GPU Implementation of a Stencil Code with More Than 90% of the Peak Theoretical Performance

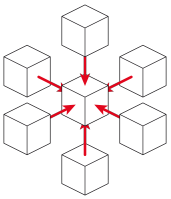
I Pershin, V Levchenko, A Perepelkina
Keldysh Institute of Applied Mathematics RAS, Moscow

MOSCOW, RUSSIA, SEPTEMBER 23–24, 2019

Stencil tasks & GPU

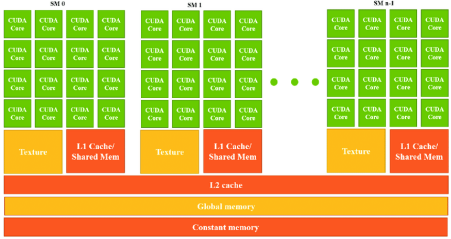
Stencil task

- A large range of applications
- High parallelism
- Homogeneous calculations
- Low intencity



Nvidia GPU

- High peak performance $\Pi_{peak} > 10$ TFLOPS
- About 1000 CUDA Threads per Block/SM
- SIMT execution model
- High Bandwidth Memory
- Large register file in each SM
- About 50 independent SMs
- Considerable L2 cache latency



(2r+2)-point cross stencil

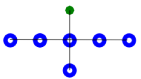
1-D Wave Equation

$$u_{tt} = c^2 u_{xx}, \text{ plus B.C. and I.C.}$$

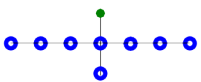
The cross-stencil

$$u_k^{n+1} = -u_k^{n-1} + \underbrace{(\alpha_0 u_k^n + \alpha_{-1} u_{k-1}^n + \alpha_1 u_{k+1}^n + \ldots + \alpha_{-r} u_{k-r}^n + \alpha_r u_{k+r}^n)}_{(2r+1) \text{ spatial terms}}$$

2^{nd} order



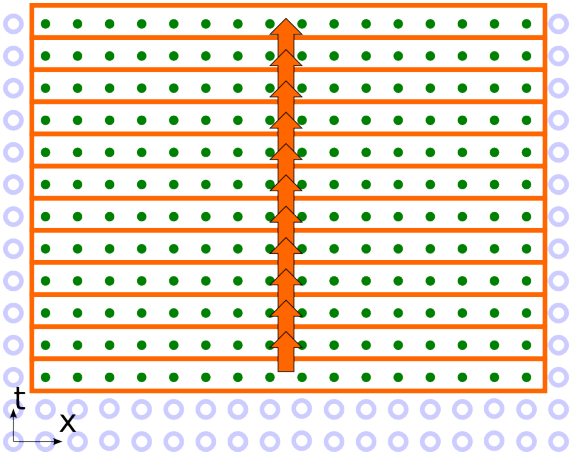
4^{th} order



6^{th} order

Stepwise Algorithm

- Implemented in most applied codes
- u^n and u^{n-1} are stored in the global memory: 10^9 cells
- 1 CUDA thread manages 1 cell
- The number of threads per block doesn't matter
- The number of blocks is not limited
- The kernel updates the whole domain once; it is executed in a loop
- Automatic synchronization after each Δt



Roofline Model for Stepwise Algorithm

Any algorithm performance limit

$$\Pi_{alg} \leq \min(\Pi_{peak}, \Theta \cdot I_{alg}) \equiv \min(\Pi_{peak}, \Theta \cdot \frac{O_{alg}}{D_{alg}})$$

