Het ontwikkelen van een CUDA solver voor grote ijle matrices voor MARIN

Martijn de Jong

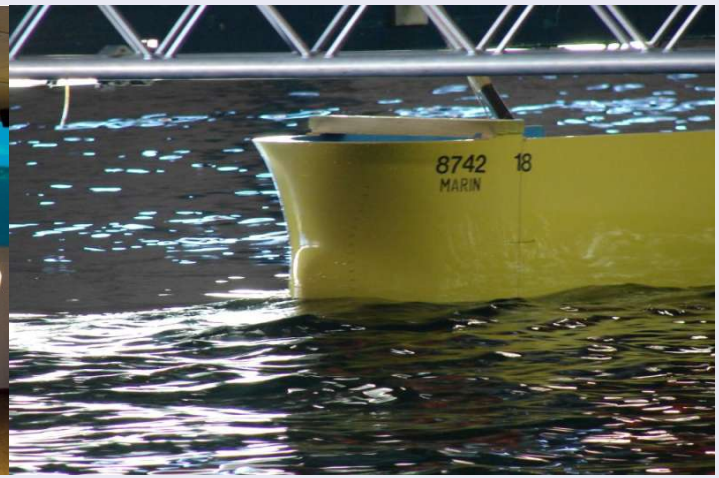
13 februari 2012

Inhoud

Achtergronden, model, probleem formulering

Maritime Research Institute Netherlands (MARIN)

- ✓ Wageningen, opgericht in 1932
- ✓ Onafhankelijk instituut
- ✓ 300 werknemers
- ✓ Innovatieve producten voor offshore en scheepvaart
- ✓ Simulatoren, trainingen, onderzoek, advies



Interactieve Golven

Huidige simulator:

- ✓ Deterministisch golvenmodel

Interactieve Golven

Huidige simulator:

- ✓ Deterministisch golvenmodel
- ✓ Bewegingen van het schip zijn realistisch

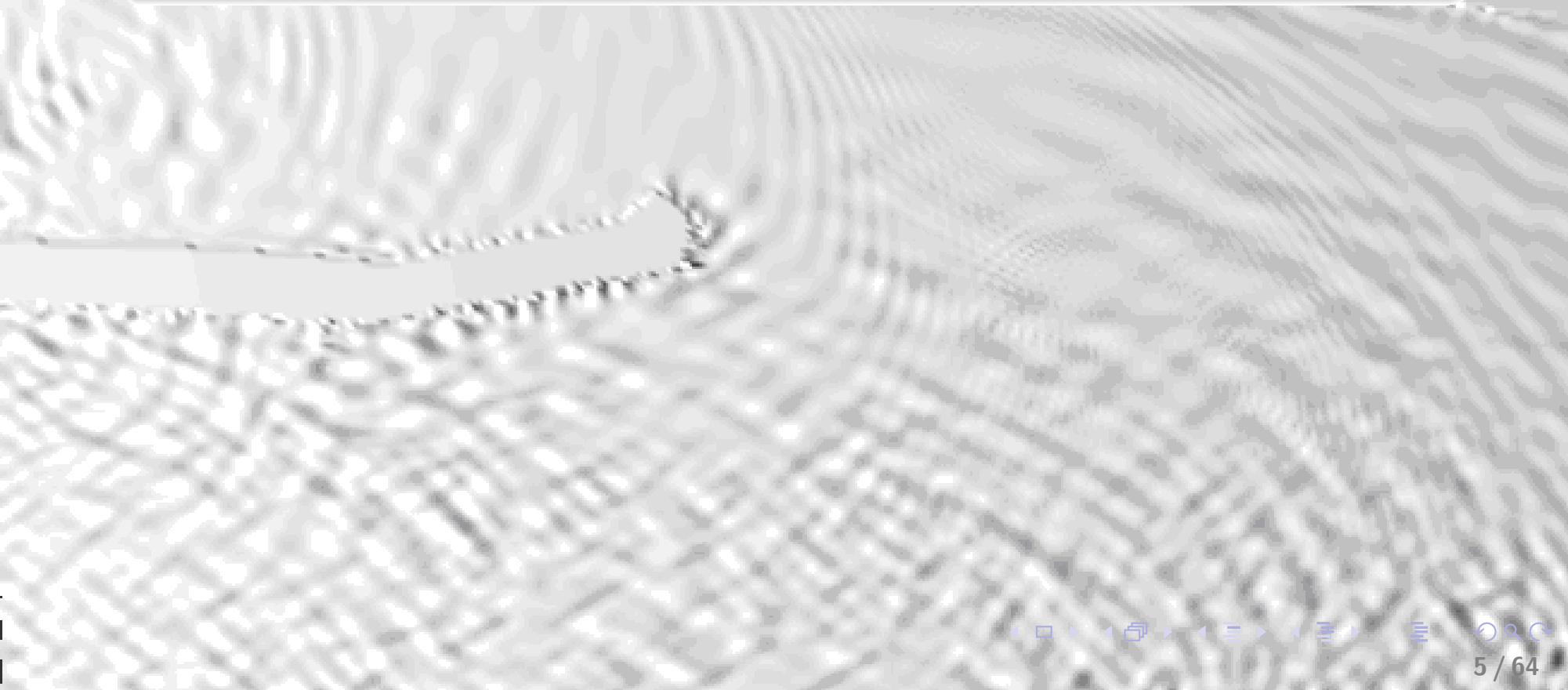
Interactieve Golven

Huidige simulator:

- ✓ Deterministisch golvenmodel
- ✓ Bewegingen van het schip zijn realistisch
- ✓ Golven reageren niet op het schip, pieren, kades, etc.

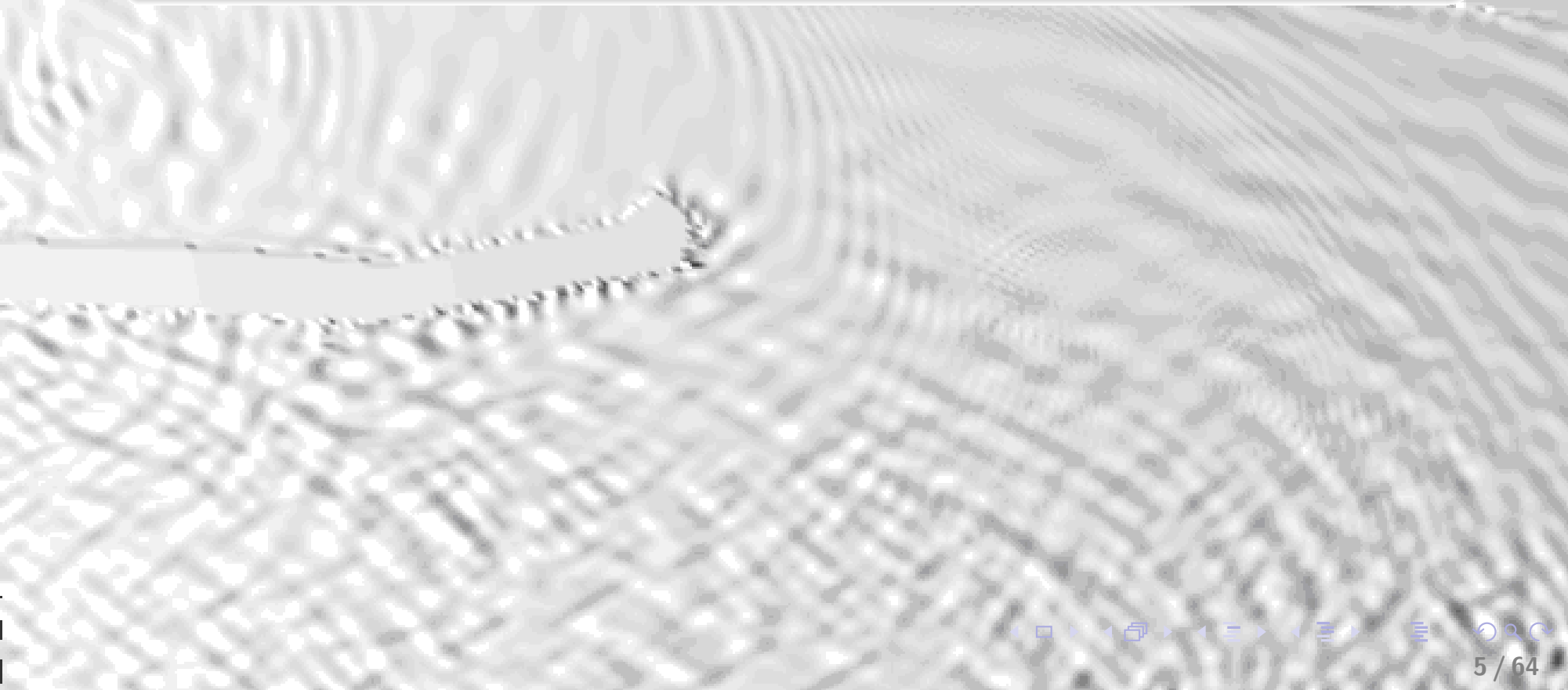
Verbeterpunten

- ✓ Realistisch golvenmodel



Verbeterpunten

- ✓ Realistisch golvenmodel
- ✓ De golven reageren ook op het schip (en andere obstakels)



Verbeterpunten

- ✓ Realistisch golvenmodel
- ✓ De golven reageren ook op het schip (en andere obstakels)
- ✓ Vandaar de projectnaam: “Interactieve Golven”

Verbeterpunten

- ✓ Realistisch golvenmodel
- ✓ De golven reageren ook op het schip (en andere obstakels)
- ✓ Vandaar de projectnaam: “Interactieve Golven”

Nieuw golvenmodel

- ✓ Variationeel Boussinesq model (VBM)

Verbeterpunten

- ✓ Realistisch golvenmodel
- ✓ De golven reageren ook op het schip (en andere obstakels)
- ✓ Vandaar de projectnaam: "Interactieve Golven"

Nieuw golvenmodel

- ✓ Variationeel Boussinesq model (VBM)
- ✓ Voorgesteld door "golvenguru" Gert Klopman

Verbeterpunten

- ✓ Realistisch golvenmodel
- ✓ De golven reageren ook op het schip (en andere obstakels)
- ✓ Vandaar de projectnaam: “Interactieve Golven”

Nieuw golvenmodel

- ✓ Variationeel Boussinesq model (VBM)
- ✓ Voorgesteld door “golvenguru” Gert Klopman
- ✓ Nadeel: veel meer rekenwerk

Verbeterpunten

- ✓ Realistisch golvenmodel
- ✓ De golven reageren ook op het schip (en andere obstakels)
- ✓ Vandaar de projectnaam: “Interactieve Golven”

Nieuw golvenmodel

- ✓ Variationeel Boussinesq model (VBM)
- ✓ Voorgesteld door “golvenguru” Gert Klopman
- ✓ Nadeel: veel meer rekenwerk
- ✓ Testomgeving: de `lin_wacu` software (C++)

Verbeterpunten

- ✓ Realistisch golvenmodel
- ✓ De golven reageren ook op het schip (en andere obstakels)
- ✓ Vandaar de projectnaam: "Interactieve Golven"

Nieuw golvenmodel

- ✓ Variationeel Boussinesq model (VBM)
- ✓ Voorgesteld door "golvenguru" Gert Klopman
- ✓ Nadeel: veel meer rekenwerk
- ✓ Testomgeving: de `lin_wacu` software (C++)

Eerder onderzoek vanuit TU Delft

- ✓ Elwin van 't Wout (afstudeerwerk, 2009)

  (Verbetering van de solver + beschrijving VBM)

Gelineariseerde VBM vergelijkingen

$$\frac{\partial \zeta}{\partial t} + \nabla \cdot (\zeta \mathbf{U} + h\nabla\varphi - h\mathcal{D}\nabla\psi) = 0, \tag{1a}$$

$$\frac{\partial \varphi}{\partial t} + \mathbf{U} \cdot \nabla\varphi + g\zeta = -P_s, \tag{1b}$$

$$\mathcal{M}\psi + \nabla \cdot (h\mathcal{D}\nabla\varphi - \mathcal{N}\nabla\psi) = 0. \tag{1c}$$

Hierin is:

ζ	waterhoogte	h	waterdiepte
φ	snelheidspotentiaal	$\mathbf{U}$	stroming
ψ	verticale structuur	P_s	"drukpuls" schip
g	gravitieversnelling	$\mathcal{D}, \mathcal{M}, \mathcal{N}$	model parameters

Discretisatiemethode

- ✓ Rechthoekig equidistant rekenrooster

  (noodzakelijk volgens Gert Klopman)
- ✓ Eindige volume methode (FVM) voor plaats
- ✓ Leapfrog methode voor tijdsintegratie

Discretisatiemethode

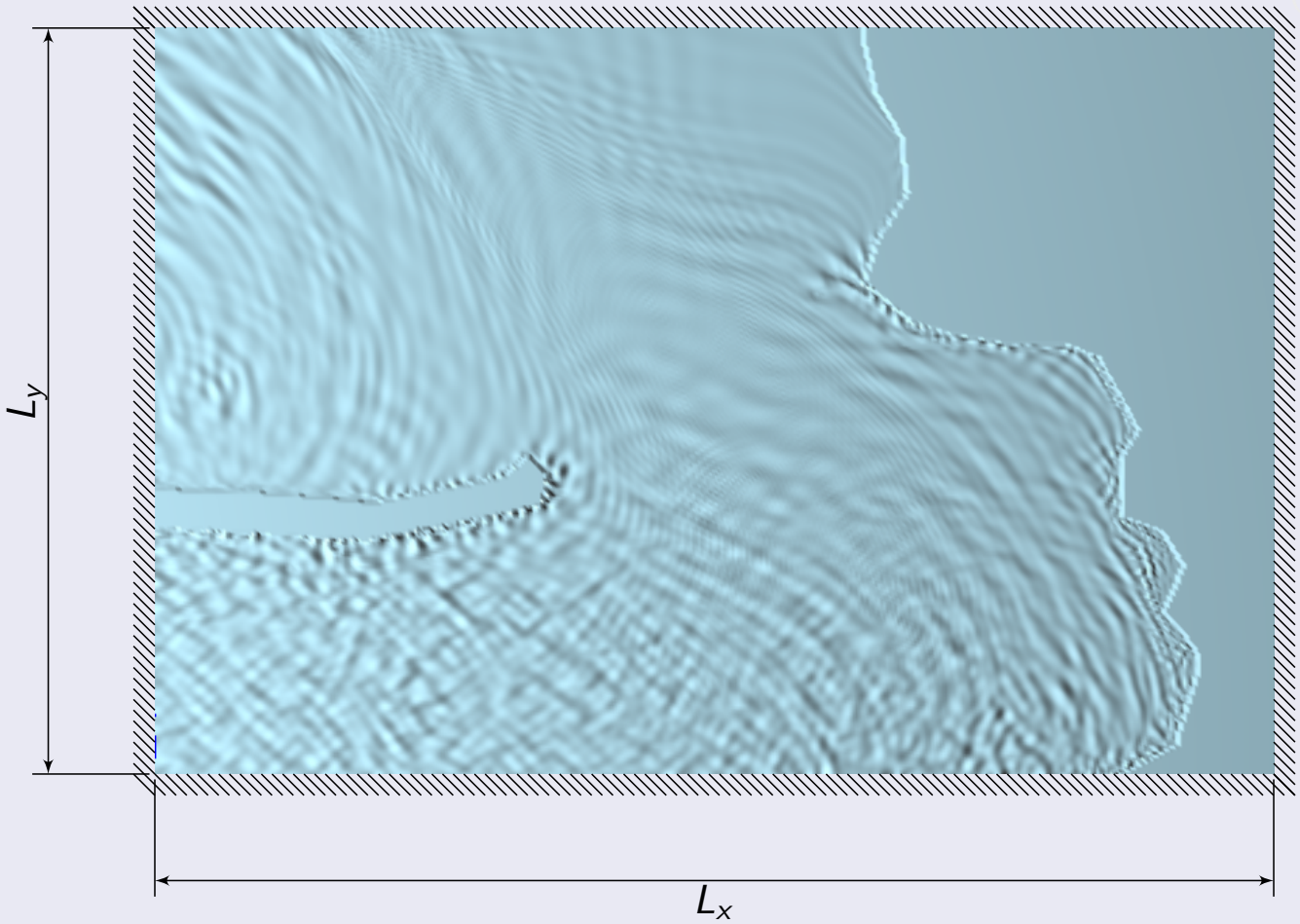
- ✓ Rechthoekig equidistant rekenrooster

