GPU implementation of Cell Dynamics simulation for block copolymer systems

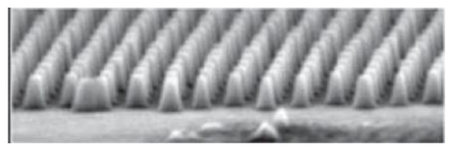
Ludwig Schreier, Marco Pinna, Andrei V. Zvelindovsky
lschreier@uclan.ac.uk

University of Central Lancashire
School of Computing, Engineering and Physical Sciences
Computational Physics Group
PR1 2HE Preston
United Kingdom

Using NVIDIA CUDA programming language, we implemented Cell Dynamics simulation (CDS) for the modelling of block copolymers on the GPU. The code was developed, tested and benchmarked on a NVIDIA Quadro FX4600 and Tesla C1060 card. We compare results with a two-dimensional to one-dimensional domain decomposed version in C language and conventional Fortran 90 version using nested-array-loops. Performance of the code as a whole as well as of various internal parts has been analysed in detail.

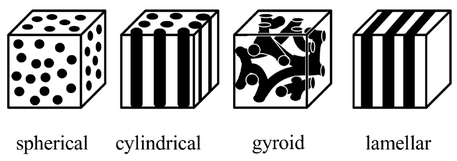
Two CUDA based version of CDS, one with a two Kernel and one with a four Kernel approach were developed for testing reasons. For lamellae systems in two dimensional simulation box of 1024*1024 grid points, enormous speedups can be achieved using CUDA compared to Fortran 90 code. The created C version shows tremendous speedup optimisation which can be related to SIMD features of the CPU. The boundary condition implementation was identified as bottleneck for further speedup optimisation.

$$\frac{\partial \psi}{\partial t} = M\nabla^2 \frac{\delta \mathcal{F}(\psi)}{\delta \psi} + \eta\xi$$

