and j.

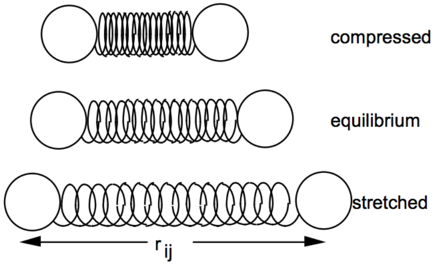


Figure: Bond Stretching.

Bending Energy

E_{bend} is the energy required to *bend* a bond from its equilibrium angle, θ_{eq}. Once more, this system can be modeled by a spring, and the energy is given by the Hookian potential with respect to the angle:

$$E_{\text{bend}} = \frac{1}{2} \sum_{\text{bend angles}} k^{b}_{ijk} \left(\theta_{ijk} - \theta_{eq}\right) , \tag{8}$$

where

- k^{b}_{ijk} is the bending force constant.
- θ_{ijk} is the instantaneous bond angle.

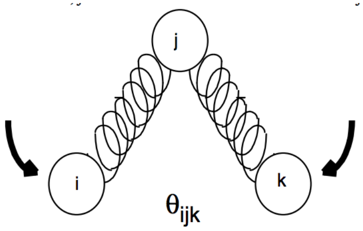


Figure: Bond Bending.

Torsion Energy

E_{tor} is the energy of *torsion* needed to rotate about bonds. Torsional interactions can be modeled by the following potential:

$$E_{\text{tor}} = \frac{1}{2} \sum_{\text{torsion angles}} \left(k_{tor,1}(1+\cos\phi) + k_{tor,2}(1+\cos 2\phi) + k_{tor,3}(1+\cos 3\phi)\right), \qquad (9)$$

where

- The angle ϕ is the dihedral angle about the bond.
- The constants $k_{tor,1}, k_{tor,2}, k_{tor,3}$ are the torsional constants for one-fold, two-fold, and three-fold rotational barriers, respectively.

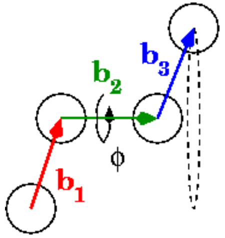


Figure: Bond Torsion.

Bonded Interactions

Putting it all together, we have the following for the bonded interactions:

$$\begin{aligned} E_{\text{bonded}} &= \frac{1}{2} \sum_{\text{bonds}} k^s_{ij} \left(r_{ij} - r_{\text{eq}} \right)^2 \\ &+ \frac{1}{2} \sum_{\text{bend angles}} k^b_{ijk} \left(\theta_{ijk} - \theta_{eq} \right) \\ &+ \frac{1}{2} \sum_{\text{torsion angles}} \left(k_{tor,1}(1 + \cos\phi) + k_{tor,2}(1 + \cos 2\phi) + k_{tor,3}(1 + \cos 3\phi) \right), \end{aligned} \tag{10}$$

Non-Bonded Interactions

We can model the non-bonded energy as follows:

$$E_{\text{non-bonded}} = E_{\text{Coul}} + E_{\text{vdW}}, \tag{11}$$

where

- E_{Coul} is the electrostatic energy resulting from charged molecules.
- E_{vdW} is the energy resulting from the attraction of intermolecular forces, i.e., between molecules.

Non-Bonded Interactions

In order to model the *van der Waals forces*, we can use the Lennard-Jones function:

$$E_{\text{vdW}} = \sum_{i=1}^{N-1} \sum_{j>i} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right], \qquad (12)$$

where

- r_{ij} is the *distance* between atoms i and j.
- σ_{ij} is the finite distance at which inter-particle potential is zero.
- ϵ_{ij} is the depth of the potential well, which is a region surrounding a local minimum of potential energy.