  (noodzakelijk volgens Gert Klopman)
- ✓ Eindige volume methode (FVM) voor plaats
- ✓ Leapfrog methode voor tijdsintegratie

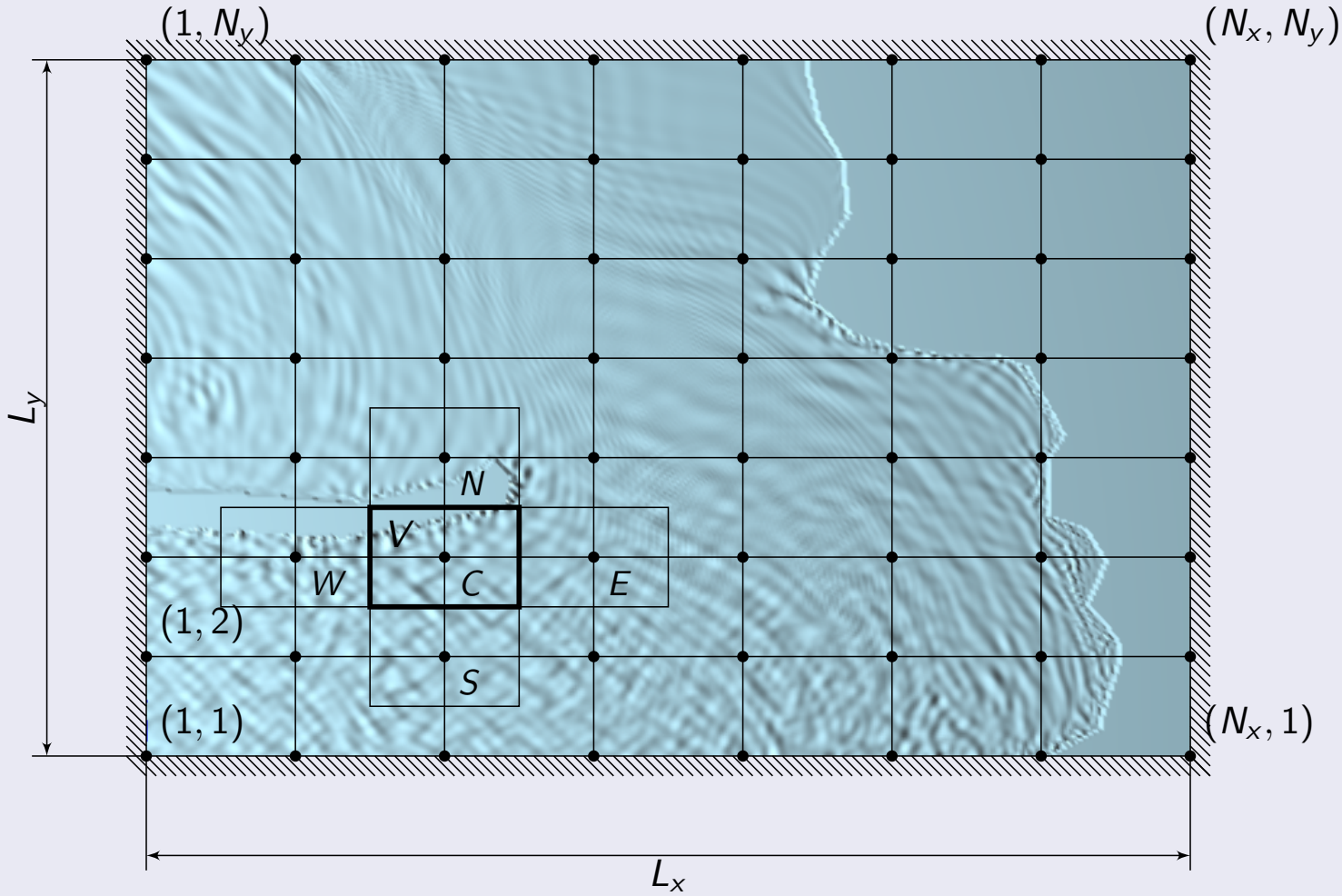
Het resultaat zal zijn:

- ✓ Tijdsvergelijking: $\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = L\mathbf{q} + \mathbf{f}$
- ✓ Systeem: $S\vec{\psi} = \mathbf{b}$

Rekenrooster



Rekenrooster + FVM



Na discretisatie:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\vec{\zeta}\\ \vec{\varphi}\\ \mathbf{0}\end{bmatrix}+\begin{bmatrix}S_{\zeta\zeta} & S_{\zeta\varphi} & S_{\zeta\psi}\\ S_{\varphi\zeta} & S_{\varphi\varphi} & S_{\varphi\psi}\\ S_{\psi\zeta} & S_{\psi\varphi} & S_{\psi\psi}\end{bmatrix}\begin{bmatrix}\vec{\zeta}\\ \vec{\varphi}\\ \vec{\psi}\end{bmatrix}=\begin{bmatrix}\mathbf{0}\\ -\vec{P}_s\\ \mathbf{0}\end{bmatrix}, \tag{2}$$

oftewel,

$$\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t}=L\mathbf{q}+\mathbf{f}, \tag{3a}$$

$$S\vec{\psi}=\mathbf{b}. \tag{3b}$$

(Voor details: zie verslag van Elwin van 't Wout)

Het probleem

Oplossen van het systeem

$$S\vec{\psi} = \mathbf{b},$$

waarin de matrix S gegeven is door een 5-punts stencil:

$$\begin{bmatrix} 0 & -\frac{\Delta x}{\Delta y}\overline{\mathcal{N}_N} & 0 \\ -\frac{\Delta y}{\Delta x}\overline{\mathcal{N}_W} & \frac{\Delta x}{\Delta y}\overline{\mathcal{N}_N} + \frac{\Delta y}{\Delta x}\overline{\mathcal{N}_E} + \Delta x \Delta y \mathcal{M}_C + \frac{\Delta x}{\Delta y}\overline{\mathcal{N}_S} + \frac{\Delta y}{\Delta x}\overline{\mathcal{N}_W} & -\frac{\Delta y}{\Delta x}\overline{\mathcal{N}_E} \\ 0 & -\frac{\Delta x}{\Delta y}\overline{\mathcal{N}_S} & 0 \end{bmatrix}.$$

Het probleem

Oplossen van het systeem

$$S\vec{\psi} = \mathbf{b},$$

waarin de matrix S gegeven is door een 5-punts stencil:

$$\begin{bmatrix} 0 & -\frac{\Delta x}{\Delta y}\overline{\mathcal{N}_N} & 0 \\ -\frac{\Delta y}{\Delta x}\overline{\mathcal{N}_W} & \frac{\Delta x}{\Delta y}\overline{\mathcal{N}_N} + \frac{\Delta y}{\Delta x}\overline{\mathcal{N}_E} + \Delta x \Delta y \mathcal{M}_C + \frac{\Delta x}{\Delta y}\overline{\mathcal{N}_S} + \frac{\Delta y}{\Delta x}\overline{\mathcal{N}_W} & -\frac{\Delta y}{\Delta x}\overline{\mathcal{N}_E} \\ 0 & -\frac{\Delta x}{\Delta y}\overline{\mathcal{N}_S} & 0 \end{bmatrix}.$$

Dit moet in

real-time!

Probleemgrootte en real-time eis

✓ Real-time: $20\,fps \rightarrow \Delta t = 50\,ms$

Probleemgrootte en real-time eis

- ✓ Real-time: $20\,fps \rightarrow \Delta t = 50\,ms$
- ✓ Gebied van 10 bij 10 km, maaswijdte $h = 5$ m

Probleemgrootte en real-time eis

- ✓ Real-time: $20\,fps \rightarrow \Delta t = 50\,ms$
- ✓ Gebied van 10 bij 10 km, maaswijdte $h = 5$ m

  $\rightarrow$ Grid van 2000 bij 2000 gridpunten

Probleemgrootte en real-time eis

- ✓ Real-time: $20\,fps \rightarrow \Delta t = 50\,ms$
- ✓ Gebied van 10 bij 10 km, maaswijdte $h = 5$ m
  - $\rightarrow$ Grid van 2000 bij 2000 gridpunten
  - $\rightarrow$ Matrix S van 4 miljoen bij 4 miljoen elementen (!)

Probleemgrootte en real-time eis

- ✓ Real-time: $20\,fps \;\rightarrow\; \Delta t = 50\,ms$
- ✓ Gebied van 10 bij 10 km, maaswijdte $h = 5$ m
  - $\rightarrow$ Grid van 2000 bij 2000 gridpunten
  - $\rightarrow$ Matrix S van 4 miljoen bij 4 miljoen elementen (!)

Dus: we moeten $20\times$ per seconde een systeem $S\vec{\psi} = \mathbf{b}$ oplossen bestaande uit miljoenen vergelijkingen.

Probleemgrootte en real-time eis

- ✓ Real-time: $20\,fps \;\rightarrow\; \Delta t = 50\,ms$
- ✓ Gebied van 10 bij 10 km, maaswijdte $h = 5$ m
  - $\rightarrow$ Grid van 2000 bij 2000 gridpunten
  - $\rightarrow$ Matrix S van 4 miljoen bij 4 miljoen elementen (!)

Dus: we moeten $20\times$ per seconde een systeem $S\vec{\psi} = \mathbf{b}$ oplossen bestaande uit miljoenen vergelijkingen.

Voorheen

Met de snelste solver toen beschikbaar (de RRB-solver op de CPU) kon real-time worden uitgerekend een gebied van 400 bij 200 gridpunten (2 bij 1 km).

Wat hebben we nodig?

Wat hebben we nodig?

✓ Een methode (lees: algoritme) die het probleem $S\vec{\psi} = \mathbf{b}$ "slim" weet op te lossen (lees: in weinig stappen).

Wat hebben we nodig?

- ✓ Een methode (lees: algoritme) die het probleem $S\vec{\psi} = \mathbf{b}$ "slim" weet op te lossen (lees: in weinig stappen).
- ✓ Heel veel rekenkracht

Wat hebben we nodig?

- ✓ Een methode (lees: algoritme) die het probleem $S\vec{\psi} = \mathbf{b}$ "slim" weet op te lossen (lees: in weinig stappen).
- ✓ Heel veel rekenkracht

$$\Longrightarrow \text{Graphics Processing Unit (GPU)}$$
(= videokaart in PC)

Oplostechniek

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

Matrix S is:

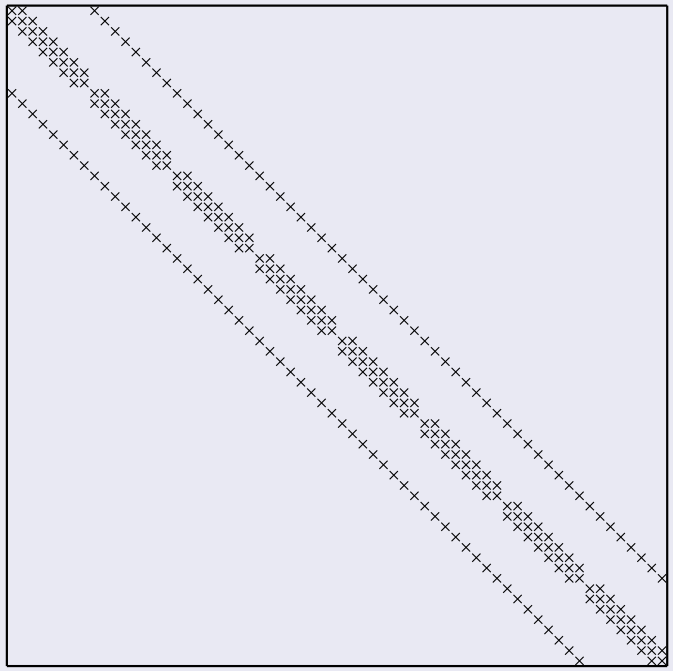
Oplostechniek

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

Matrix S is:

- ✓ pentadiagonaal, dus heel ijl (leeg/sparse)

$S =$ 

Oplostechniek

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

Matrix S is:

- ✓ pentadiagonaal, dus heel ijl (leeg/sparse)
- ✓ (strikt) diagonaal dominant

$$\begin{bmatrix} 0 & & -\frac{\Delta x}{\Delta y}\overline{\mathcal{N}_N} & & 0 \\ -\frac{\Delta y}{\Delta x}\overline{\mathcal{N}_W} & \frac{\Delta x}{\Delta y}\overline{\mathcal{N}_N} + \frac{\Delta y}{\Delta x}\overline{\mathcal{N}_E} + \Delta x \Delta y \mathcal{M}_C + \frac{\Delta x}{\Delta y}\overline{\mathcal{N}_S} + \frac{\Delta y}{\Delta x}\overline{\mathcal{N}_W} & & -\frac{\Delta y}{\Delta x}\overline{\mathcal{N}_E} \\ 0 & & -\frac{\Delta x}{\Delta y}\overline{\mathcal{N}_S} & & 0 \end{bmatrix}$$

Oplostechniek

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

Matrix S is:

- ✓ pentadiagonaal, dus heel ijl (leeg/sparse)
- ✓ (strikt) diagonaal dominant

  (voor kleine maaswijdtes h niet erg sterk)

Oplostechniek

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

Matrix S is:

- ✓ pentadiagonaal, dus heel ijl (leeg/sparse)
- ✓ (strikt) diagonaal dominant

  (voor kleine maaswijdtes h niet erg sterk)
- ✓ symmetrisch

Oplostechniek

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

Matrix S is:

- ✓ pentadiagonaal, dus heel ijl (leeg/sparse)
- ✓ (strikt) diagonaal dominant

  (voor kleine maaswijdtes h niet erg sterk)
- ✓ symmetrisch
- ✓ positief-definiet

Oplostechniek

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

Matrix S is:

- ✓ pentadiagonaal, dus heel ijl (leeg/sparse)
- ✓ (strikt) diagonaal dominant

  (voor kleine maaswijdtes h niet erg sterk)
- ✓ symmetrisch
- ✓ positief-definiet

Goede methode voor een systeem $S\vec{\psi} = \mathbf{b}$ waarin S SPD is:

Preconditioned Conjugate Gradients (PCG) methode

Conjugate Gradients (CG)

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

✓ CG is een iteratieve methode:

Conjugate Gradients (CG)

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

✓ CG is een iteratieve methode:

Startvector $\vec{\psi}_0$

Conjugate Gradients (CG)

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

✓ CG is een iteratieve methode:

Startvector $\vec{\psi}_0$

Iteraties $\vec{\psi}_1, \vec{\psi}_2, \vec{\psi}_3, \ldots$ zolang als nodig totdat $\vec{\psi}_i$ "goed genoeg lijkt" op de echte oplossing $\vec{\psi}$.

Conjugate Gradients (CG)

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

- ✓ CG is een iteratieve methode:

  Startvector $\vec{\psi}_0$

  Iteraties $\vec{\psi}_1, \vec{\psi}_2, \vec{\psi}_3, \ldots$ zolang als nodig totdat $\vec{\psi}_i$ "goed genoeg lijkt" op de echte oplossing $\vec{\psi}$.
- ✓ Wanneer is dat?

Conjugate Gradients (CG)

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

- ✓ CG is een iteratieve methode:

  Startvector $\vec{\psi}_0$

  Iteraties $\vec{\psi}_1, \vec{\psi}_2, \vec{\psi}_3, \ldots$ zolang als nodig totdat $\vec{\psi}_i$ "goed genoeg lijkt" op de echte oplossing $\vec{\psi}$.
- ✓ Wanneer is dat? Als (b.v.)

$$\|\mathbf{b} - S\vec{\psi}_i\| < \mathit{psitol} \cdot \|\mathbf{b} - S\vec{\psi}_0\|.$$

(dit heet het stopcriterium)

Conjugate Gradients (CG)

Systeem

$$S\vec{\psi} = \mathbf{b}.$$

- ✓ CG is een iteratieve methode:

  Startvector $\vec{\psi}_0$

  Iteraties $\vec{\psi}_1, \vec{\psi}_2, \vec{\psi}_3, \ldots$ zolang als nodig totdat $\vec{\psi}_i$ "goed genoeg lijkt" op de echte oplossing $\vec{\psi}$.
- ✓ Wanneer is dat? Als (b.v.)

$$\|\mathbf{b} - S\vec{\psi}_i\| < \mathit{psitol} \cdot \|\mathbf{b} - S\vec{\psi}_0\|.$$

  (dit heet het stopcriterium)
- ✓ Maatgevend is het aantal iteraties tot convergentie

Preconditioned Conjugate Gradients (PCG)

Bekijk

$$M^{-1}S\vec{\psi} = M^{-1}\mathbf{b}$$

in plaats van

$$S\vec{\psi} = \mathbf{b}.$$

Preconditioned Conjugate Gradients (PCG)

Bekijk

$$M^{-1}S\vec{\psi} = M^{-1}\mathbf{b}$$

in plaats van

$$S\vec{\psi} = \mathbf{b}.$$

✓ M^{-1} (of M) is de zogenaamde "preconditioner"

Preconditioned Conjugate Gradients (PCG)

Bekijk

$$M^{-1}S\vec{\psi} = M^{-1}\mathbf{b}$$

in plaats van

$$S\vec{\psi} = \mathbf{b}.$$

- ✓ M^{-1} (of M) is de zogenaamde "preconditioner"
- ✓ PCG convergeert (veel!) sneller dan CG

Preconditioned Conjugate Gradients (PCG)

Bekijk

$$M^{-1}S\vec{\psi} = M^{-1}\mathbf{b}$$

in plaats van

$$S\vec{\psi} = \mathbf{b}.$$

- ✓ M^{-1} (of M) is de zogenaamde "preconditioner"
- ✓ PCG convergeert (veel!) sneller dan CG
- ✓ Vraag: Wat nemen we voor M^{-1}?

