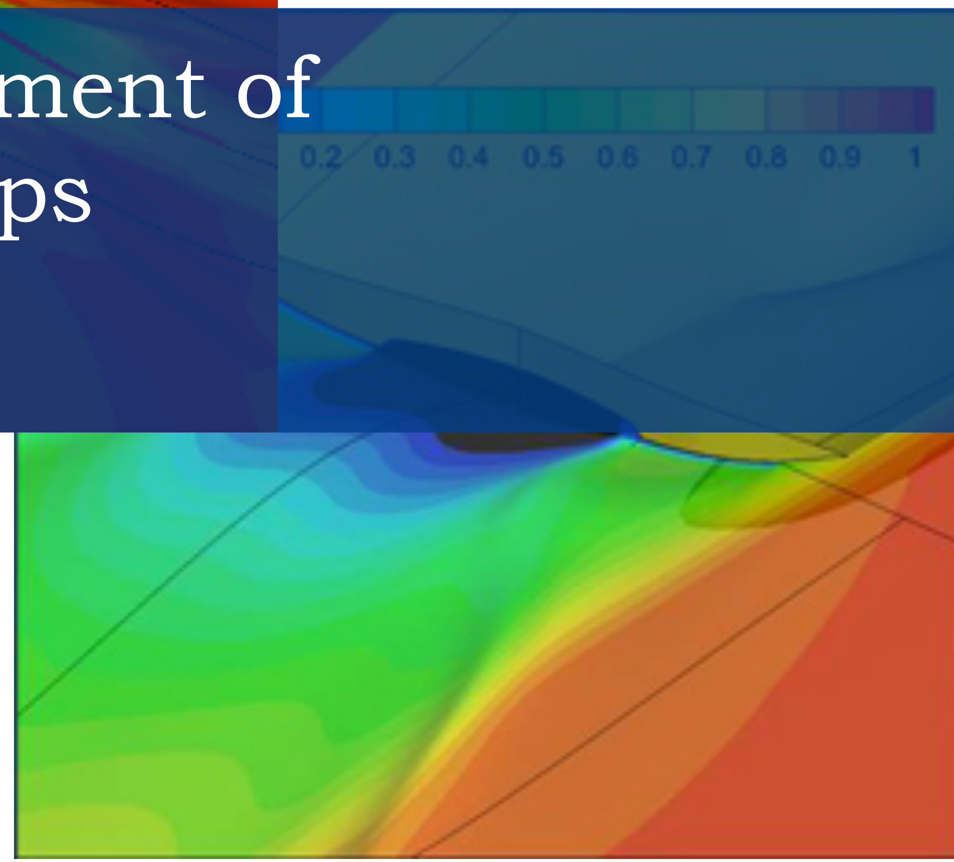
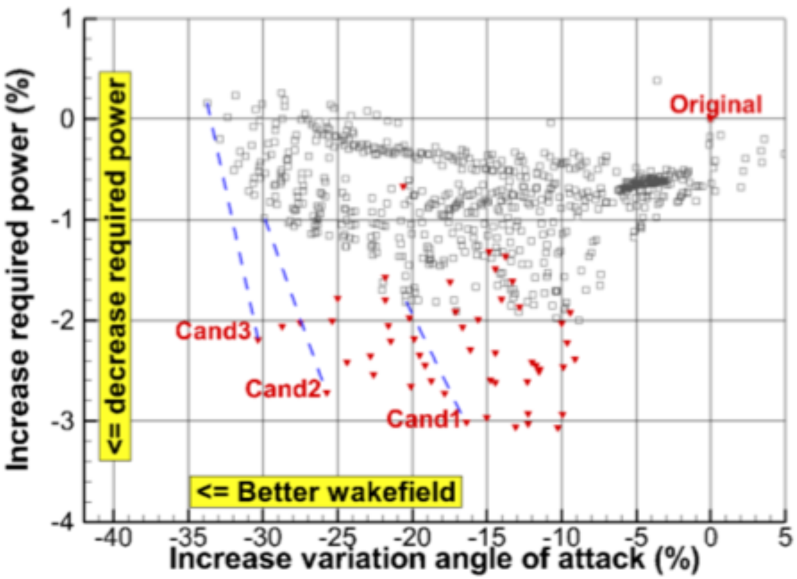
Efficiency improvement of optimization of ships

Literature Survey

March 17, 2017





- CFD
- Parnassos
- Automatic optimization
- Time-consuming
- 80% Ax=b

Improve performance of linear solver by using GPUs?