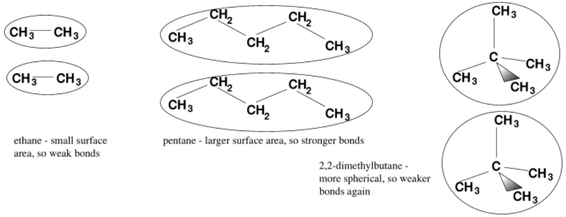

CH_3—CH_3
CH_3—CH_3
ethane - small surface area, so weak bonds
CH_3 CH_2 CH_2 CH_2 CH_3
CH_3 CH_2 CH_2 CH_2 CH_3
pentane - larger surface area, so stronger bonds
2,2-dimethylbutane - more spherical, so weaker bonds again
CH_3 C CH_3 CH_3 CH_3
CH_3 C CH_3 CH_3 CH_3

Figure: van der Waals Forces.

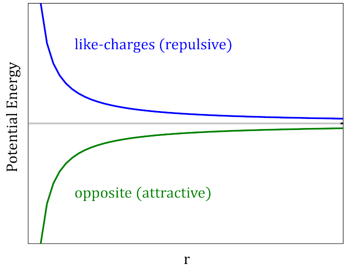
Non-Bonded Interactions

If charges are present, we can model the electrostatic energy due to the atomic charges using the Coulombic potential function:

$$E_{\text{Coul}} = \sum_{i=1}^{N-1} \sum_{j>i} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}}, \tag{13}$$

where

- r_{ij} is the distance between atoms i and j.
- q_i and q_j are the charges of atoms i and j, respectively.
- $\epsilon_0 \approx 8.85 » 10^{-12} m^{-3} kg^{-1} s^4 A^2$ (permittivity of free space).



Updating Position

Once more, once we have computed the potential energy, we can compute the force on an atom as the negative gradient of this potential at the position of the atom:

$$f_i = -\frac{\partial E_p}{\partial r_i}, \tag{14}$$

and we can then proceed to update the position of the atom based on a given algorithm, such as $r_i^{k+1} = r_i^k + 0.5 \cdot \Delta t \cdot f$.

Simulation

- **Idea**
  - Given, initial positions, masses and other parameters (Forces) for the particles future trajectory can be computed numerically for each particle.
  - Time step is thus composed of two parts:
    - Compute forces on all particles
    - Update positions

Force Field

- It is the model of potential energy and forces that acts between particles.
- Commonly used force fields model the potential energy function as the sum of:

$$E_p = E_{bonded} + E_{Coul} + E_{VdW} \tag{15}$$

- Which is summation of Bonded Energy, Electrostatic Energy and Van der Waals Energy.

Computing the forces (Short Range)

- Mostly for non-bonded forces: electrostatic interactions and Van der Waals interactions.
- Naïve approach:
  - Examine all other particles and compute their distance to particle ‘i’.
  - For ‘n’ particles, the complexity of this approach is O(n^2), which is equivalent to computing forces between all pairs of particles.
- Cell Lists:
  - Particle space is put in a grid.
  - The forces is computed on particle ’i’ in a grid cell with respect to 8 adjacent cells.
  - Complexity is O(n x average no. of particles in 9 cells).

Computing the forces (Long Range)

- In order to avoid complexity O(n^2) using naïve approach, following are some accurate methods.
- **Particle Mesh Method:**
  - A system of particles, converted into mesh or grid values.
  - we exploit the Poisson equation:

$$\nabla^2 \phi = -\frac{1}{\epsilon}\rho \tag{16}$$

  - Relates potential ϕ to charge density ρ, where $\frac{1}{\epsilon}$ is a constant.
  - Charges are assigned to mesh point and equation (16) is solved to arrive at the potential on the mesh.
- The force is the negative gradient of the potential.
- Fast methods such as multi-grid method and FFT are used to solve Poisson's equation.
- **Ewald Method:**
  - The force is split between short-range and long-range.
  - Short-range part is computed using particle-particle methods.
  - Long-range part is computed using Fourier Transforms.

Parallel Approaches

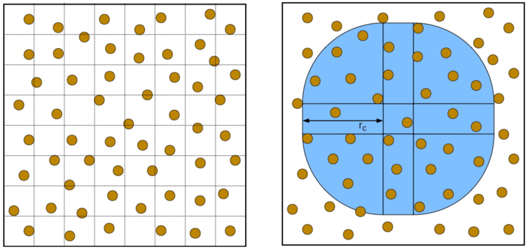
- **Atom Decomposition:**
  - Each particle is assigned to one processor, which is responsible for computing the particle's forces and updating its position for the entire simulation.
  - Each processor communicates with all other processors to share updated particle positions.
  - An Force-matrix is constructed, which is an n-by-n matrix; the rows and columns are numbered by particle indices.
  - Each point in force-matrix computes, force on 'i' due to particle 'j'.
  - When cutoffs are used, the matrix is sparse.