Goede preconditioner: RRB-methode

RRB staat voor: “Repeated Red-Black”.

Goede preconditioner: RRB-methode

RRB staat voor: "Repeated Red-Black".

✓ De RRB-methode bepaalt een factorisatie

$$S = LDL^T + R \quad \Longrightarrow \quad M = LDL^T \approx S$$

Goede preconditioner: RRB-methode

RRB staat voor: "Repeated Red-Black".

✓ De RRB-methode bepaalt een factorisatie

$$S = LDL^T + R \quad \Longrightarrow \quad M = LDL^T \approx S$$

✓ De RRB-methode wordt gebruikt als preconditioner voor CG

Goede preconditioner: RRB-methode

RRB staat voor: "Repeated Red-Black".

- ✓ De RRB-methode bepaalt een factorisatie

$$S = LDL^T + R \quad \Longrightarrow \quad M = LDL^T \approx S$$

- ✓ De RRB-methode wordt gebruikt als preconditioner voor CG
- ✓ De resulterende solver is een "Multigrid"-achtige solver

  (#CG-iteraties groeit heel weinig met groter wordende problemen)

Goede preconditioner: RRB-methode

RRB staat voor: "Repeated Red-Black".

- ✓ De RRB-methode bepaalt een factorisatie

$$S = LDL^T + R \quad \Longrightarrow \quad M = LDL^T \approx S$$

- ✓ De RRB-methode wordt gebruikt als preconditioner voor CG
- ✓ De resulterende solver is een "Multigrid"-achtige solver

  (#CG-iteraties groeit heel weinig met groter wordende problemen)
- ✓ We noemen de solver:

De RRB-solver

De RRB-solver

Basisprincipes achter de RRB-solver

De RRB-solver

Basisprincipes achter de RRB-solver

✓ Herhaalde rood-zwart nummering

De RRB-solver

Basisprincipes achter de RRB-solver

- ✓ Herhaalde rood-zwart nummering
- ✓ CG voor helft onbekenden

De RRB-solver

Basisprincipes achter de RRB-solver

- ✓ Herhaalde rood-zwart nummering
- ✓ CG voor helft onbekenden

Met een rood-zwart nummering kan de matrix S geschreven worden als een 2×2 blokmatrix, zodat $S\vec{\psi} = \mathbf{b}$ wordt:

$$\begin{bmatrix} D_z & S_{zr} \\ S_{rz} & D_r \end{bmatrix} \begin{bmatrix} \vec{\psi}_z \\ \vec{\psi}_r \end{bmatrix} = \begin{bmatrix} \mathbf{b}_z \\ \mathbf{b}_r \end{bmatrix}.$$

Hierin zijn D_r en D_z diagonaal matrices en $S_{rz} = S_{zr}^T$ matrices met 4 diagonalen.
Vervolgens passen we *Gauss eliminatie* toe:

$$\begin{bmatrix} D_z & S_{zr} \\ 0 & D_r - S_{rz} D_z^{-1} S_{zr} \end{bmatrix} \begin{bmatrix} \vec{\psi}_z \\ \vec{\psi}_r \end{bmatrix} = \begin{bmatrix} \mathbf{b}_z \\ \mathbf{b}_r - S_{rz} D_z^{-1} S_{zr} \mathbf{b}_z \end{bmatrix}.$$

De matrix $S_1 := D_r - S_{rz} D_z^{-1} S_{zr}$ heet het 1^e Schur complement en wordt gegeven door een 9-punts stencil.

CG voor half onbekenden

In plaats van $S\vec{\psi} = \mathbf{b}$ lossen we op:

$$S_1\vec{\psi}_r = \mathbf{b}_1, \tag{4}$$

waarin $\mathbf{b}_1 := \mathbf{b}_r - S_{rz}D_z^{-1}\mathbf{b}_z$.

CG voor half onbekenden

In plaats van $S\vec{\psi} = \mathbf{b}$ lossen we op:

$$S_1\vec{\psi}_r = \mathbf{b}_1, \tag{4}$$

waarin $\mathbf{b}_1 := \mathbf{b}_r - S_{rz}D_z^{-1}\mathbf{b}_z$.

- ✓ Dit is een probleem op uitsluitend de rode punten

CG voor half onbekenden

In plaats van $S\vec{\psi} = \mathbf{b}$ lossen we op:

$$S_1\vec{\psi}_r = \mathbf{b}_1, \tag{4}$$

waarin $\mathbf{b}_1 := \mathbf{b}_r - S_{rz}D_z^{-1}\mathbf{b}_z$.

- ✓ Dit is een probleem op uitsluitend de rode punten
- ✓ Niet automatisch $2\times$ goedkoper!

CG voor half onbekenden

In plaats van $S\vec{\psi} = \mathbf{b}$ lossen we op:

$$S_1\vec{\psi}_r = \mathbf{b}_1, \tag{4}$$

waarin $\mathbf{b}_1 := \mathbf{b}_r - S_{rz}D_z^{-1}\mathbf{b}_z$.

- ✓ Dit is een probleem op uitsluitend de rode punten
- ✓ Niet automatisch $2\times$ goedkoper!
- ✓ Met de oplossing voor $\vec{\psi}_r$ kunnen we $\vec{\psi}_z$ berekenen via substitutie

CG voor half onbekenden

In plaats van $S\vec{\psi} = \mathbf{b}$ lossen we op:

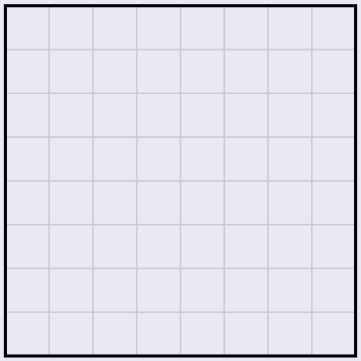
$$S_1\vec{\psi}_r = \mathbf{b}_1, \tag{4}$$

waarin $\mathbf{b}_1 := \mathbf{b}_r - S_{rz}D_z^{-1}\mathbf{b}_z$.

- ✓ Dit is een probleem op uitsluitend de rode punten
- ✓ Niet automatisch $2\times$ goedkoper!
- ✓ Met de oplossing voor $\vec{\psi}_r$ kunnen we $\vec{\psi}_z$ berekenen via substitutie
- ✓ Samenvattend:
  - bereken $\mathbf{b}_1 = \mathbf{b}_r - S_{rb}D_b^{-1}\mathbf{b}_z$
  - pas CG toe op (4) $\implies$ resultaat: $\vec{\psi}_r$
  - bereken $\vec{\psi}_z$ via $\vec{\psi}_z = D_z^{-1}(\mathbf{b}_z - S_{zr}\vec{\psi}_r)$

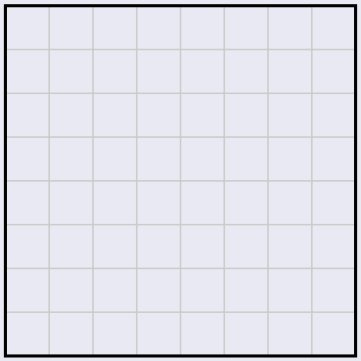
De RRB-nummering

Voorbeeld: grid met 8×8 gridpunten



De RRB-nummering

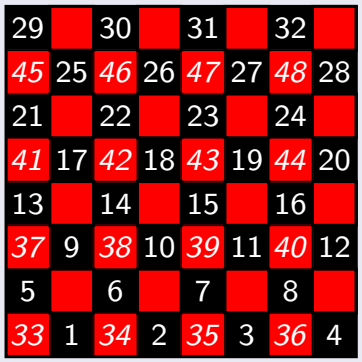
Voorbeeld: grid met 8×8 gridpunten



$$\Longrightarrow \text{Matrix } S \in \mathbb{R}^{64\times 64}$$

De RRB-nummering

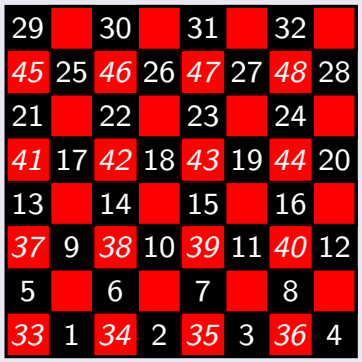
Voorbeeld: grid met 8×8 gridpunten



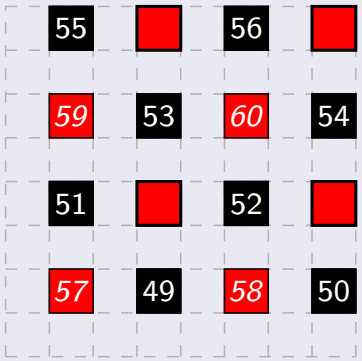
(1)

De RRB-nummering

Voorbeeld: grid met 8×8 gridpunten



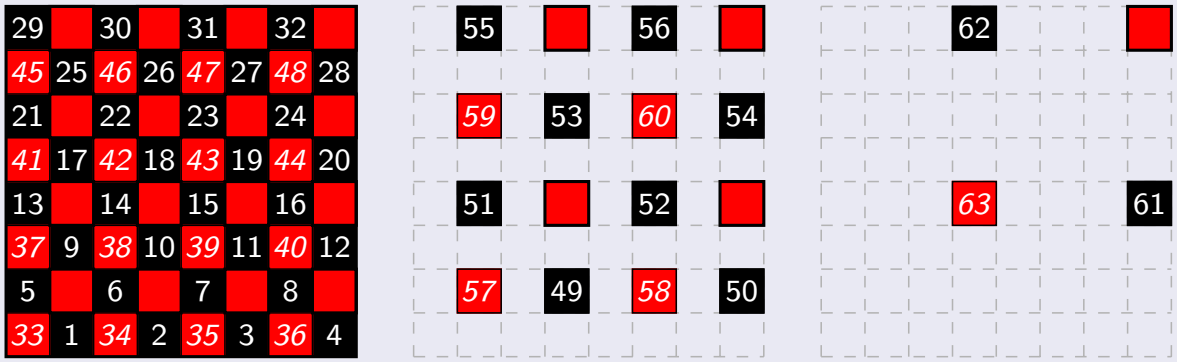
(1)



(2)

De RRB-nummering

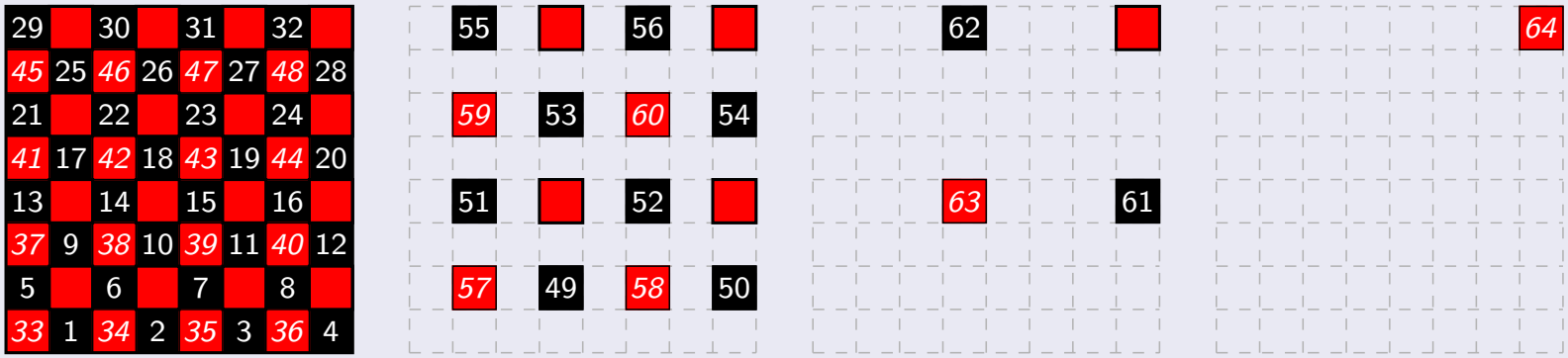
Voorbeeld: grid met 8×8 gridpunten



(1) (2) (3)

De RRB-nummering

Voorbeeld: grid met 8×8 gridpunten



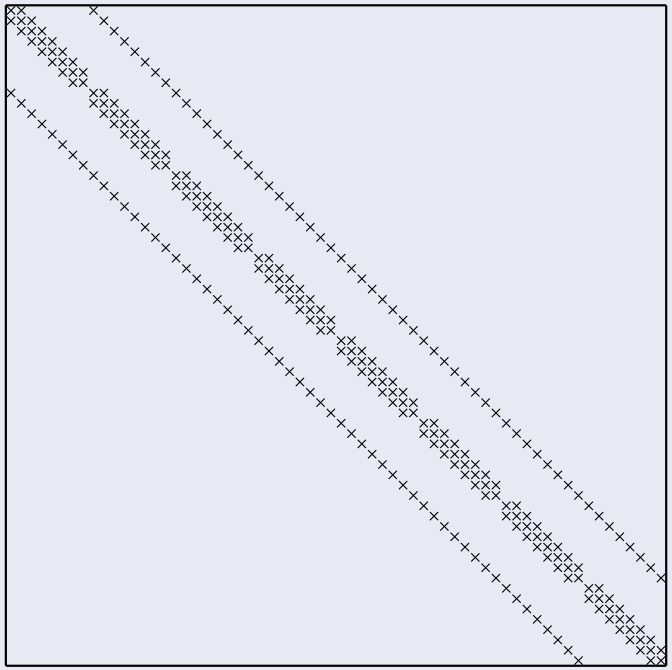
(1) (2) (3) (4)

De RRB-nummering

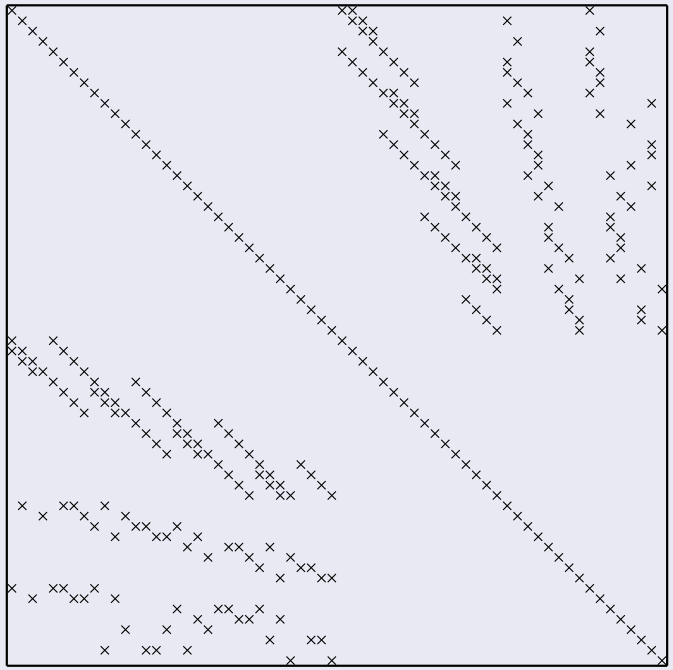
Alle niveaus samen:

29	55	30	62	31	56	32	64
45	25	46	26	47	27	48	28
21	59	22	53	23	60	24	54
41	17	42	18	43	19	44	20
13	51	14	63	15	52	16	61
37	9	38	10	39	11	40	12
5	57	6	49	7	58	8	50
33	1	34	2	35	3	36	4

Invloed op ijlheidspatroon



wordt



Werk voor 1 PCG-iteratie

While-loop	
Vector-update	$\mathbf{p} = \mathbf{z} + \beta\mathbf{p}$
Matrix-vector	$\mathbf{q} = S_1\mathbf{p}$
Inproduct	$\sigma = \langle \mathbf{p}, \mathbf{q} \rangle$
Vector-update	$\vec{\psi} = \vec{\psi} + \alpha\mathbf{p}$
Vector-update	$\mathbf{r} = \mathbf{r} - \alpha\mathbf{q}$
Los op	$M\mathbf{z} = \mathbf{r}$
Inproduct	$\rho_{\text{nieuw}} = \langle \mathbf{r}, \mathbf{z} \rangle$
Einde while-loop	

Het hart van de RRB-solver

Preconditioner stap $M\mathbf{z} = \mathbf{r}$

We zagen: $M = LDL^T$ (incomplete RRB-factorisatie)

Preconditioner stap $M\mathbf{z} = \mathbf{r}$

We zagen: $M = LDL^T$ (incomplete RRB-factorisatie)

Opmerking: $z = M^{-1}r$, maar M^{-1} berekenen we in de regel nooit expliciet! (als M ijl is, dan is M^{-1} een volle matrix!)