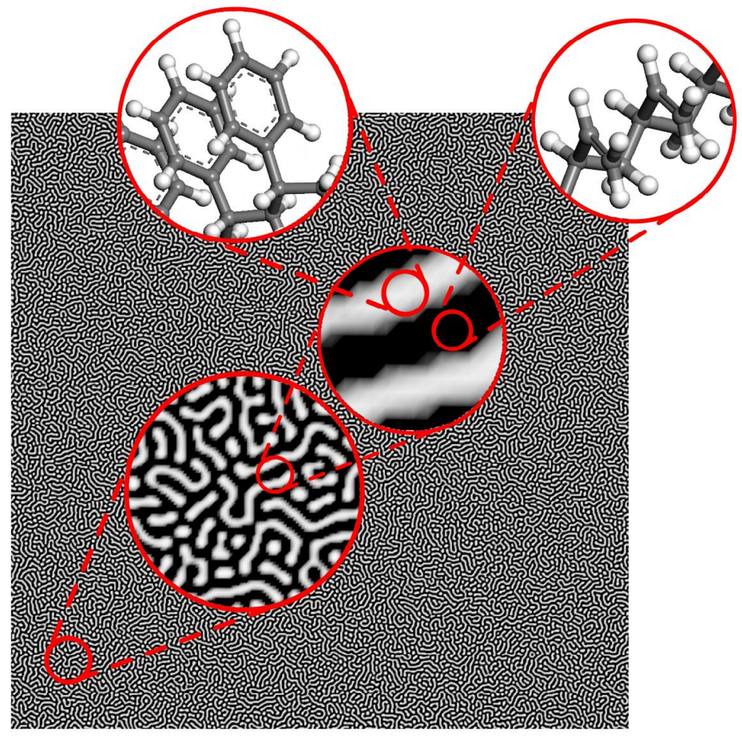


(c) Hitachi Global Storage Technologies

~ 1-1000 nm



Correlation between theory, simulation and experiment



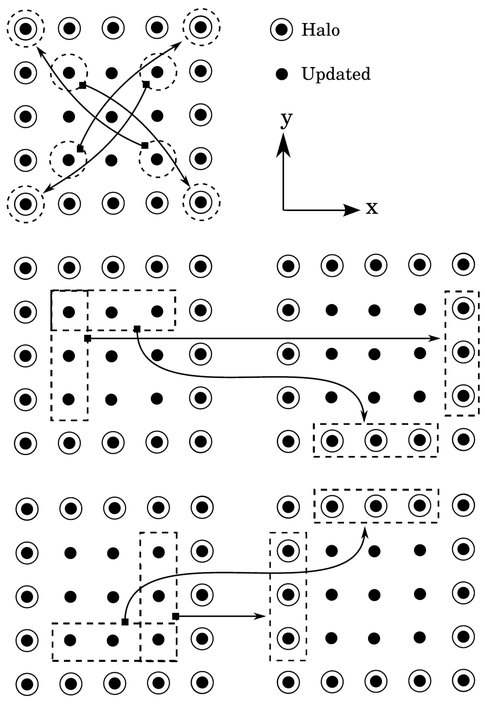
Zoom-image of simulation regime (mesoscale) for diblock copolymer

```
Define simulation parameters
Allocate host/device memory arrays
Initialize ψ with random values for host/device (both same)
Copy host/device memory
    Start time loop (CUDA)
    {
        Call Kernel 1 Laplacian and eq. (2.16), ψ^new ← ψ^t
        Call Kernel 2 boundary conditions, ψ^new' ← ψ^new
        Call Kernel 3 Laplacian and eq. (2.13), ψ^new'' ← ψ^new'
        Call Kernel 4 boundary conditions, ψ^(t+1) ← ψ^new''
        Output data every T time steps
    }
    Stop time loop (CUDA)
    Start time loop (Gold)
    {
    Run Gold version (Cell Dynamics in C language)
    Output data every T time steps
    }
    Stop time loop (Gold)
Free all host/device memory
Output system parameters and timer measurements
(Postprocess visualize and analyze results)
```

Cell Dynamics CUDA implementation using 4 Kernel approach

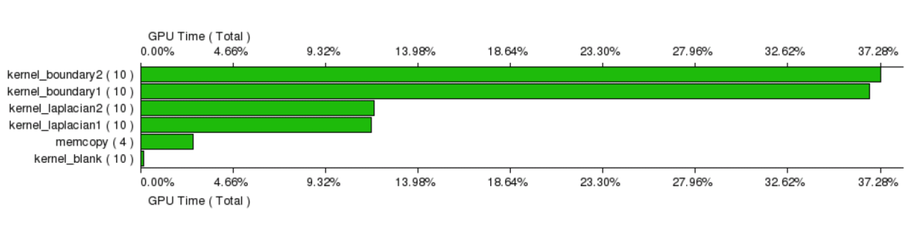
```
Define simulation parameters
Allocate host/device arrays
Initialize ψ with random values for host/device (both same)
Copy host/device memory arrays
    Start time loop (CUDA)
    {
        Call Kernel 1 Laplacian and eq. (2.16), ψ^new ← ψ^t
        Copy device/host memory
        Calculate boundary conditions on host, ψ^new' ← ψ^new
        Copy host/device memory
        Call Kernel 2 Laplacian and eq. (2.13), ψ^new'' ← ψ^new'
        Copy device/host memory
        Calculate boundary conditions on host, ψ^(t+1) ← ψ^new''
        Output data every T time steps
        Copy host/device memory
    }
    Stop time loop (CUDA)
    Start time loop (Gold)
    {
    Run Gold version (Cell Dynamics in C language)
    Output data every T time steps
    }
    Stop time loop (Gold)
Free all host/device memory
Output system parameters and timer measurements
(Postprocess visualize and analyze results)
```