Outline

- Parnassos
- Iterative methods
- Graphics Processing Unit (GPU)
- Iterative methods on the GPU
- Conclusions literature survey

PARNASSOS

A RANS solver for structured grids

Accuracy

Robustness

Efficiency

Flexibility

PARNASSOS

A RANS solver for structured grids

Accuracy

High-order finite difference schemes

Robustness

Efficiency

Flexibility

PARNASSOS

A RANS solver for structured grids

Accuracy

High-order finite difference schemes

Robustness

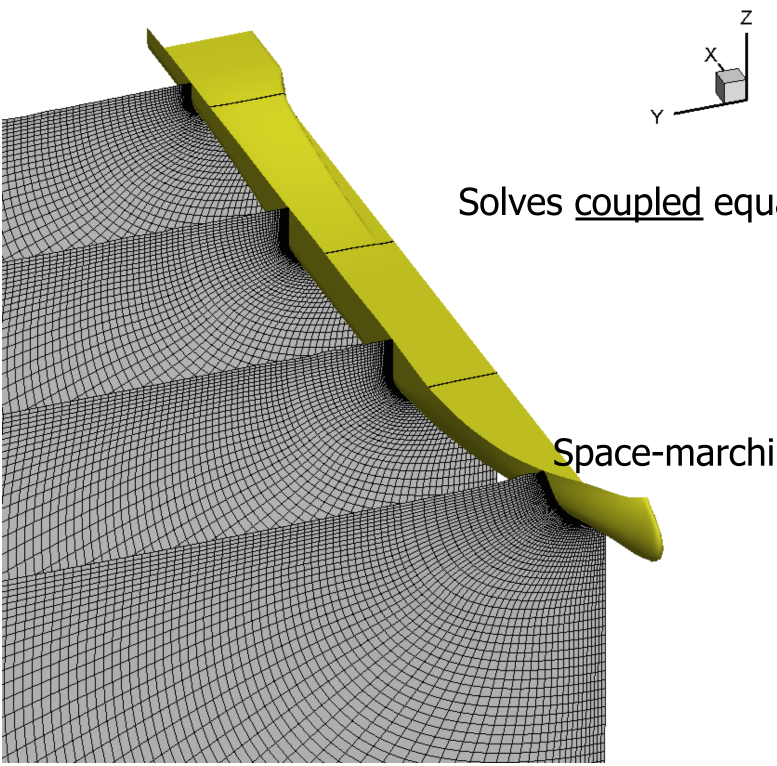
Solves coupled equations (+ uncoupled turbulence model)

Efficiency

Flexibility

PARNASSOS

A RANS solver for structured grids



Accuracy

High-order finite difference schemes

Robustness

Solves <u>coupled</u> equations (+ uncoupled turbulence model)

Efficiency

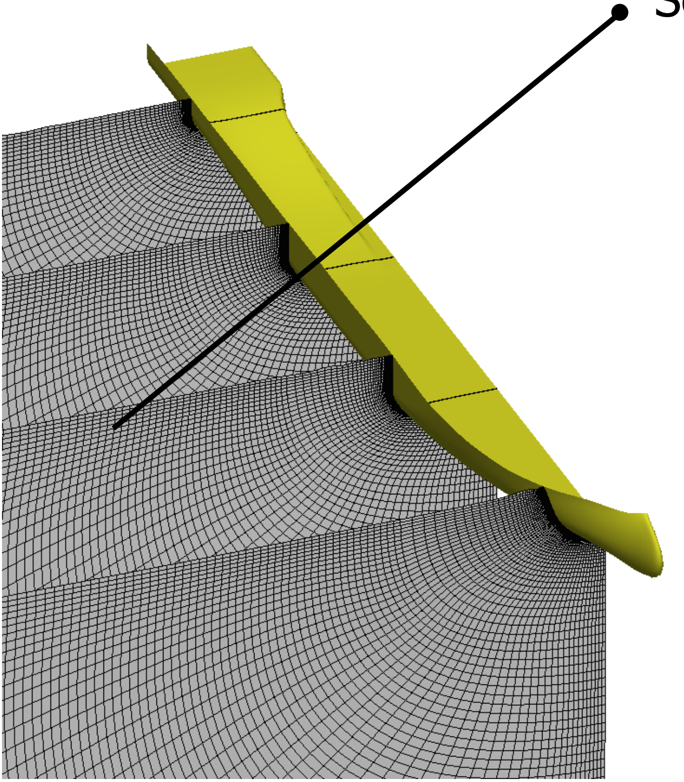
Space-marching method: inner-outer iterations