Algorithm 1 Atom decomposition time step

```
1: send/receive particle positions to/from all other processors
2: (if nonbonded cutoffs are used) determine which nonbonded forces need to be computed
3: compute forces for particles assigned to this processor
4: update positions (integration) for particles assigned to this processor
```

Parallel Approaches

- **Spatial Decomposition**
  - Space is divided into cells and each cell is assigned to a processor, responsible for computing the forces on particles that lie inside the cell.
  - Since particles move during the simulation, the assignment of particles to cells changes as well.
  - Cutoff radius decides import region and the given cell must import positions of particles lying in this region to perform its force calculation.



(a) Decomposition into 64 cells. (b) Import region for one cell.

Parallel Approaches

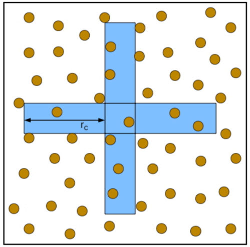
- **Neutral Territory Method:**
  - Particles are assigned to processors according to a partitioning of space.
  - Each processor computes the forces between two sets of particles, these particles may be unrelated to the particles that have been assigned to the processor.
  - For example: The given processor is assigned the computation of forces between particles lying in the horizontal bar with particles lying in the vertical bar.

Algorithm 5 Spatial decomposition time step

```
1: send positions needed by other processors for particles in their import regions; receive positions for
   particles in my import region
2: compute forces for my assigned particles
3: update positions (integration) for my assigned particles
```

Parallel Approaches

- These two regions thus form the import region.



- After the forces are computed, the given processor sends the forces it has computed to the processors that need these forces for integration.

OpenACC

”The execution of CUDA with the simplicity of OpenMP”

- **What is OpenACC?**
  - Accelerator based parallel programming
  - Like OpenMP, OpenACC uses directive based programming
    - Compiler generates the parallel code
  - Used with C/C++ and Fortan
  - OpenACC uses GPU (or multicore CPU) acceleration incrementally

OpenACC

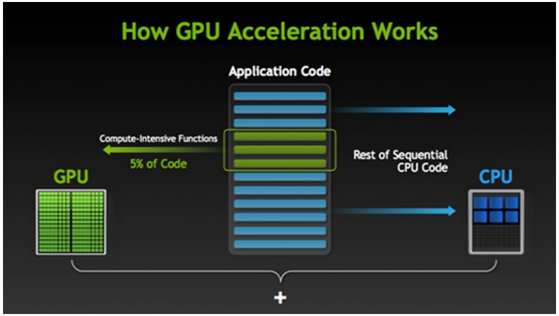
- **Advantages**
  - High level
    - The compiler will ultimately make the decision as to where this memory will be allocated and how it is addressed
    - Programmer only deals with abstractions of memory and not memory itself
  - Performance portable
  - Single source - No forking off a separate GPU code
  - Efficient - favorable comparison to low-level implementations of same algorithms
  - Incremental - can port and tune parts of their application as resources and profiling dictates.
    - This is the disadvantage of CUDA
  - Implicit use of accelerators
- **Disadvantages**
  - Some cases performance is 50% lower than CUDA implemented parallelization
  - Complexity of implementation increases with complexity of loops
    - This means that the time-saving advantage may be negated
  - Lack of a programming interface for the shared memory

Where is OpenACC used?