Cell Dynamics CUDA implementation using 2 Kernel approach

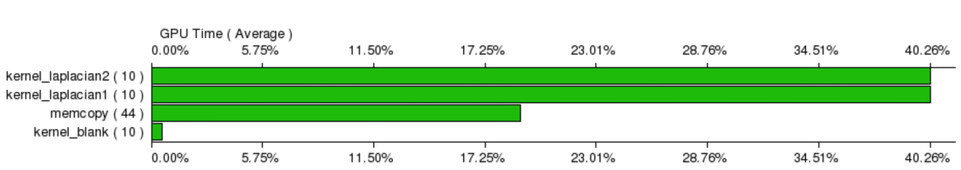


Boundary condition exchange via halos (ghostpoints)

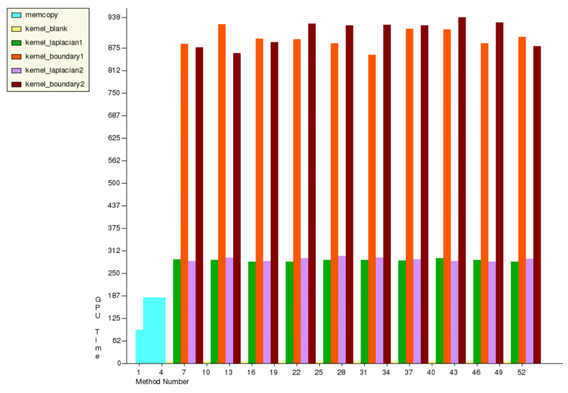
Development platforms:
Quadro FX4600 and Tesla C1060

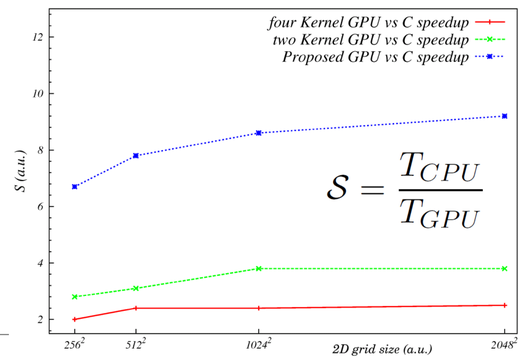


CUDA Visual Profiler benchmarking results

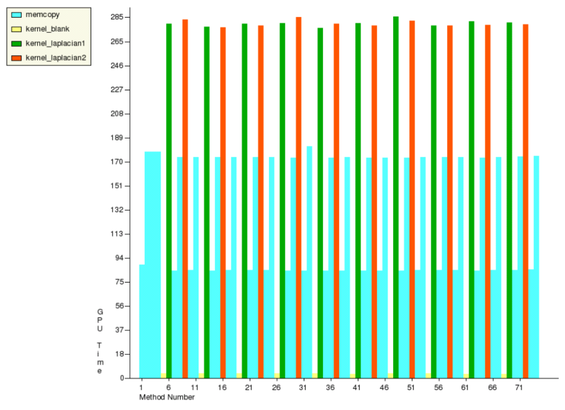


CUDA Visual Profiler benchmarking results

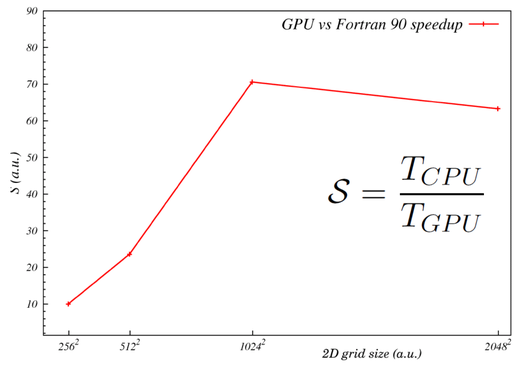




Speedup of CUDA vs C



CUDA Visual Profiler benchmarking results



Speedup of CUDA vs Fortran

Acknowledgments
The work is supported by Accelrys Ltd. Via EPSRC CASE research studentship.