Flexibility

PARNASSOS

A RANS solver for structured grids

- Solve **Ax=b**, with A of size $(4 \times g \times NY \times NZ)^2$

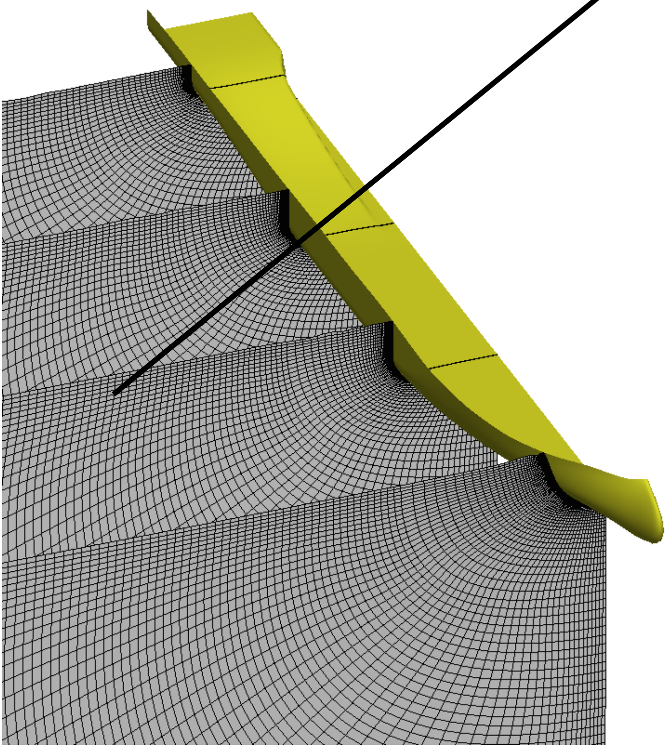


$$A = \begin{bmatrix} d & e & f & 0 & 0 & 0 & \cdots \\ c & d & e & f & 0 & 0 & \cdots \\ b & c & d & e & f & 0 & \cdots \\ 0 & b & c & d & e & f & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & b & c & d \end{bmatrix}$$

PARNASSOS

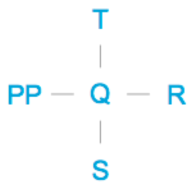
A RANS solver for structured grids

$$A = \begin{bmatrix} d & e & f & 0 & 0 & 0 & \cdots \\ c & d & e & f & 0 & 0 & \cdots \\ b & c & d & e & f & 0 & \cdots \\ 0 & b & c & d & e & f & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & b & c & d \end{bmatrix}$$



- Solve **Ax=b**, with A of size $(4 \times g \times NY \times NZ)^2$

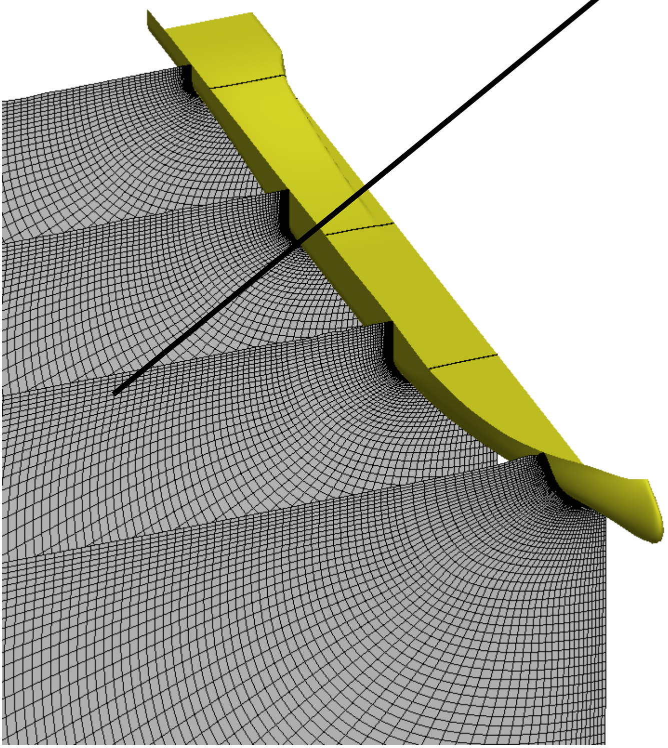
$$d = \begin{bmatrix}
Q & R & 0 & \cdots & T & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
PP & Q & R & 0 & \cdots & T & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & PP & Q & R & 0 & \cdots & T & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & PP & Q & 0 & \cdots & \cdots & 0 & T & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
S & 0 & \cdots & \cdots & 0 & Q & R & 0 & \cdots & 0 & T & 0 & \cdots & \cdots & \cdots & 0 \\
0 & S & 0 & \cdots & \cdots & PP & Q & R & 0 & \cdots & 0 & T & 0 & \cdots & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & 0 & S & 0 & \cdots & 0 & PP & Q & R & \cdots & \cdots & 0 & T & 0 \\
0 & \cdots & \cdots & \cdots & 0 & S & 0 & \cdots & 0 & PP & Q & 0 & \cdots & \cdots & 0 & T \\
0 & \cdots & \cdots & \cdots & \cdots & 0 & S & 0 & \cdots & \cdots & 0 & Q & R & \cdots & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & S & 0 & \cdots & 0 & PP & Q & R & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & S & 0 & \cdots & 0 & PP & Q & R \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & S & 0 & \cdots & 0 & PP & Q
\end{bmatrix}$$



PARNASSOS

A RANS solver for structured grids

$$A = \begin{bmatrix} d & e & f & 0 & 0 & 0 & \cdots \\ c & d & e & f & 0 & 0 & \cdots \\ b & c & d & e & f & 0 & \cdots \\ 0 & b & c & d & e & f & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & b & c & d \end{bmatrix}$$