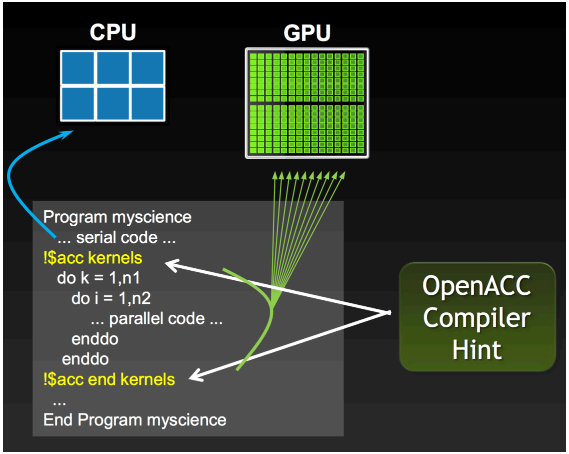
- Matrix Matrix Multiplication - 4 directives = 16x faster
- Derivative Valuation - Few hours = 70x faster
- N-body/Particle Simulations = 1 week = 3x faster
- Real-time Image extraction - 3 directives = 4.1x faster
- Fluid Dynamics and Fuel Combustion = 4x Faster with less than 1% of code modified
- DNA sequence Analysis - 4 directives = 16x faster
- Designing circuits for quantum computing = 1 week = 40x faster

GPU Acceleration

- GPU vs CPU performance
  - CPU consists of a few cores optimized for sequential serial processing
  - GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.
  - GPU-accelerated computing offloads compute-intensive portions of the application to the GPU
  - Remainder of the code still runs on the CPU.


How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
Rest of Sequential CPU Code
GPU
CPU
+

How OpenACC works?



```
Program myscience
 ... serial code ...
!$acc kernels
  do k = 1,n1
    do i = 1,n2
      ... parallel code ...
    enddo
  enddo
!$acc end kernels
 ...
End Program myscience
```

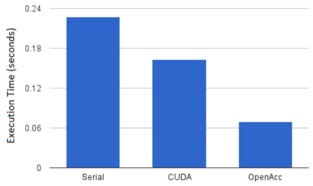
- Parallelizeable code is moved from the host CPU to an accelerator device (GPU) for execution
- Internal Control Variables (ICVs) - determine the type of accelerator device and which accelerator device
- The environment variables (can be altered) $\rightarrow$ directive can be applied to certain accelerators or all accelerators
  - device type
  - device number

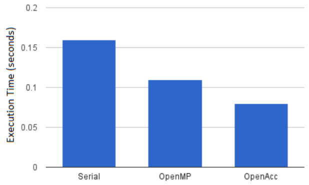
Directives

- #pragma acc parallel loop directive
  - Caution: the compiler will attempt to run the code in the for loop in parallel even if it is not safe to do so
  - This directive should be used with caution only when the underlying algorithm is completely understood
- OpenACC kernel construct
  - Make the code within the block parallel by creating accelerator kernels where possible and safe to do so.
    - safe way to make your code parallel
    - Disadvantage = generated code may not always be the most efficient code possible.
  - Kernels generated will run on the GPU

Performance comparison

- OpenACC and CUDA
  - simple matrix computation - asked to compiler to run kernels on the device where possible
- OpenACC and OpenMP
  - computes the number of brights spots in a matrix that represents pixel brightness in an image.





Conclusion

- In conclusion, N-body simulations are considered of great importance and it is thus essential to execute simulations as efficiently as possible.
- Accelerator-based parallelization like OpenACC may prove to be a more efficient means to parallelize particle simulations because it is evident that this has not been explored thoroughly.

References

- Farber, R., 2016.Parallel programming with OpenACC. Newnes.
- Gonzales, R., Martin, M., Mittow, N., and Rasmuss, R.,2016, An Introduction to OpenAcc.ECS 158 Final Project.
- Li, X., Shih, P.C., Overbey, J., Seals, C. and Lim, A., 2016. Comparing programmer productivity in OpenACC and CUDA: an empirical investigation.International Journal of Computer Science, Engineering and Applications (IJCSEA),6(5), pp.1-15.
- Memeti, S., Li, L., Pllana, S., Kołodziej, J. and Kessler, C., 2017, July. Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption. InProceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing(pp. 1-6). ACM.
- Urbanic, J., 2013. Introduction to Directive Based Programming.
- OpenACC Programming and Best Practices. Guide http://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0.pdf
- http://www.nvidia.com/object/what-is-gpu-computing.html
- Allen, M.P., 2004. Introduction to molecular dynamics simulation. Computational soft matter: from synthetic polymers to proteins, 23, pp.1-28.
- Eijkhout, V., 2014. Introduction to High Performance Scientific Computing. Lulu. com.