OpenACC - Performance and Productivity

Paul Springer

Aachen Institute for Advanced Study in
Computational Engineering Science

Aachen, 14.03.13 – PPCES 2013

Outline

Molecular Dynamics Simulation

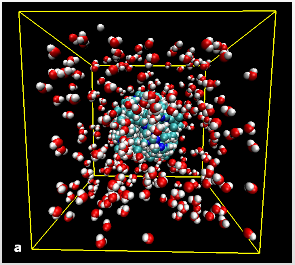
Molecular Dynamics

- System of N interacting particles
    - E.g.: Atoms, molecules, planets
- Simulate their motion
- Detect chemical reactions
- Forces of particle i

$$\vec{f}_i = m_i \vec{a}_i = -\nabla_i U(t) \quad (1)$$

- Potential

$$U(t) = \frac{1}{2} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} U_{i,j}(\|\vec{r}_{i,j}\|) \quad (2)$$



©Amir Niazi, AICES

Molecular Dynamics

Algorithm 1 Overview of the main Molecular Dynamics routine.

```
1: for $i = 1$ to $M$ do
2:     $t \leftarrow t + dt$
3:     compute_forces($\vec{r}, \vec{f}$)
4:     integrate($\vec{r}, \vec{f}, \vec{v}, dt$)
5:     //Do something with the data
6: end for
```

- *compute_forces* has a complexity of $\mathcal{O}(N^2)$

Molecular Dynamics

Algorithm 2 Compute_forces routine.

```
1: for $i = 1$ to $N$ do
2:     $\vec{f_i} \leftarrow 0$
3:     for $j = 1$ to $N$ do
4:         $\vec{r}_{i,j} \leftarrow \vec{r_j} - \vec{r_i}$
5:         $f_{i,j} \leftarrow$ compute_force($\|\vec{r}_{i,j}\|$)
6:         $\vec{f_i} \leftarrow \vec{f_i} + f_{i,j}\ \vec{r}_{i,j}$
7:     end for
8: end for
```

Molecular Dynamics

Algorithm 3 Naive OpenACC compute_forces routine.

```
1: #pragma acc kernels
2: for $i = 1$ to $N$ do
3:     $\vec{f_i} \leftarrow 0$
4:     for $j = 1$ to $N$ do
5:         $\vec{r}_{i,j} \leftarrow \vec{r_j} - \vec{r_i}$
6:         $f_{i,j} \leftarrow \text{compute\_force}(\|\vec{r}_{i,j}\|)$
7:         $\vec{f_i} \leftarrow \vec{f_i} + f_{i,j}\ \vec{r}_{i,j}$
8:     end for
9: end for
```

- Inner loop can not be parallelized
    - Loop-carried dependencies

Molecular Dynamics

Algorithm 4 Improved compute_forces routine.

```
#pragma acc kernels
for $i = 1$ to $N$ do
    $\vec{f_i} \leftarrow 0$
    #pragma acc loop reduction(+:$\vec{f_i}$)
    for $j = 1$ to $N$ do
        $\vec{r}_{i,j} \leftarrow \vec{r_j} - \vec{r_i}$
        $f_{i,j} \leftarrow$ compute_force($\|\vec{r}_{i,j}\|$)
        $\vec{f_i} \leftarrow \vec{f_i} + f_{i,j}\ \vec{r}_{i,j}$
    end for
end for
```

- Good: Inner loop can be parallelized
- Bad: Arrays are reallocated in every iteration

Molecular Dynamics

Algorithm 5 Overview of the main Molecular Dynamics routine with an OpenACC data region.

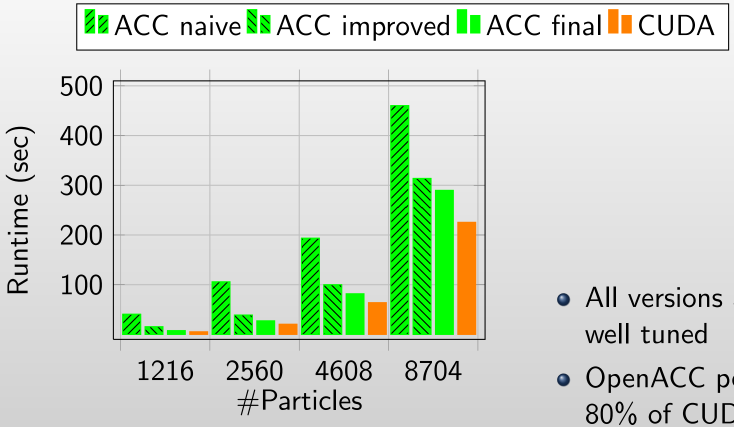
```
1: #pragma acc data create ($\vec{r}$[0:N],$\vec{f}$[0:N])
2: for $i = 1$ to $M$ do
3:     $t \leftarrow t + dt$
4:     compute_forces($\vec{r}, \vec{f}$)
5:     integrate($\vec{r}, \vec{f}, \vec{v}, dt$)
6:     //Do something with the data
7: end for
```