Preconditioner stap $M\mathbf{z} = \mathbf{r}$

We zagen: $M = LDL^T$ (incomplete RRB-factorisatie)

Opmerking: $z = M^{-1}r$, maar M^{-1} berekenen we in de regel nooit expliciet! (als M ijl is, dan is M^{-1} een volle matrix!)

Het oplossen van $M\mathbf{z} = \mathbf{r}$ kan efficiënt stapsgewijs. Schrijf $\mathbf{y} := L^T\mathbf{z}$ en $\mathbf{x} := DL^T\mathbf{z} = D\mathbf{y}$, dan:

Preconditioner stap $M\mathbf{z} = \mathbf{r}$

We zagen: $M = LDL^T$ (incomplete RRB-factorisatie)

Opmerking: $z = M^{-1}r$, maar M^{-1} berekenen we in de regel nooit expliciet! (als M ijl is, dan is M^{-1} een volle matrix!)

Het oplossen van $M\mathbf{z} = \mathbf{r}$ kan efficiënt stapsgewijs. Schrijf $\mathbf{y} := L^T\mathbf{z}$ en $\mathbf{x} := DL^T\mathbf{z} = D\mathbf{y}$, dan:

1. Los op $L\mathbf{x} = \mathbf{r}$ met voorwaartse substitutie

Preconditioner stap $M\mathbf{z} = \mathbf{r}$

We zagen: $M = LDL^T$ (incomplete RRB-factorisatie)

Opmerking: $z = M^{-1}r$, maar M^{-1} berekenen we in de regel nooit expliciet! (als M ijl is, dan is M^{-1} een volle matrix!)

Het oplossen van $M\mathbf{z} = \mathbf{r}$ kan efficiënt stapsgewijs. Schrijf $\mathbf{y} := L^T\mathbf{z}$ en $\mathbf{x} := DL^T\mathbf{z} = D\mathbf{y}$, dan:

1. Los op $L\mathbf{x} = \mathbf{r}$ met voorwaartse substitutie
2. Bereken $\mathbf{y} = D^{-1}\mathbf{x}$

Preconditioner stap $M\mathbf{z} = \mathbf{r}$

We zagen: $M = LDL^T$ (incomplete RRB-factorisatie)

Opmerking: $z = M^{-1}r$, maar M^{-1} berekenen we in de regel nooit expliciet! (als M ijl is, dan is M^{-1} een volle matrix!)

Het oplossen van $M\mathbf{z} = \mathbf{r}$ kan efficiënt stapsgewijs. Schrijf $\mathbf{y} := L^T\mathbf{z}$ en $\mathbf{x} := DL^T\mathbf{z} = D\mathbf{y}$, dan:

1. Los op $L\mathbf{x} = \mathbf{r}$ met voorwaartse substitutie
2. Bereken $\mathbf{y} = D^{-1}\mathbf{x}$
3. Los op $L^T\mathbf{z} = \mathbf{y}$ met achterwaartse substitutie

Preconditioner stap $M\mathbf{z} = \mathbf{r}$

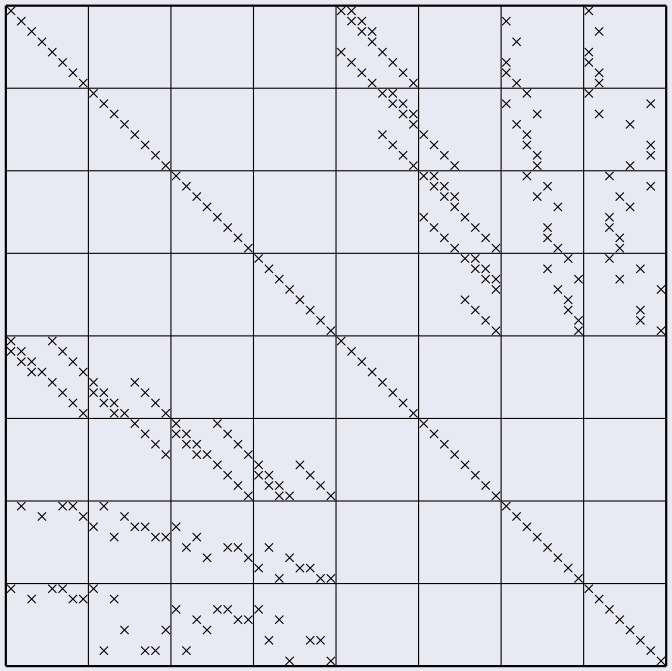
We zagen: $M = LDL^T$ (incomplete RRB-factorisatie)

Opmerking: $z = M^{-1}r$, maar M^{-1} berekenen we in de regel nooit expliciet! (als M ijl is, dan is M^{-1} een volle matrix!)

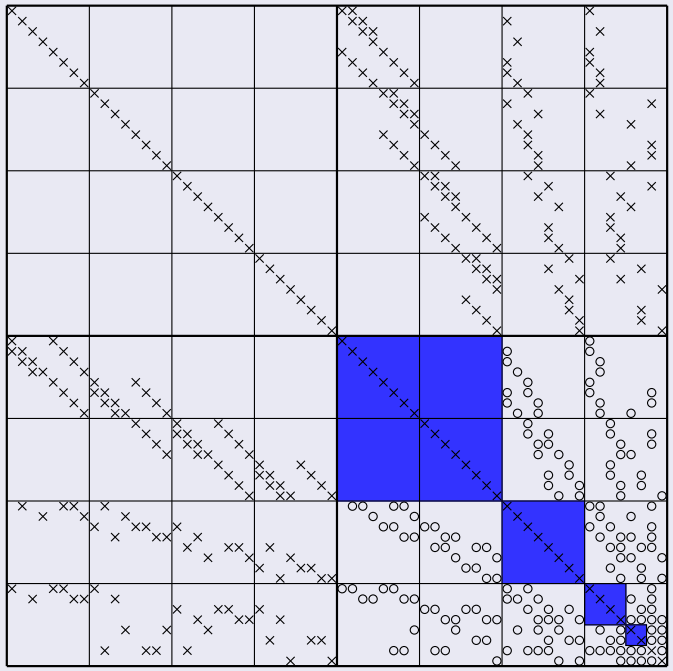
Het oplossen van $M\mathbf{z} = \mathbf{r}$ kan efficiënt stapsgewijs. Schrijf $\mathbf{y} := L^T\mathbf{z}$ en $\mathbf{x} := DL^T\mathbf{z} = D\mathbf{y}$, dan:

1. Los op $L\mathbf{x} = \mathbf{r}$ met voorwaartse substitutie
2. Bereken $\mathbf{y} = D^{-1}\mathbf{x}$
3. Los op $L^T\mathbf{z} = \mathbf{y}$ met achterwaartse substitutie

IJlheidspatroon van $L + D + L^T$



wordt



Korte samenvatting

We hebben gezien:

- ✓ MARIN wil een nog betere simulator
- ✓ Beter golvenmodel: VBM (Gert Klopman)
- ✓ Twee rekenstappen (per tijdframe):
  - Oplossen $S\vec{\psi} = \mathbf{b}$ (ons probleem)
  - Tijdsintegratie
- ✓ Matrix S is pentadiagonaal en SPD
- ✓ Real-time vraagt snelle methode + veel rekenkracht
- ✓ Oplosmethode: (PCG) RRB-solver (C++ versie is er al)
  - CG voor helft onbekenden
  - Herhaalde rood-zwart nummering
  - "Multigrid"-achtig

Implementatie in CUDA

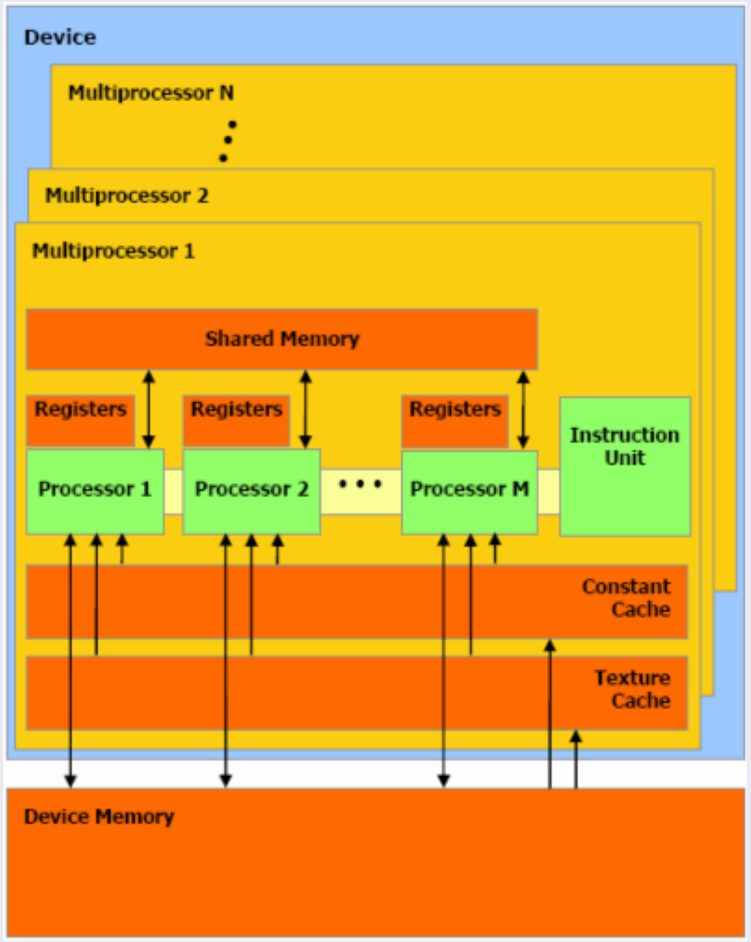
Wat is CUDA?

- ✓ CUDA staat voor: "Compute Unified Device Architecture"
- ✓ Uitbreiding op C programmeertaal
- ✓ Ontwikkeld door NVIDIA in 2006
- ✓ Uitsluitend voor NVIDIA GPU's

  (in tegenstelling tot b.v. OpenCL)

Architectuur van de NVIDIA GPU

- ✓ Global memory
- ✓ (Texture) cache
- ✓ Shared memory
- ✓ Registers
- ✓ Streaming Multiprocessor (SM)
- ✓ Streaming Processor (SP)



Programmeren op de GPU is lastig

Heel veel "regeltjes" waarmee je rekening moet houden, zoals:

- ✓ "Coalesced memory": data moet naast elkaar en netjes uitgelijnd liggen in het geheugen
- ✓ "Bank conflicts": meerdere threads mogen niet tegelijkertijd uit dezelfde geheugenbank lezen
- ✓ "Tiling": het werk moet optimaal worden verdeeld over de fysieke processors/cores (occupancy)
- ✓ "Registers versus shared memory": registers zijn sneller, maar zijn tevens erg schaars

Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

Het voornaamste probleem is “coalesced memory”.

Waarom?

Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

Het voornaamste probleem is “coalesced memory”.

Waarom?

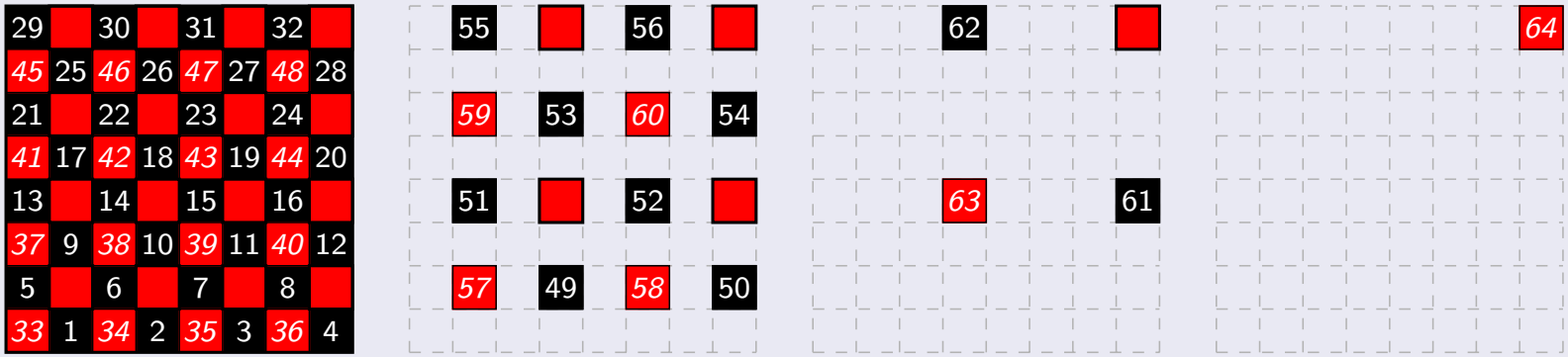
- ✓ Herinner je de herhaalde rood-zwart nummering: het aantal gridpunten in het volgende niveau wordt steeds $4\times$ kleiner

Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

Het voornaamste probleem is “coalesced memory”.

Waarom?