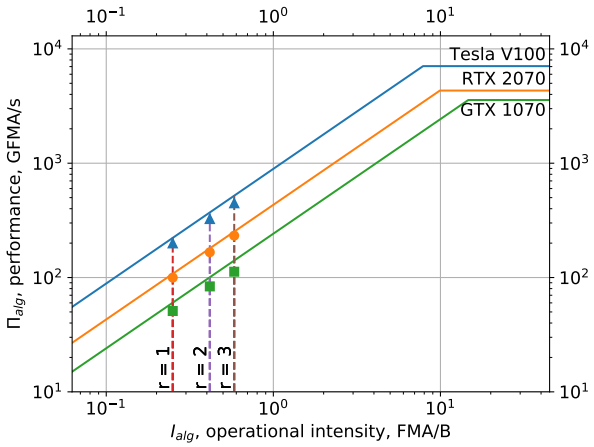
Cross-stencil

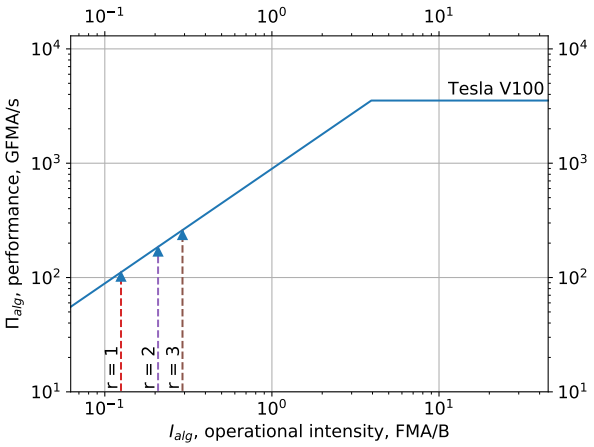
$$u_k^{n+1} = \underbrace{\alpha_{\pm r}(\overbrace{u_{k+r}^n + u_{k-r}^n}^{\text{FMA}}) + \ldots + \underbrace{\alpha_{\pm 1}(\overbrace{u_{k+1}^n + u_{k-1}^n}^{\text{FMA}}) + \overbrace{\alpha_0 u_k^n - u_k^{n-1}}^{\text{FMA}}}_{\text{FMA}}}_{\text{FMA}}$$

- Operation count $O_{alg} = (2r+1)\ \frac{\text{FMA}}{\text{cell}\cdot\text{step}}$
- Data $D_{alg} = 3 \cdot s \frac{\text{B}}{\text{cell}\cdot\text{step}}$, since the rest is take from the cache; $s = 4$ or 8

Arithmetic Intensity: $I_{alg} = \frac{O_{alg}}{D_{alg}} = \frac{2r+1}{3s}\ \frac{\text{FMA}}{\text{B}}$

Roofline Model for Stepwise Algorithm

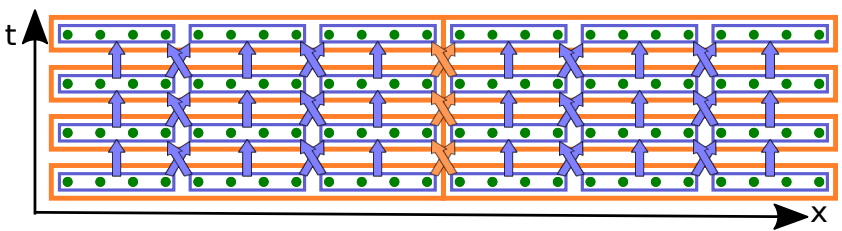




Recursive Domain Decomposition

- u^n and u^{n-1} *are stored in registers*: up to 10^6 cells
- 1 thread per localized group of cells
- 256 or 512 threads per block
- CUDA-block number $\sim$ SM number
- Δt loop is inside the kernel

- Inter-thread data exchange: shared memory
- Inter-block data exchange: L2 cache
- Thread synchronization: syncthreads()
- Block synchronization: Cooperative Groups / Semaphores



SM Synchronization Methods

Cooperative Groups

- The official API for synchronization
- Barrier synchronization for all SMs at once
- Available for Compute Capability ≥ 6.1
- Easy to use
- May be used for any problems and any stencils
- However, it has not provided significant performance gain over the stepwise algorithm

Semaphore Synchronization

- Classical tool for synchronization
- Separate SMs may be synchronized with each other
- Available for any Compute Capability
- Manual implementation required
- The code has to be adapted for the problem and the stencil
- Fast

SM synchronization: Implementation

Cooperative Groups

```
#include <cooperative_groups.h>
using namespace cooperative_groups;

grid_group grid = this_grid();
grid.sync();
```