- Solve **Ax=b**, with A of size (4 x g x NY x NZ)^2

$$b = \begin{bmatrix} \mathrm{UPSTRb} & 0 & \cdots \\ 0 & \mathrm{UPSTRb} & \cdots \\ \vdots & \ddots & \ddots \\ 0 & \cdots & \mathrm{UPSTRb} \end{bmatrix}$$

$$c = \begin{bmatrix} \mathrm{UPSTRa} & 0 & \cdots \\ 0 & \mathrm{UPSTRa} & \cdots \\ \vdots & \ddots & \ddots \\ 0 & \cdots & \mathrm{UPSTRa} \end{bmatrix}$$

$$e = \begin{bmatrix} \mathrm{DOWNSTRa} & 0 & \cdots \\ 0 & \mathrm{DOWNSTRa} & \cdots \\ \vdots & \ddots & \ddots \\ 0 & \cdots & \mathrm{DOWNSTRa} \end{bmatrix}$$

$$f = \begin{bmatrix} \mathrm{DOWNSTRb} & 0 & \cdots \\ 0 & \mathrm{DOWNSTRb} & \cdots \\ \vdots & \ddots & \ddots \\ 0 & \cdots & \mathrm{DOWNSTRb} \end{bmatrix}$$

Iterative Methods

Parnassos: Solve **Ax=b**, with A of size (4 x g x NY x NZ)^2

$$A = \begin{bmatrix} d & e & f & 0 & 0 & 0 & \cdots \\ c & d & e & f & 0 & 0 & \cdots \\ b & c & d & e & f & 0 & \cdots \\ 0 & b & c & d & e & f & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & 0 & b & c & d \end{bmatrix}$$

- Large
- Sparse
- Non-symmetric
- Diagonally structured

Preconditioned Krylov solvers

Iterative Methods

Preconditioned Krylov solvers for sparse linear systems

Non-symmetric systems Ax=b:

- Optimal methods: GMRES
- Short recurrences: BiCGStab
- Hybrid: IDR(s)

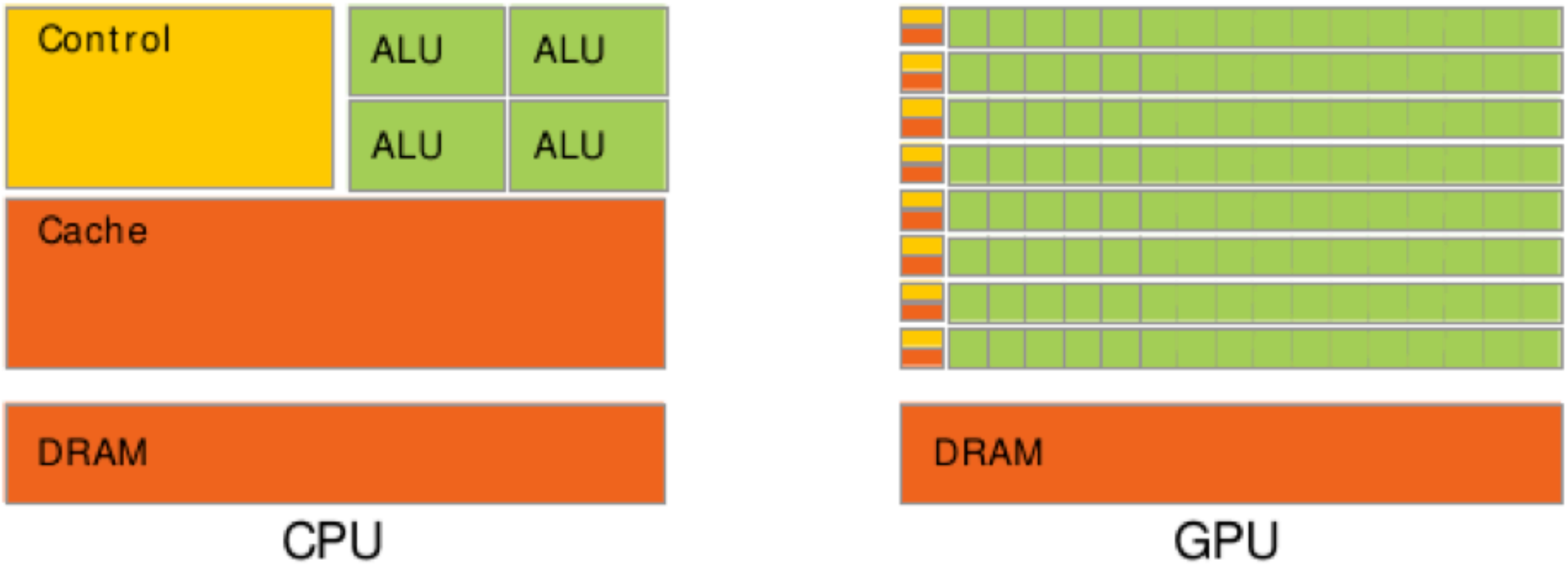
- Preconditioning: ILU

Parnassos: ILU preconditioned GMRES

80% of CPU time

Graphics Processing Unit (GPU)