- ✓ Herinner je de herhaalde rood-zwart nummering: het aantal gridpunten in het volgende niveau wordt steeds 4× kleiner

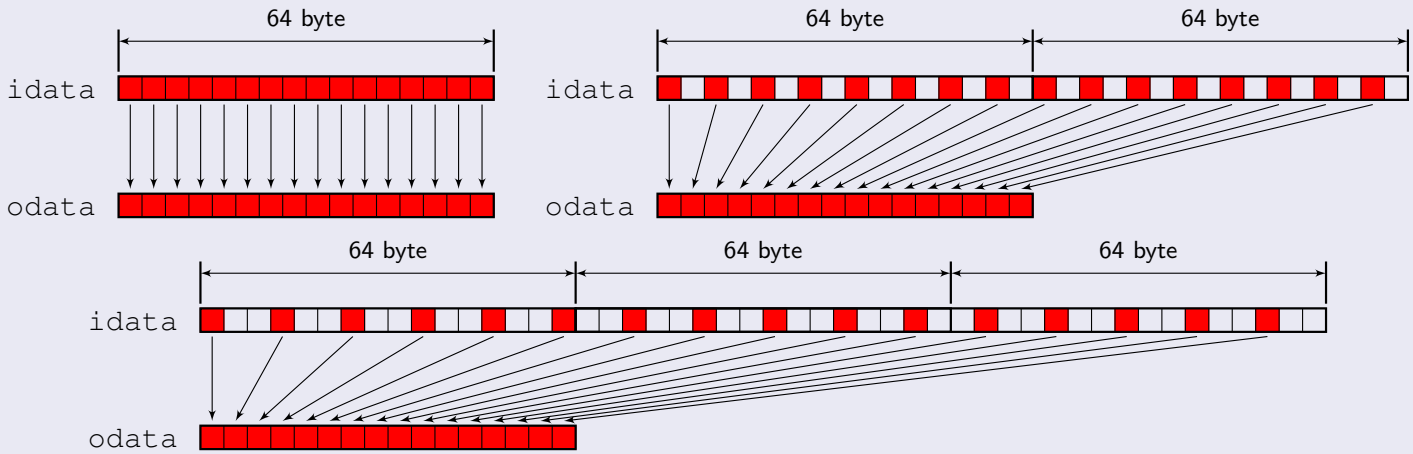


Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

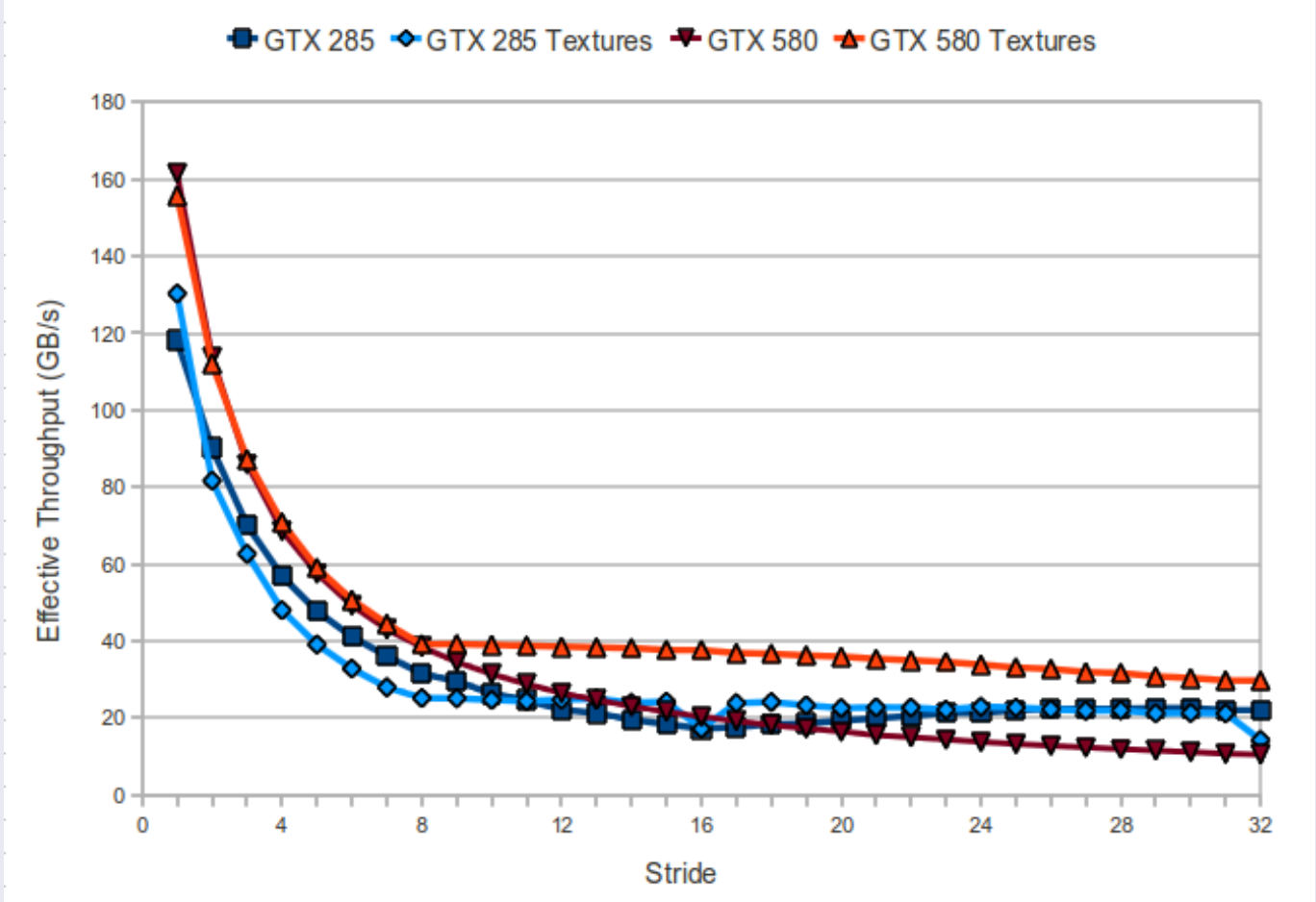
- ✓ Data wordt gelezen en geschreven uit het global memory via 32-, 64- of 128-byte operaties

Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

- ✓ Data wordt gelezen en geschreven uit het global memory via 32-, 64- of 128-byte operaties
- ✓ Lezen van data met een stride



Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA



Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

Andere problemen zijn: idle cores en overhead

Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

Andere problemen zijn: idle cores en overhead

- ✓ Omdat elk niveau er $4\times$ minder werk is, bereiken we een punt dat er minder werk is dan dat er cores zijn $\rightarrow$ idle cores

Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

Andere problemen zijn: idle cores en overhead

- ✓ Omdat elk niveau er $4\times$ minder werk is, bereiken we een punt dat er minder werk is dan dat er cores zijn $\rightarrow$ idle cores
- ✓ Bovendien wordt het lanceren van een nieuwe rekentaak en de communicatie tussen cores relatief steeds duurder (= overhead)

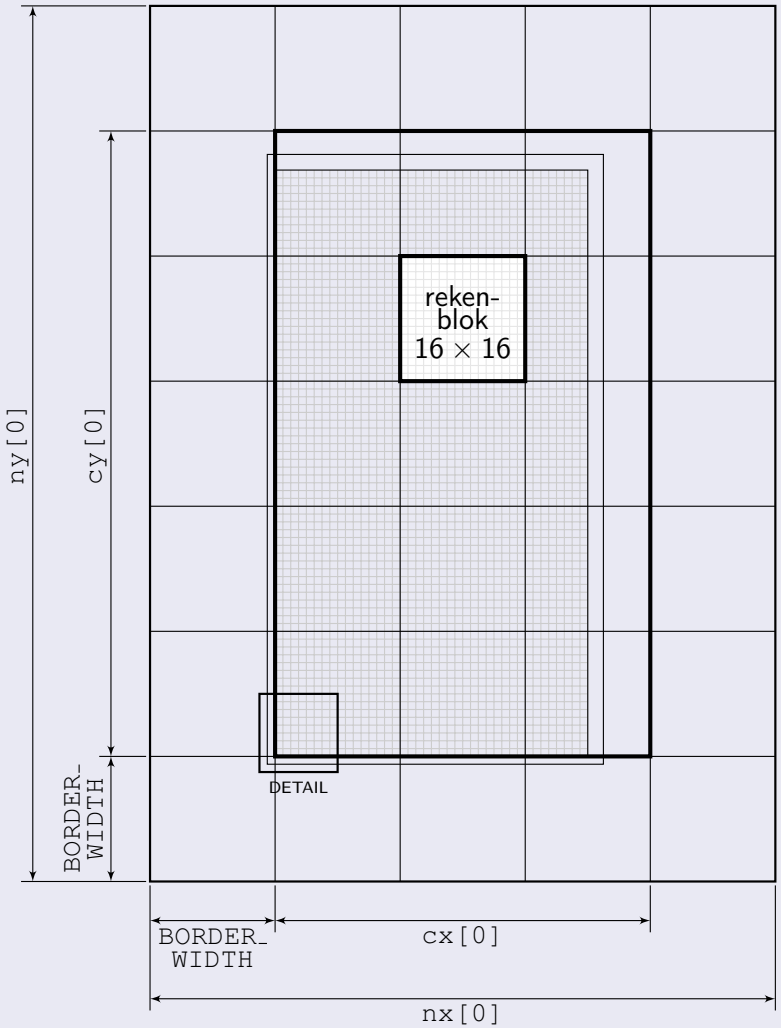
Voornaamste problemen voor implementatie van de RRB-solver op de GPU met CUDA

Andere problemen zijn: idle cores en overhead

- ✓ Omdat elk niveau er $4\times$ minder werk is, bereiken we een punt dat er minder werk is dan dat er cores zijn $\rightarrow$ idle cores
- ✓ Bovendien wordt het lanceren van een nieuwe rekentaak en de communicatie tussen cores relatief steeds duurder (= overhead)

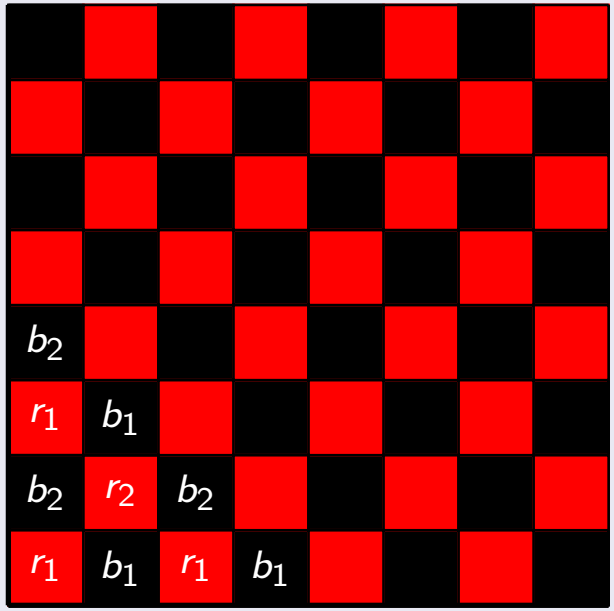
Op de grovere niveaus zullen we performance verliezen door idle cores en overhead

Data eerst inbedden in een optimaler grid

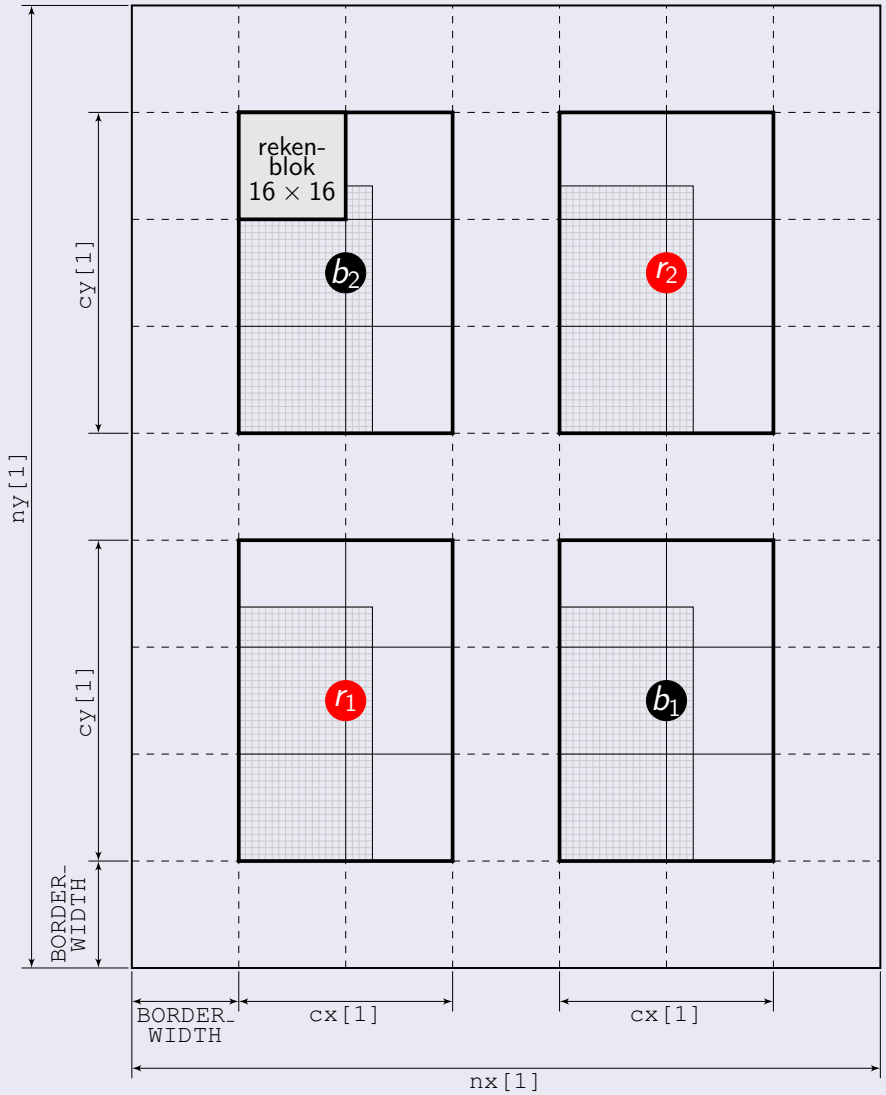


Slim opslagformaat: $r_1/r_2/b_1/b_2$

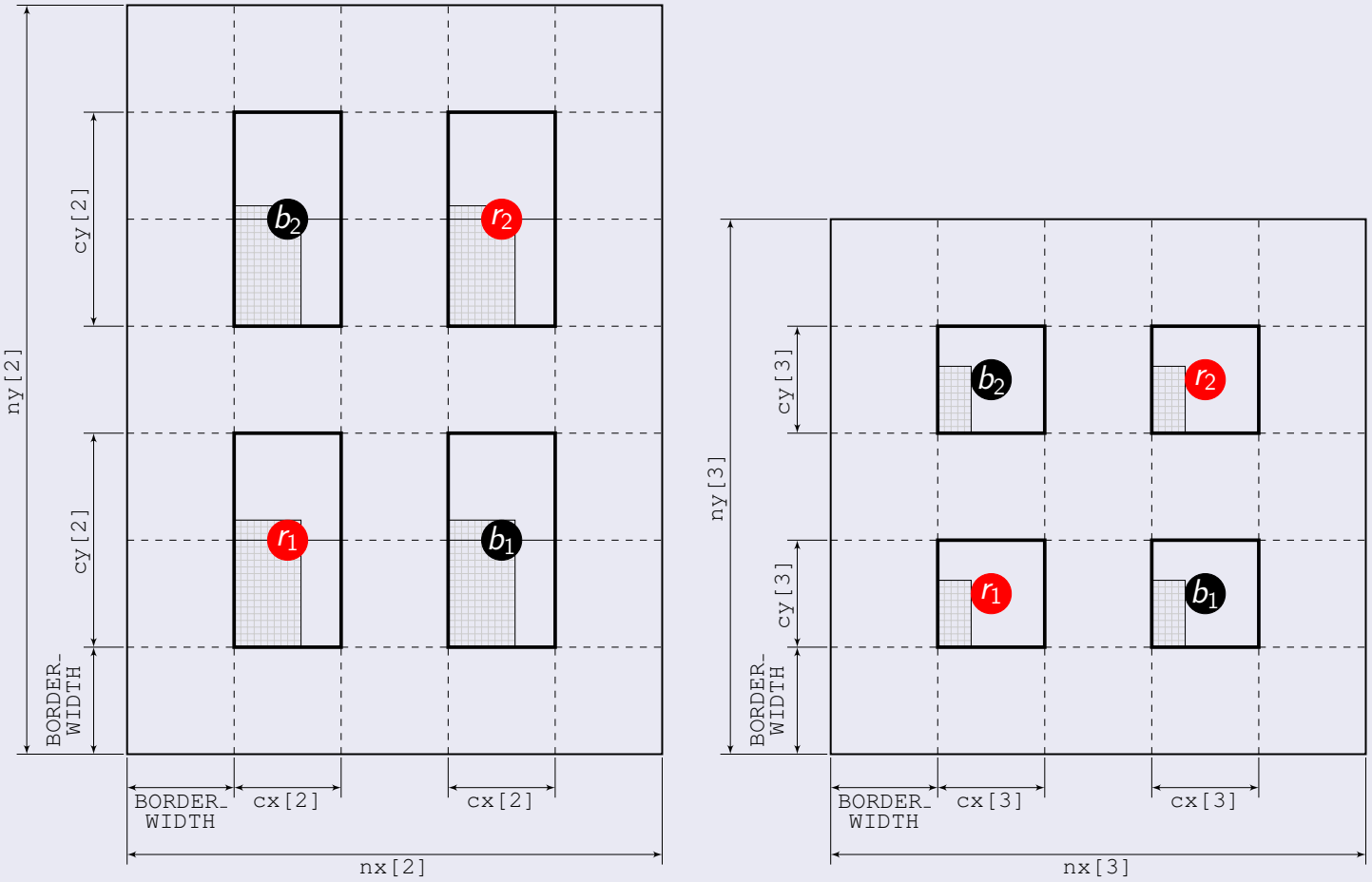
De gridpunten worden verdeeld in 4 groepen:



Slim opslagformaat: $r_1/r_2/b_1/b_2$



Slim opslagformaat: $r_1/r_2/b_1/b_2$



Slim opslagformaat: $r_1/r_2/b_1/b_2$

V: Waarom dit opslagformaat?

Slim opslagformaat: $r_1/r_2/b_1/b_2$

V: Waarom dit opslagformaat?