Semaphore Synchronization

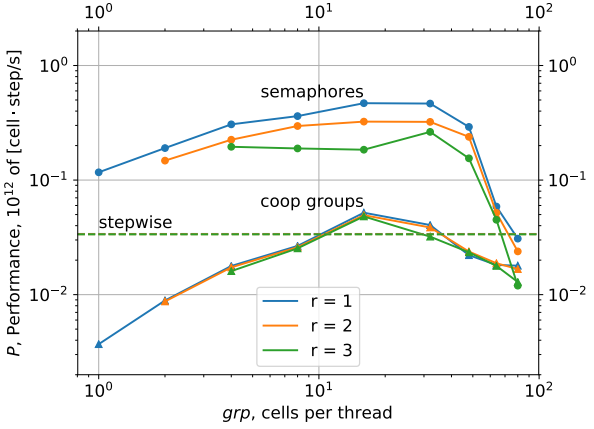
```
volatile float *data;
volatile int *semaph;

while(semaph[blockIdx +- 1] != READABLE);
//read data written by neighboring blocks..
semaph[blockIdx +- 1] = WRITABLE;


while(semaph[blockIdx] != WRITABLE);
//write data for neighboring blocks to read
__threadfence();
semaph[blockIdx] = READABLE;
```

Performance Test for Recursive Domain Decomposition

- P grows with cell number per thread (grp) since the overhead per cell decreases
- P falls abruptly for large grp due to lack of space in registers
- Optimal grp is $\sim 16 \div 48$
- With synchronization through cooperative groups, there is no advantage over the stepwise algorithm
- With manual Semaphore Synchronization the performance is higher by an order of magnitude



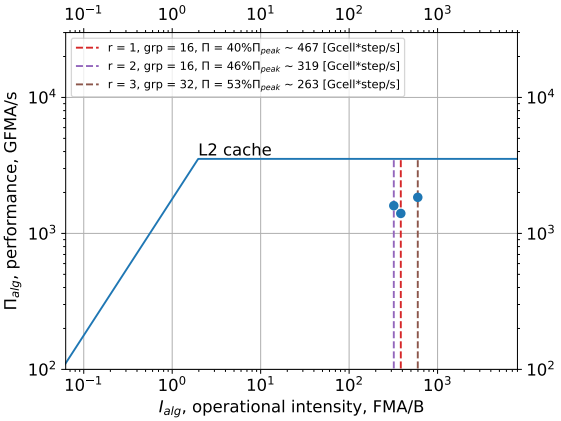
Roofline Model for Recursive Domain Decomposition

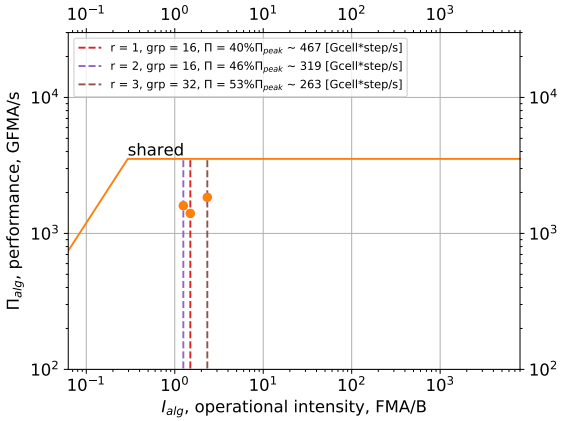
$$\Pi_{alg} \leq \min(\Pi_{peak}, \Theta_{L2} \cdot I_{alg,L2}, \Theta_{sh} \cdot I_{alg,sh})$$

- Operation count $O_{alg} = (2r+1)\ \frac{\mathrm{FMA}}{\mathrm{cell \cdot step}}$
- Data sent inter-block $D_{alg,L2} = \frac{4r}{\mathrm{grp \cdot threads}} \cdot s \frac{\mathrm{B}}{\mathrm{cell \cdot step}} \sim \frac{r}{512} \frac{\mathrm{B}}{\mathrm{cell \cdot step}}$
- Data sent inter-thread $D_{alg,sh} = \frac{4r}{\mathrm{grp}} \cdot s \frac{\mathrm{B}}{\mathrm{cell \cdot step}} \sim \frac{r}{2} \frac{\mathrm{B}}{\mathrm{cell \cdot step}}$

$$I_{alg} = \frac{O_{alg}}{D_{alg}} \sim \begin{cases} 512 \cdot \frac{2r+1}{r}\ \frac{\mathrm{FMA}}{\mathrm{B}}, \text{ inter-block via L2 cache} \\ 2 \cdot \frac{2r+1}{r}\ \frac{\mathrm{FMA}}{\mathrm{B}}, \text{ inter-thread via shared memory} \end{cases}$$