Molecular Dynamics

Algorithm 6 Final compute_forces routine.

```
#pragma acc update device($\vec{r}$[0:N])
#pragma acc kernels present($\vec{r}$[0:N], $\vec{f}$[0:N])
for $i = 1$ to $N$ do
    $\vec{f_i} \leftarrow 0$
    #pragma acc loop reduction(+:$\vec{f_i}$)
    for $j = 1$ to $N$ do
        $\vec{r}_{i,j} \leftarrow \vec{r_j} - \vec{r_i}$
        $f_{i,j} \leftarrow$ compute_force($\|\vec{r}_{i,j}\|$)
        $\vec{f_i} \leftarrow \vec{f_i} + f_{i,j}\ \vec{r}_{i,j}$
    end for
end for
#pragma acc update host($\vec{f}$[0:N])
```

Performance

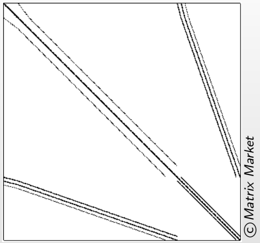


- All versions are equally well tuned
- OpenACC performs at 80% of CUDA

Figure: Runtime of a Molecular Dynamics (MD) Simulation for different problem sizes over 10.000 iterations. All calculations are run in double precision. OpenMP: 16 core SMP node. OpenACC/Cuda: Nvidia Quadro 6000 GPU.

Conjugate Gradient Method

Conjugate Gradient Method



- Iterative solver
- Solve a large sparse linear system

$$Ax = b \tag{3}$$

- Frequently arise from *partial differential equations* in physics
- Runtime dominated by *Sparse Matrix-Vector Multiplication* SPMV

Performance

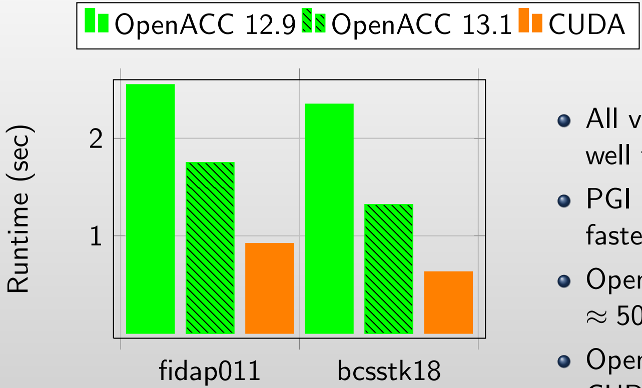


Figure: Runtime of a Conjugate Gradient (CG) Method for two sparse matrices. All calculations are run in double precision. OpenMP: 16 core SMP node. OpenACC/Cuda: Nvidia Quadro 6000 GPU.

- All versions are equally well tuned
- PGI 13.1 50%/80% faster than PGI 12.9
- OpenACC performs at $\approx$ 50% of CUDA
- OpenMP outperforms CUDA

Productivity

Productivity - Contra

- Function calls require inlining
- PGI compiler does not support C++
- Limited debugging support for PGI compiler
    - Revert to debugging the logic of your application

Productivity - Pro

	OpenMP	OpenACC	CUDA
MD	23	16	92
CG	8	16	156

Table: Number of added and modified lines of source code for each case study and paradigm with respect to the serial version.

- Few added/modified lines of source code
- Data transfers are straight forward
- Reductions require almost no additional effort
- No need to worry about "boundary conditions"
- Compiler is able to tune for a specific coprocessor

Conclusion

- High productivity (if you don't run into compiler bugs)
- Decent performance
- Limited debugging support for PGI compiler
- Makes coprocessor programming more straight forward
  - C code $\rightarrow$ OpenACC code $\rightarrow$ CUDA code

Conclusion

- High productivity (if you don't run into compiler bugs)
- Decent performance
- Limited debugging support for PGI compiler
- Makes coprocessor programming more straight forward
    - C code $\rightarrow$ OpenACC code $\rightarrow$ CUDA code

Thank you for your attention.