A: Veel meer lees- en schrijfbewerkingen kunnen “coalesced” worden uitgevoerd, waardoor we maximale throughput behalen

Slim opslagformaat: $r_1/r_2/b_1/b_2$

V: Waarom dit opslagformaat?

A: Veel meer lees- en schrijfbewerkingen kunnen “coalesced” worden uitgevoerd, waardoor we maximale throughput behalen

V: Dit kost toch extra tijd?

Slim opslagformaat: $r_1/r_2/b_1/b_2$

V: Waarom dit opslagformaat?

A: Veel meer lees- en schrijfbewerkingen kunnen "coalesced" worden uitgevoerd, waardoor we maximale throughput behalen

V: Dit kost toch extra tijd?

A: Ja, dat klopt, maar dit is echt minimaal (maar een paar %), min of meer eenmalig, en je krijgt er gigantisch veel extra throughput voor terug

Resultaten, conclusies, aanbevelingen

Testproblemen

- ✓ 2D Poisson (mathematisch)
- ✓ De Gelderse IJssel
- ✓ Plymouth Sound
- ✓ Port Presto

Testproblemen

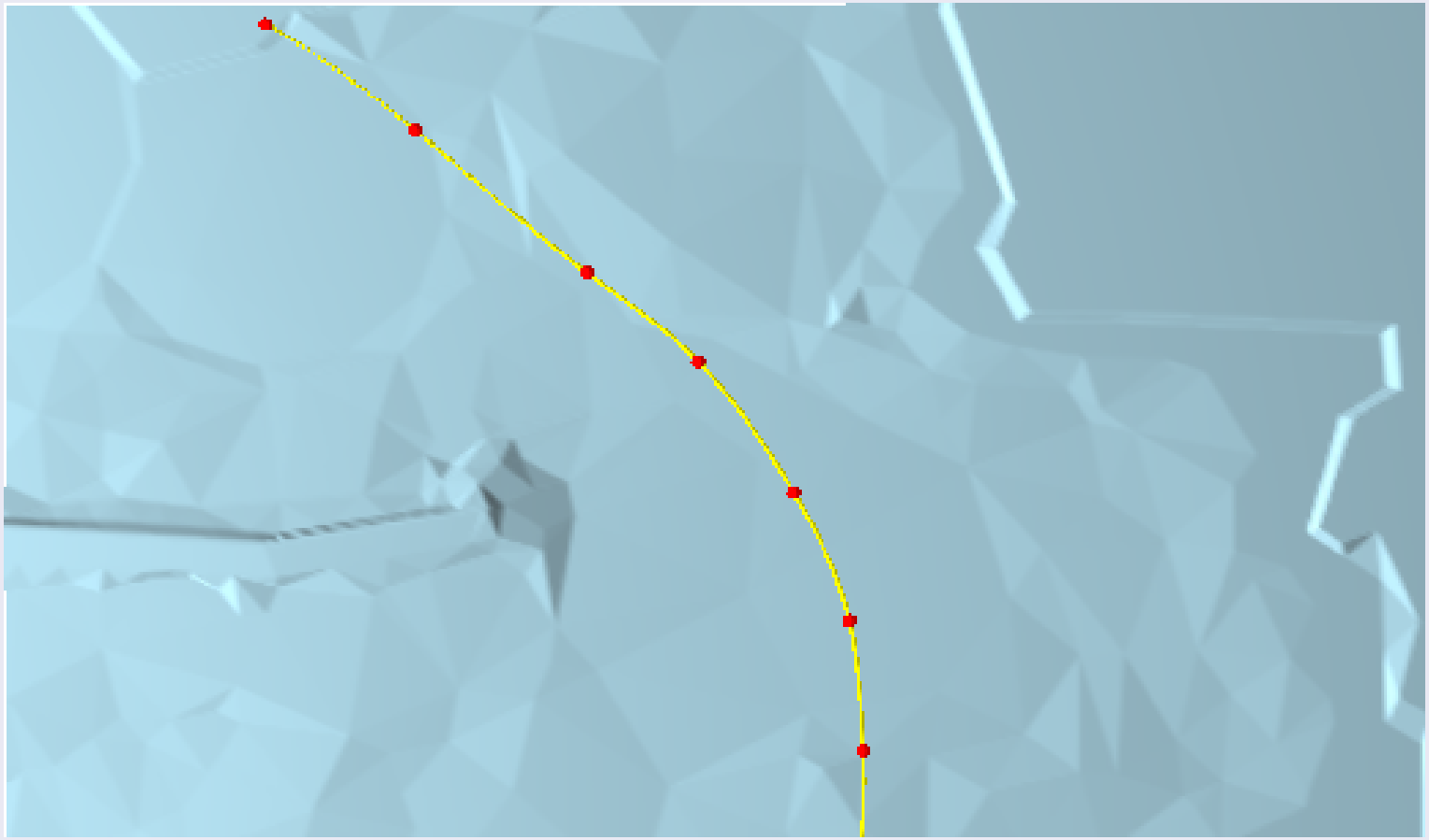
Diverse gebieden, diverse afmetingen

Voor Plymouth:

L_x [m]	L_y [m]	N_x	N_y	n (#nodes)
2,000	1,250	400	250	100,000
2,500	2,000	500	400	200,000
4,000	3,125	800	625	500,000
4,000	6,250	800	1,250	1,000,000
6,000	6,250	1,200	1,250	1,500,000

Testproblemen

Plymouth 100k:



Test machine 1

Merk / Type	Dell Precision Workstation T3400
Eigenaar / machine nr.	MARIN LIN0143
CPU	Intel Core 2 Duo E6850 @ 3.00 GHz
Aantal cores	2
Cache	64 kB L1 / 4 MB L2
Geheugen	4 GB (4 $\times$ 1 GB) RAM DDR2 @ 667 MHz
Moederbord	Dell Custom
Besturingssysteem	Ubuntu 10.04.3 LTS
Kernel	2.6.32-27-generic (x86_64)
CUDA versie	3.2
Driver versie	260.19.26
GCC versie	4.4.3
GPU (CUDA + scherm)	Asus NVIDIA GeForce GTX 285
Geheugen	1024 MB
Aantal cores	30 SM $\times$ 8 (cores/SM) = 240 cores
Compute capability	1.3

Test machine 2

Merk / Type	Dell Precision Workstation T3500
Eigenaar / machine nr.	MARIN LIN0169
CPU	Intel Xeon W3520 @ 2.67 GHz
Aantal cores	4
Cache	256 kB L1 / 1 MB L2 / 8 MB L3
Geheugen	6 GB (3 $\times$ 2 GB) RAM DDR2 @ 1066 MHz
Moederbord	Dell Custom
Besturingssysteem	Ubuntu 10.04.3 LTS
Kernel	2.6.32-34-generic (x86_64)
CUDA versie	4.0
Driver versie	270.41.19
GCC versie	4.4.3
GPU 0 (CUDA)	Asus NVIDIA GeForce GTX 580
Geheugen	1536 MB
Aantal cores	16 SM $\times$ 32 (cores/SM) = 512 cores
Compute capability	2.0
GPU 1 (scherm)	Asus NVIDIA Quadro NVS 295
Geheugen	256 MB
Aantal cores	1 SM $\times$ 8 (cores/SM) = 8 cores
Compute capability	1.1

Hoe en wat hebben we getest

Hoe en wat hebben we getest

- ✓ Het 2D Poisson probleem is gebruikt voor throughput analyse van de CUDA RRB-solver

Hoe en wat hebben we getest

- ✓ Het 2D Poisson probleem is gebruikt voor throughput analyse van de CUDA RRB-solver
- ✓ De `lin_wacu` software is gebruikt voor tijdmetingen, aantal CG-iteraties, etc.

Hoe en wat hebben we getest

- ✓ Het 2D Poisson probleem is gebruikt voor throughput analyse van de CUDA RRB-solver
- ✓ De lin_wacu software is gebruikt voor tijdmetingen, aantal CG-iteraties, etc.
  - Solver tijd (alleen $S\vec{\psi} = \mathbf{b}$)
  - Totale tijd ($S\vec{\psi} = \mathbf{b}$, tijdsintegratie, inkomende golven, etc.)

Hoe en wat hebben we getest

- ✓ Het 2D Poisson probleem is gebruikt voor throughput analyse van de CUDA RRB-solver
- ✓ De lin_wacu software is gebruikt voor tijdmetingen, aantal CG-iteraties, etc.
  - Solver tijd (alleen $S\vec{\psi} = \mathbf{b}$)
  - Totale tijd ($S\vec{\psi} = \mathbf{b}$, tijdsintegratie, inkomende golven, etc.)
- ✓ C++ RRB-solver op 1 core van CPU

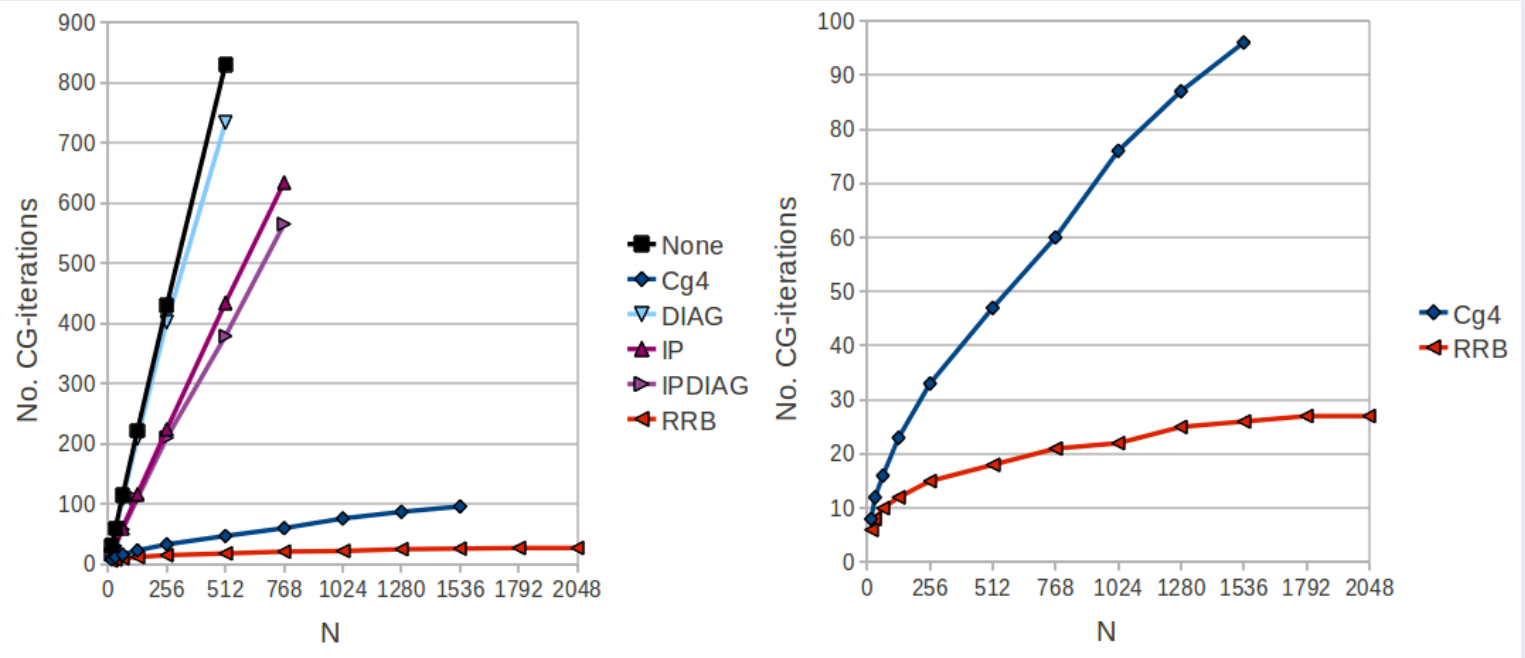
Hoe en wat hebben we getest

- ✓ Het 2D Poisson probleem is gebruikt voor throughput analyse van de CUDA RRB-solver
- ✓ De lin_wacu software is gebruikt voor tijdmetingen, aantal CG-iteraties, etc.
  - Solver tijd (alleen $S\vec{\psi} = \mathbf{b}$)
  - Totale tijd ($S\vec{\psi} = \mathbf{b}$, tijdsintegratie, inkomende golven, etc.)
- ✓ C++ RRB-solver op 1 core van CPU
- ✓ CUDA RRB-solver op alle cores van GPU

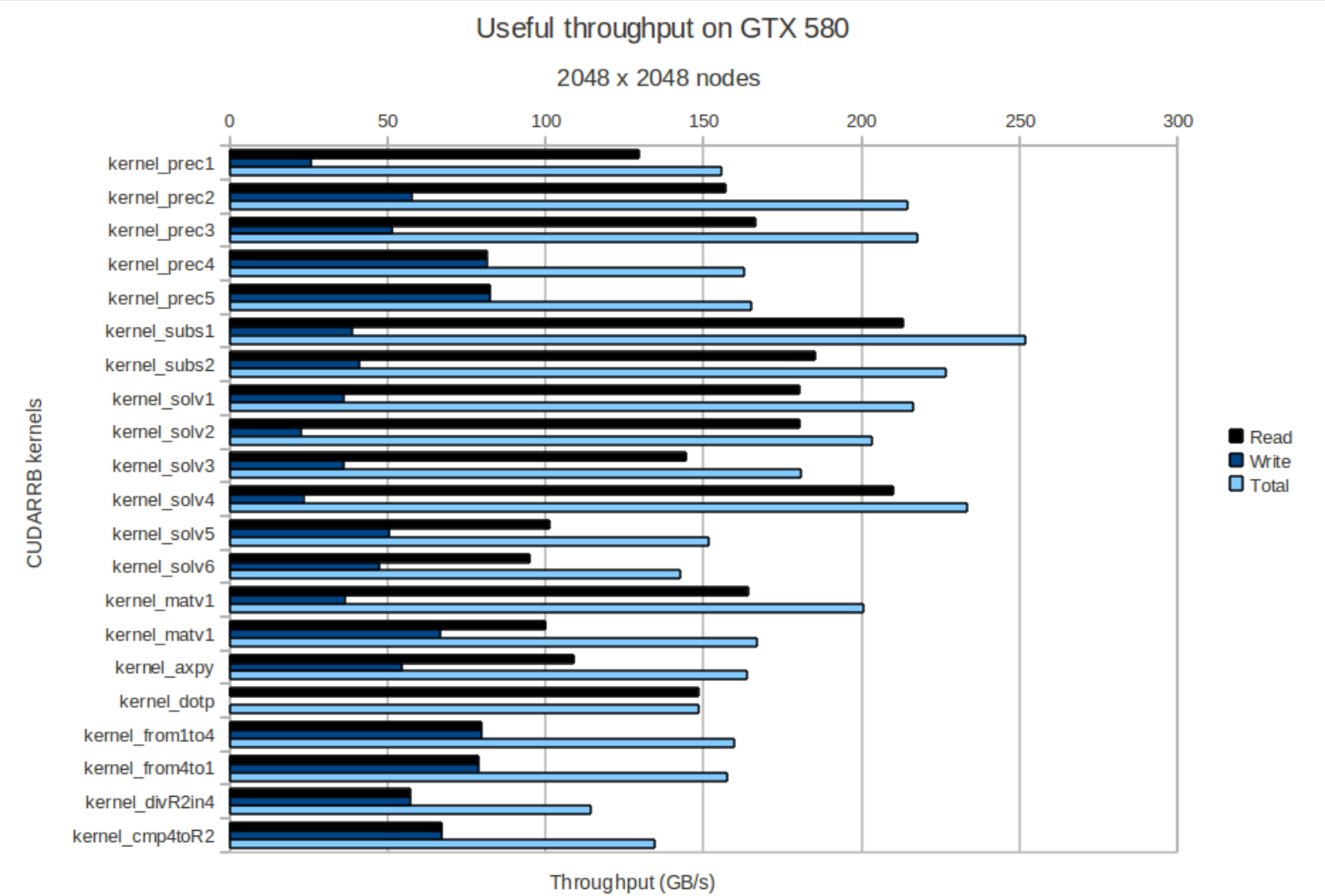
Hoe en wat hebben we getest

- ✓ Het 2D Poisson probleem is gebruikt voor throughput analyse van de CUDA RRB-solver
- ✓ De lin_wacu software is gebruikt voor tijdmetingen, aantal CG-iteraties, etc.
  - Solver tijd (alleen $S\vec{\psi} = \mathbf{b}$)
  - Totale tijd ($S\vec{\psi} = \mathbf{b}$, tijdsintegratie, inkomende golven, etc.)
- ✓ C++ RRB-solver op 1 core van CPU
- ✓ CUDA RRB-solver op alle cores van GPU
- ✓ Extra solver: CUDA IPDIAG-solver op alle cores van GPU