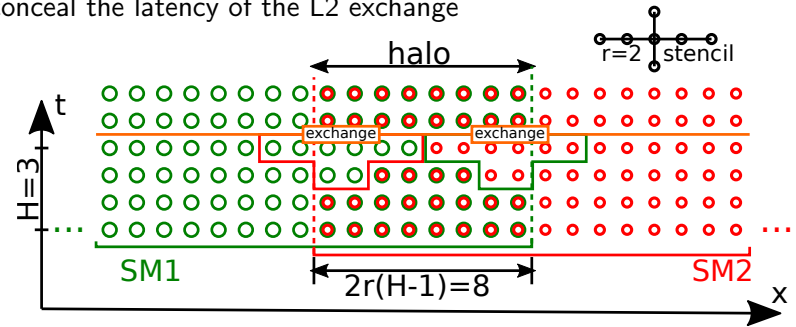
Roofline Model for Recursive Domain Decomposition





Recursive Domain Decomposition with Halo

- Instead of sending D data every step $\rightarrow$ send $\sim H \cdot D$ data every H steps
- Redundant compute in the overlapping region (halo): $2r(H-1)$ cells
- The halo is implemented for both levels of the Recursive Domain Decomposition; however, the performance gain is seen only for L2 level
- Allows to conceal the latency of the L2 exchange



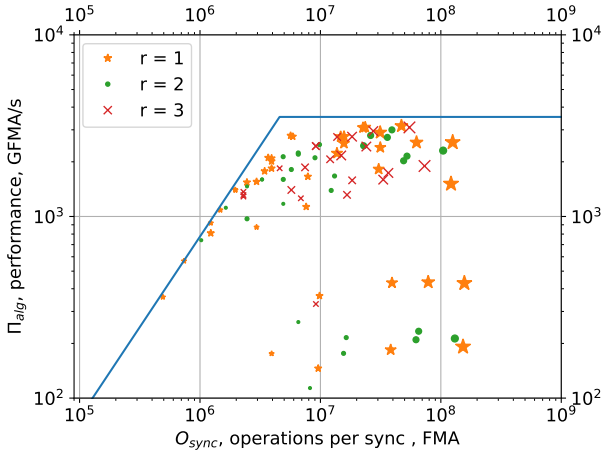
Roofline Model for Recursive Domain Decomposition with Halo

$$\Pi_{alg} \leq \min(\Pi_{peak}, \ldots, O_{\text{sync}}/\Lambda_{\text{sync}})$$

- Inter-block synchronization time $\Lambda_{\text{sync}} = 1.3$ mks
- Operation count between synchronizations $O_{\text{sync}} = KHO_{alg}$
- Operation count per cell $O_{alg} = (2r+1)\ \frac{\text{FMA}}{\text{cell}\cdot\text{step}}$

Roofline Model for Recursive Domain Decomposition with Halo

- $r = 1$: $\mathtt{grp} = 48$, $H = 16$,
  $\Pi = 90\%\ \Pi_{peak} \sim 1.05 \cdot 10^{12} \frac{\text{cell}\cdot\text{step}}{\text{s}}$;
- $r = 2$: $\mathtt{grp} = 48$, $H = 8$,
  $\Pi = 86\%\ \Pi_{peak} \sim 0.60 \cdot 10^{12} \frac{\text{cell}\cdot\text{step}}{\text{s}}$;
- $r = 3$: $\mathtt{grp} = 48$, $H = 8$,
  $\Pi = 88\%\ \Pi_{peak} \sim 0.44 \cdot 10^{12} \frac{\text{cell}\cdot\text{step}}{\text{s}}$.



Conclusion

- Stencil codes can be latency-bound instead of memory-bound, if the data is localized in registers
- Stencil codes can be compute-bound instead of latency-bound, if the data is synchronized once per several time steps
- We have developed and implemented Recursive Domain Decomposition with Halo and reached 90% of the peak performance (more than 1 trillion cell updates per second). The key features are:
  - data localization in registers
  - pairwise semaphore synchronization
  - synchronization once in several time steps (halo)

Contact:
pershin2010@gmail.com

This work is supported by grant #18-71-10004 of the Russian Science Foundation.