Scientific computing with GPUs



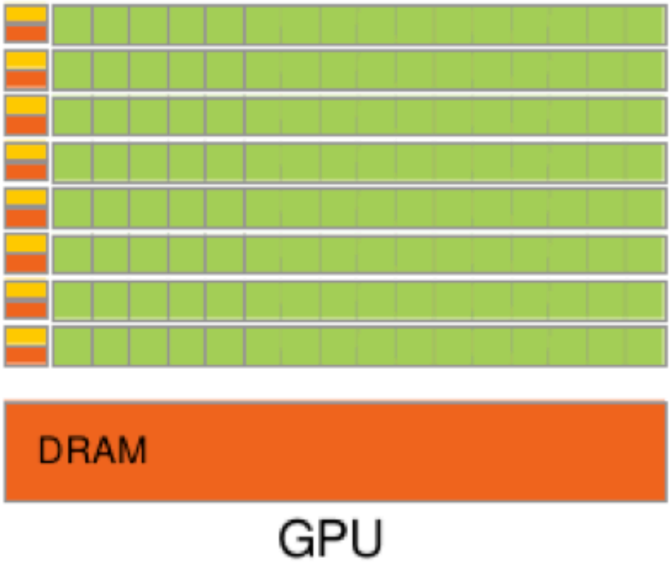
- High floating-point performance
- Cheap & available
- Scalable

Graphics Processing Unit (GPU)

Scientific computing with GPUs

SIMD ≈ SIMT (threads)

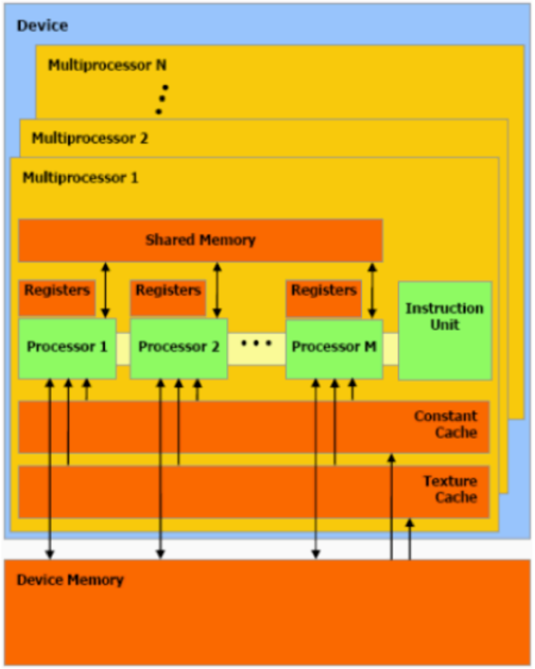
```
_global_ void VecAdd(float* A, float* B, float*C)
{
int i = threadIdx.x;
C[i] = A[i] + B[i];
}


int main()
{
...
VecAdd <<< 1, N >>> (A, B, C);
...
}
```



Fine-grained parallelism

Graphics Processing Unit (GPU)

Scientific computing with GPUs



Complex memory hierarchy

$$t_{comm} = \alpha + \beta n$$

High arithmetic intensity (flop/byte)

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

Fine-grained parallelism & High arithmetic intensity

Krylov solvers:

- Vector updates (e.g. x=x+ay)
- Dot products (e.g. $x^T y$)
- Matrix-vector products

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

Fine-grained parallelism	&	High arithmetic intensity

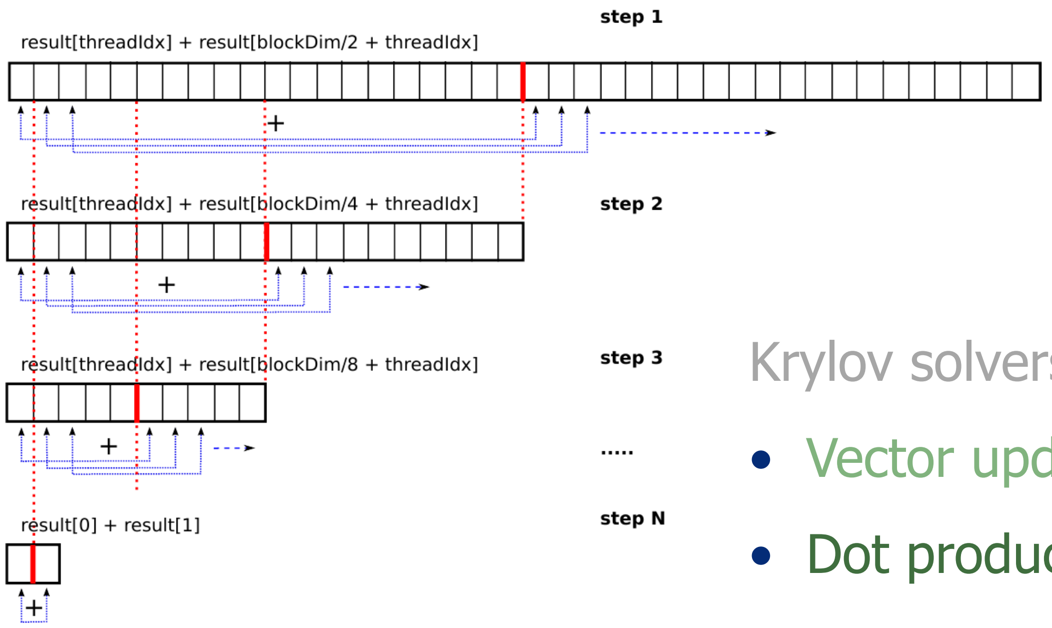
Krylov solvers:

- Vector updates (e.g. x=x+ay)
- Dot products (e.g. $x^T y$)
- Matrix-vector products

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

Sum reduction algorithm



Krylov solvers:

- Vector updates (e.g. x=x+ay)
- Dot products (e.g. $x^T y$)
- Matrix-vector products

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

Sparse matrix-vector product

(SpMV)

- Dependent on storage format: DIA

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 5 & 0 & 0 \\ 0 & 6 & 7 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 10 \\ 11 & 0 & 0 & 0 & 12 & 0 & 0 \\ 0 & 13 & 0 & 0 & 14 & 15 & 0 \\ 0 & 0 & 16 & 0 & 0 & 17 & 18 \end{pmatrix}$$

DIAG =

*	*	1	2
*	3	4	5
*	6	7	8
*	*	9	10
11	*	12	*
13	14	15	*
16	17	18	*

IOFF =

-4	-1	0	3

- Highly optimized

Krylov solvers:

- Vector updates (e.g. x=x+ay)
- Dot products (e.g. $x^T y$)
- Matrix-vector products

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

Fine-grained parallelism & High arithmetic intensity

Krylov solvers:

- Vector updates (e.g. $x=x+ay$)
- Dot products (e.g. x^Ty)
- Matrix-vector products

GMRES

BiCGStab

IDR(s)

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

Fine-grained parallelism	&	High arithmetic intensity

Preconditioning ≈ sequential operation

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

~~Fine-grained parallelism & High arithmetic intensity~~

Preconditioning ≈ sequential operation

convergence parallelism

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

~~Fine-grained parallelism~~	~~&~~	~~High arithmetic intensity~~

Preconditioning ≈ sequential operation

convergence	parallelism
GLOBAL	LOCAL

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

~~Fine-grained parallelism & High arithmetic intensity~~

Preconditioning ≈ sequential operation

convergence	⟺	parallelism
GLOBAL		LOCAL

e.g. ILU (block-Jacobi) | e.g. diagonal (block-Jacobi)

Challenge!

Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

- What to implement?
- Efficient GPU implementation ≈ cumbersome!

Why not make use of libraries!?

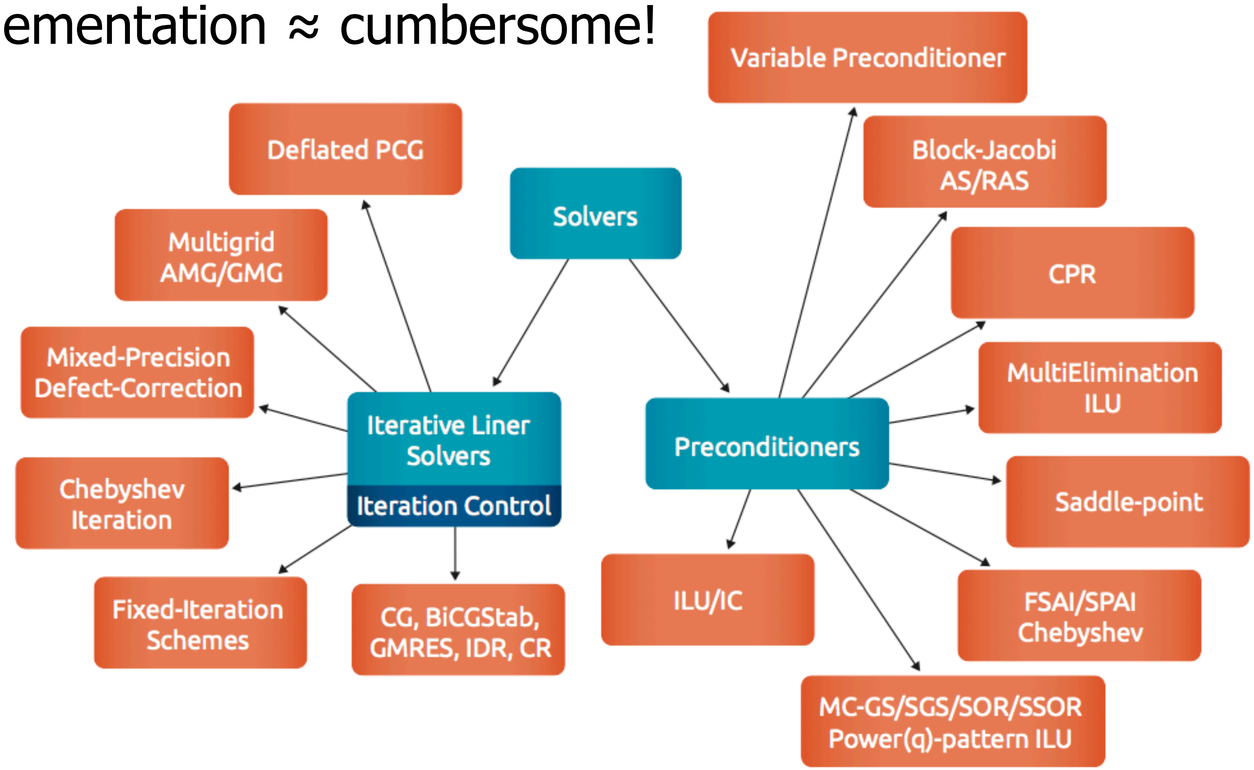
Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

- What to implement?
- Efficient GPU implementation ≈ cumbersome!

Use of libraries!

PARALUTION



Iterative methods on the GPU

Parallel implementation of preconditioned Krylov solvers

Further improvements?

- Mixed-precision techniques
- Deflation
- Multigrid
- Multi-GPU

Conclusions

Research question, experimental setup & planning

How to / Is it possible to achieve reasonable speedup of Parnassos' linear solver by making use of GPU computing?

Conclusions

Research question, experimental setup & planning

How to / Is it possible to achieve reasonable speedup of Parnassos' linear solver by making use of GPU computing?

Which preconditioned Krylov solver to use?

Strategies for fast GPU implementation?

Can the CUDA program be further optimized? How?

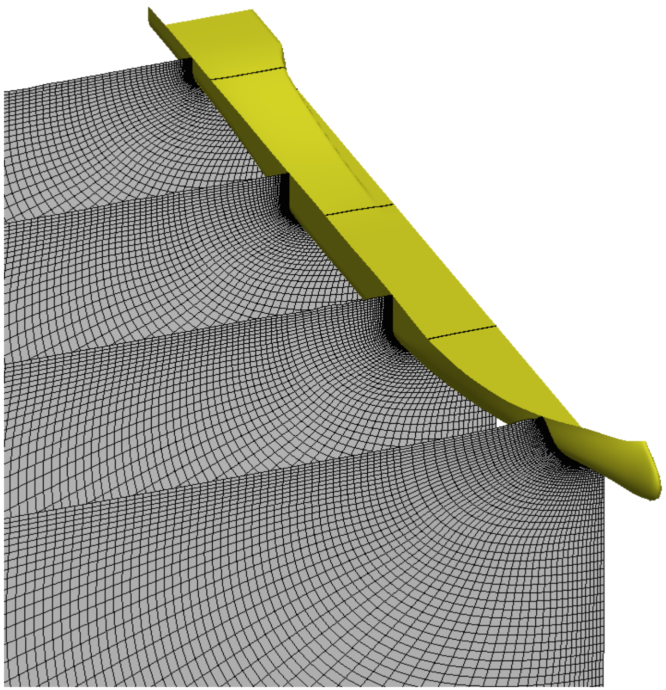
Overall speedup for Parnassos?

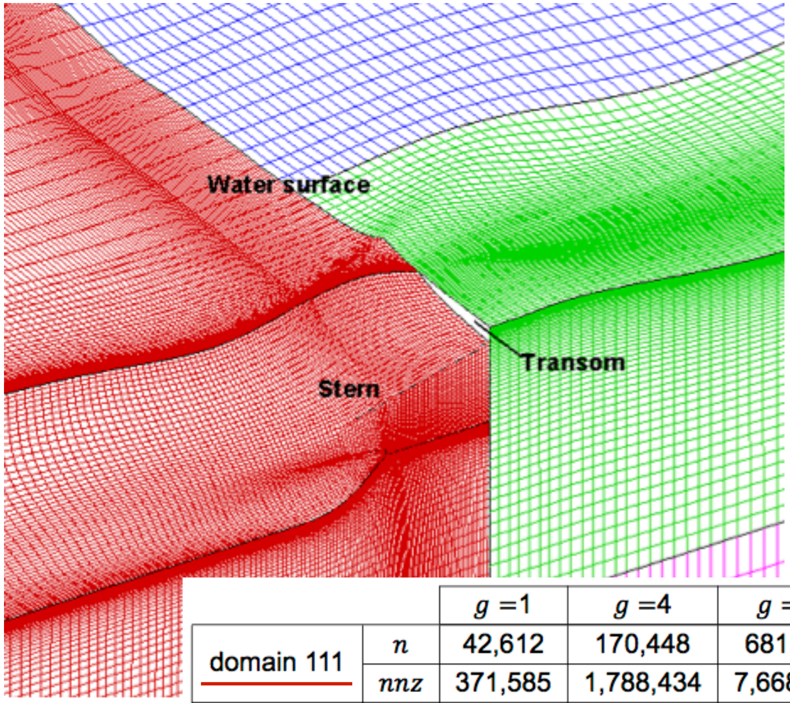
...

Conclusions

Research question, experimental setup & planning

- MARIN cluster - Marclus3
- Test problems – Parnassos





		$g=1$	$g=4$	$g=16$	$g=NX$
domain 111	n	42,612	170,448	681,792	13,678,452
	nnz	371,585	1,788,434	7,668,890	157,130,480
domain 211	n	14,484	57,936	231,744	3,027,156
	nnz	126,125	607,718	2,606,510	34,753,748
domain 212	n	16,524	66,096	264,384	3,453,516
	nnz	143,925	693,348	3,335,460	39,648,678
domain 221	n	8,100	32,400	129,600	1,692,900
	nnz	70,345	339,670	1,457,470	19,435,420

Conclusions

Research question, experimental setup & planning

$$\text{Speedup} = \frac{\text{execution time best performing sequential algorithm}}{\text{execution time best performing parallel algorithm}}$$

- Execution time
- Iteration count

Conclusions

Research question, experimental setup & planning

$$\text{Speedup} = \frac{\text{execution time } \textbf{best performing sequential algorithm}}{\text{execution time best performing parallel algorithm}}$$

- Benchmark results

Conclusions

Research question, experimental setup & planning

$$\text{Speedup} = \frac{\text{execution time best performing sequential algorithm}}{\text{execution time } \textbf{best performing parallel algorithm}}$$

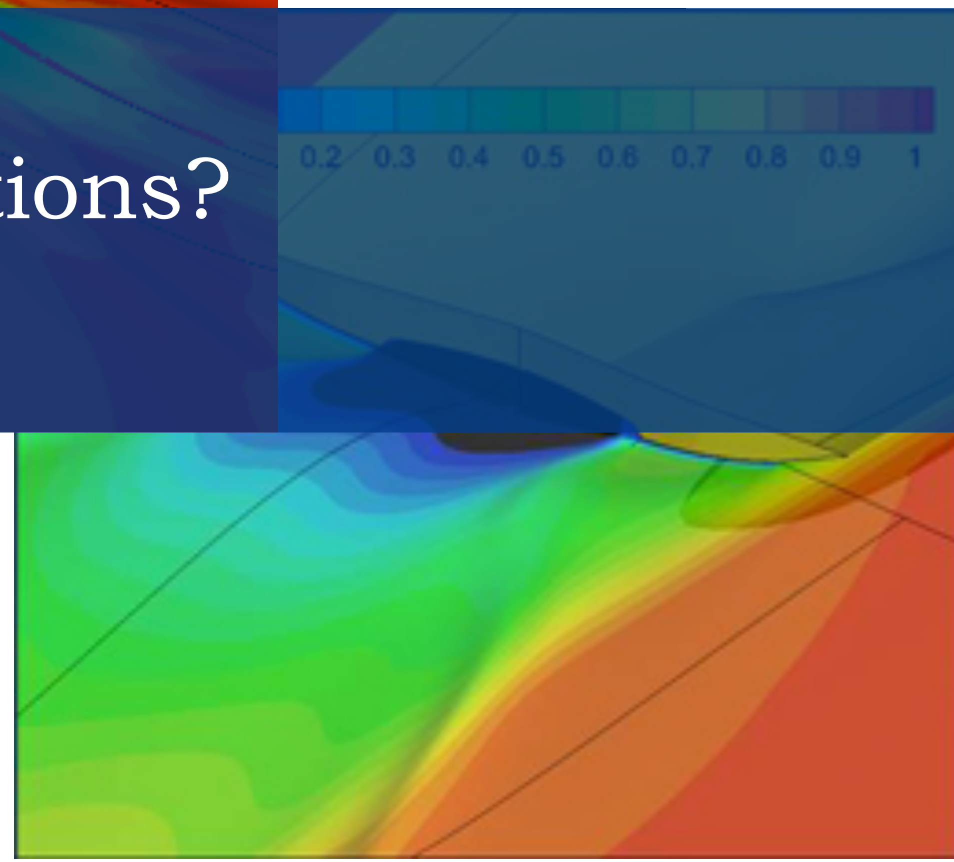
- Benchmark results
- Comparative study
    - PARALUTION
    - Several iterations (performance tuning cycle)

Conclusions

Research question, experimental setup & planning

$$\text{Speedup} = \frac{\text{execution time best performing sequential algorithm}}{\text{execution time best performing parallel algorithm}}$$

- Benchmark results
- Comparative study
    - PARALUTION
    - Several iterations (performance tuning cycle)
- CUDA implementation of best candidate…



Thank you - Questions?