Schaling

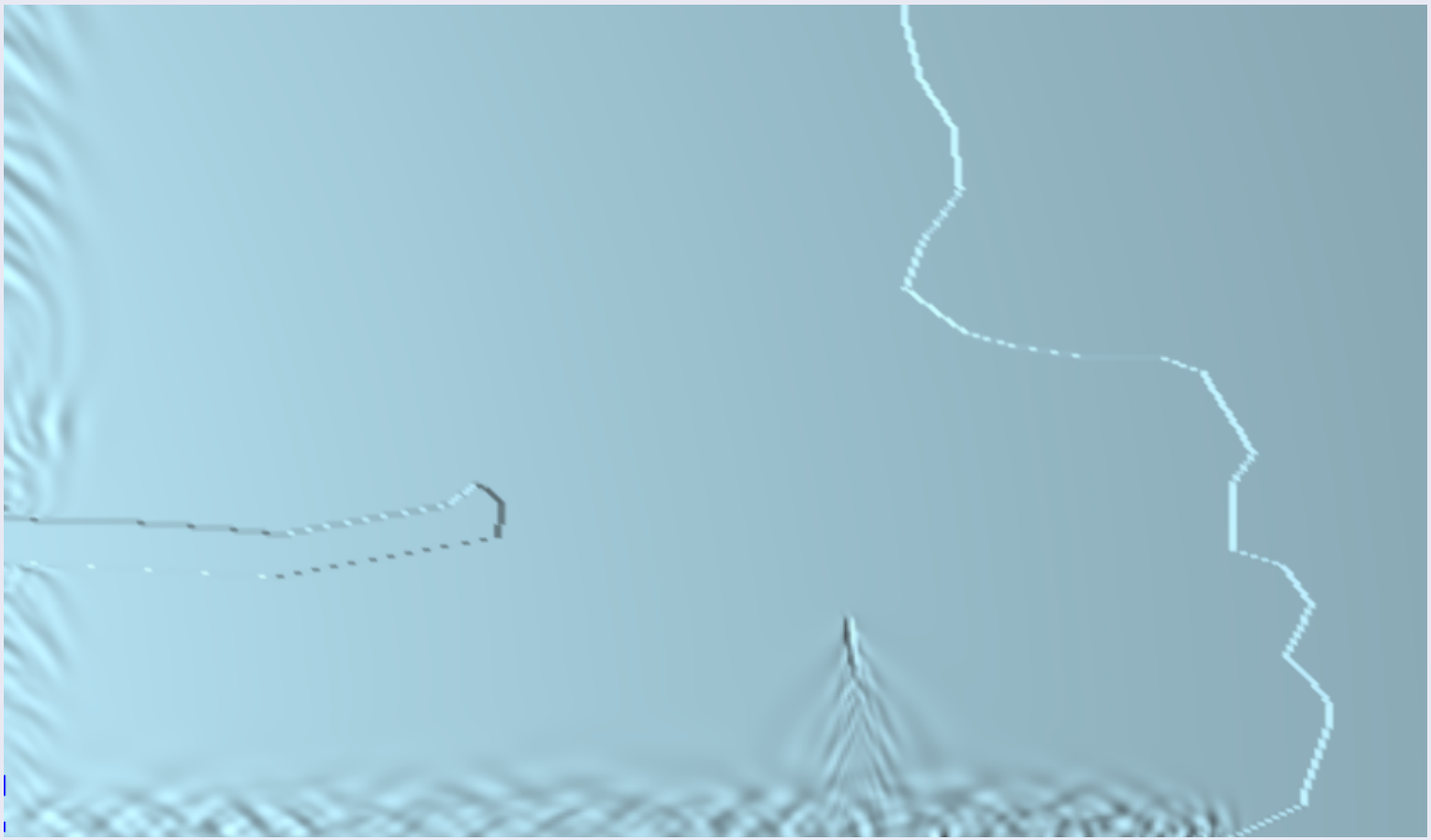

No. CG-iterations
N
None
Cg4
DIAG
IP
IPDIAG
RRB
No. CG-iterations
N
Cg4
RRB

Throughput van de CUDA RRB-solver



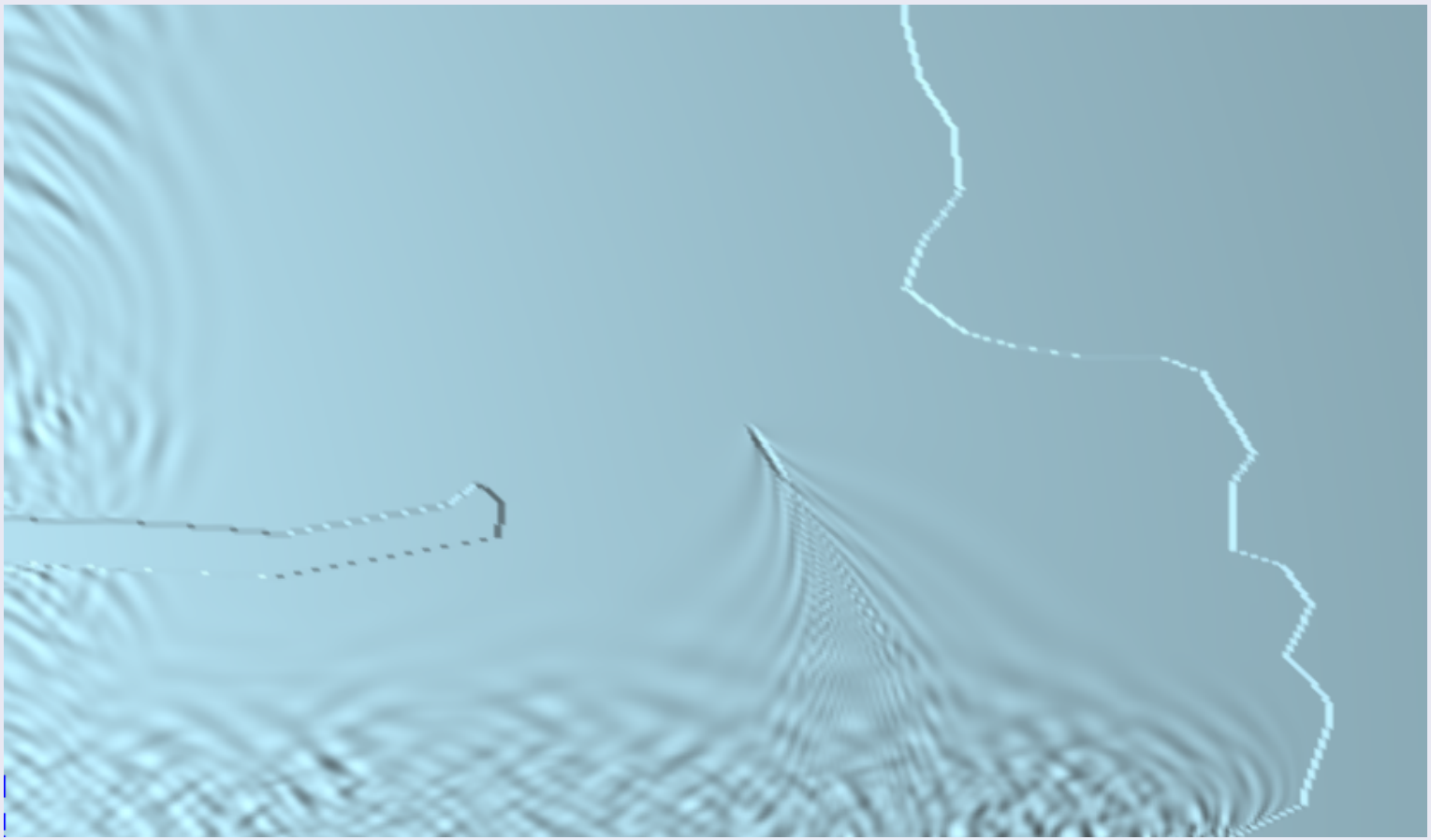
Simulatie Plymouth 100k: tijdframe 500

Plymouth 100k:



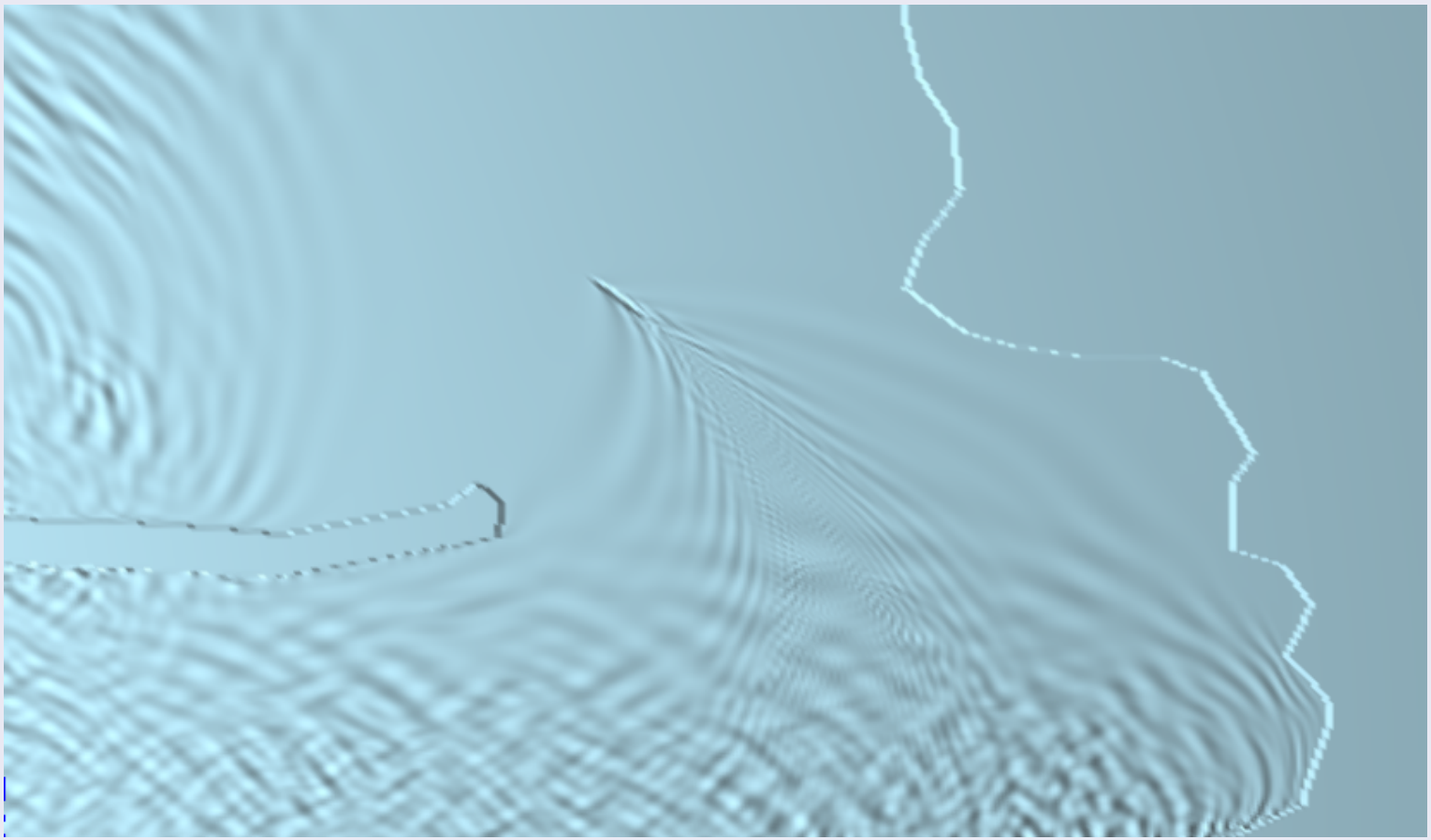
Simulatie Plymouth 100k: tijdrame 1000

Plymouth 100k:



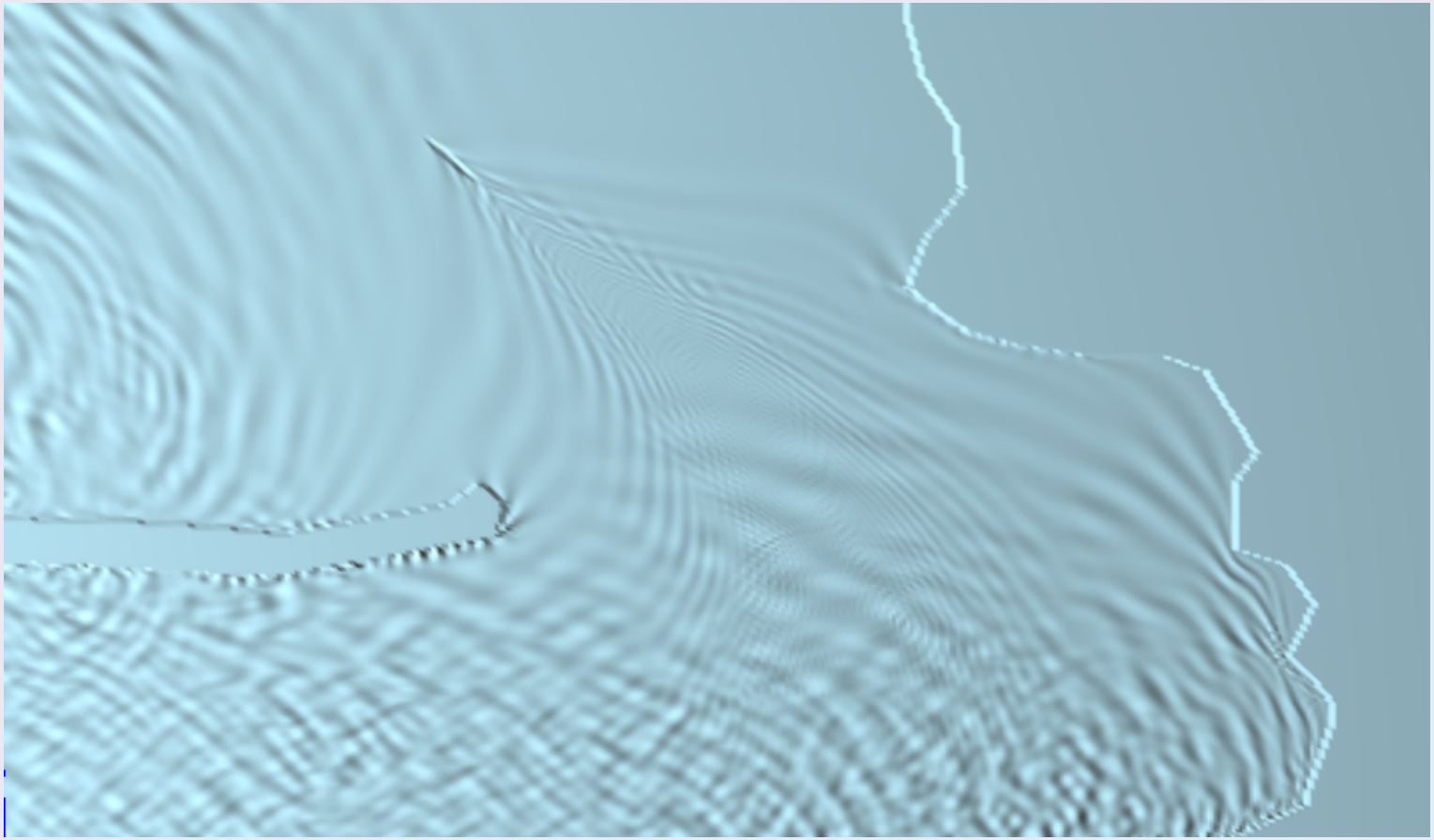
Simulatie Plymouth 100k: tijdrame 1500

Plymouth 100k:



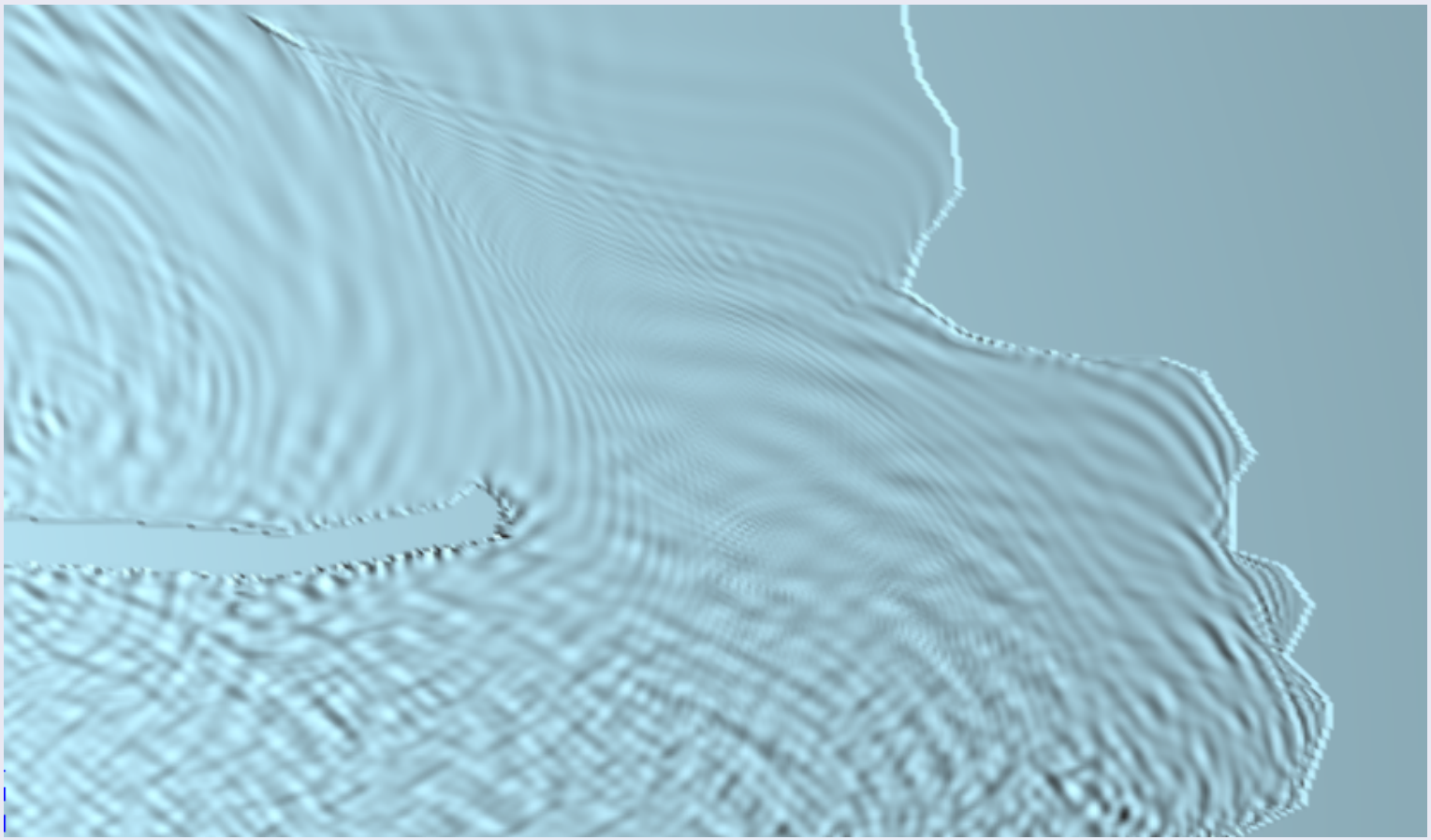
Simulatie Plymouth 100k: tijdframe 2000

Plymouth 100k:



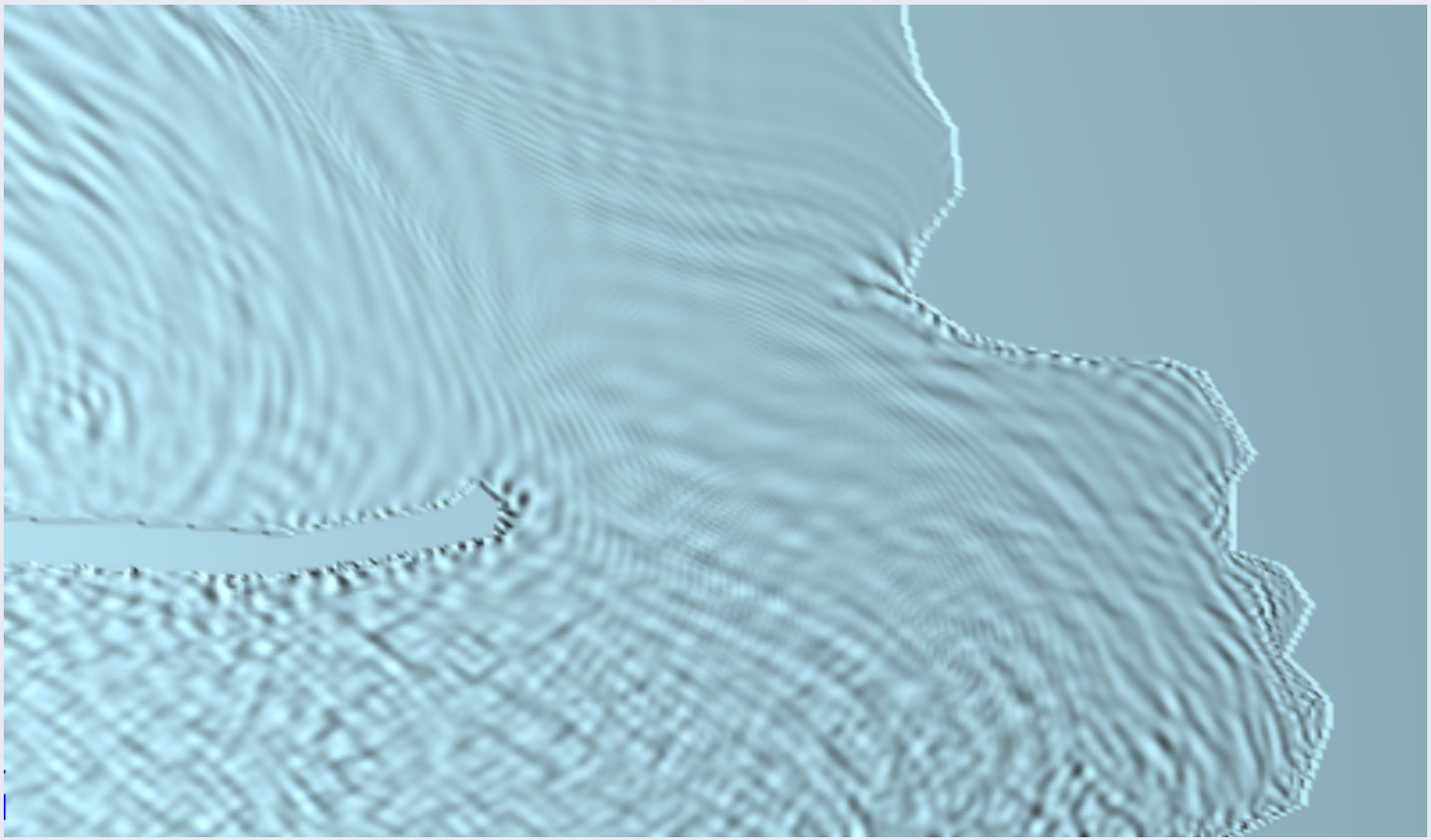
Simulatie Plymouth 100k: tijdframe 2500

Plymouth 100k:

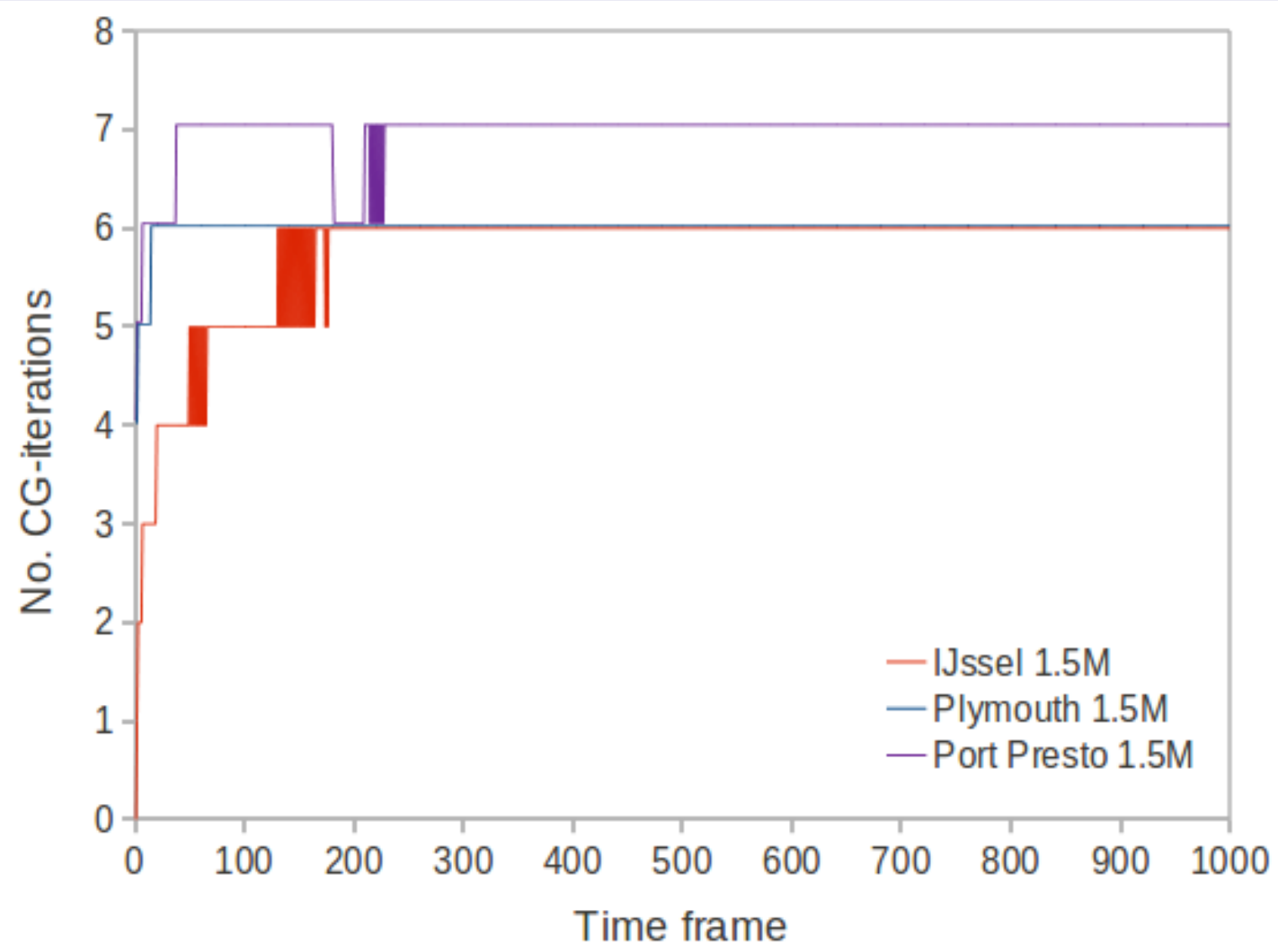


Simulatie Plymouth 100k: tijdframe 3000

Plymouth 100k:



Aantal CG-iteraties RRB-solver



Solver rekentijd Plymouth 1.5M

	Systeem 1: GTX 285			Systeem 2: GTX 580		
	C++ RRB	CUDA RRB	CUDA IPDIAG	C++ RRB	CUDA RRB	CUDA IPDIAG
100k	15.988	4.234	6.298	11.450	1.894	2.110
200k	41.022	5.472	7.850	57.661	2.628	3.072
500k	141.007	8.704	18.372	71.493	4.728	8.941
1M	332.706	13.992	30.463	178.297	7.865	16.915
1.5M	490.332	18.523	40.824	298.446	10.618	23.159

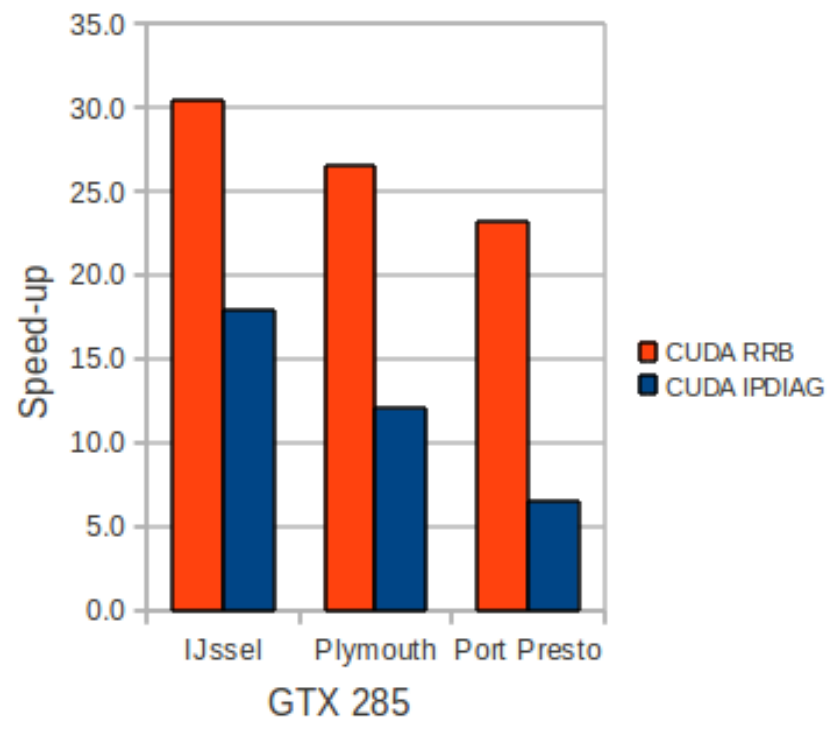
Solver rekentijd Plymouth 1.5M

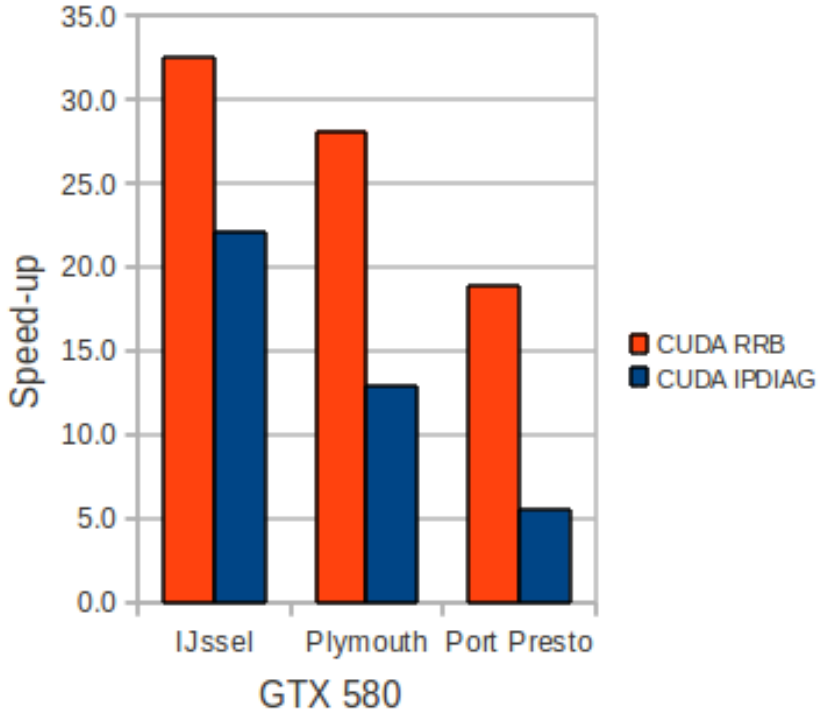
	Systeem 1: GTX 285			Systeem 2: GTX 580		
	C++ RRB	CUDA RRB	CUDA IPDIAG	C++ RRB	CUDA RRB	CUDA IPDIAG
100k	15.988	4.234	6.298	11.450	1.894	2.110
200k	41.022	5.472	7.850	57.661	2.628	3.072
500k	141.007	8.704	18.372	71.493	4.728	8.941
1M	332.706	13.992	30.463	178.297	7.865	16.915
1.5M	490.332	18.523	40.824	298.446	10.618	23.159

Totale rekentijd Plymouth 1.5M

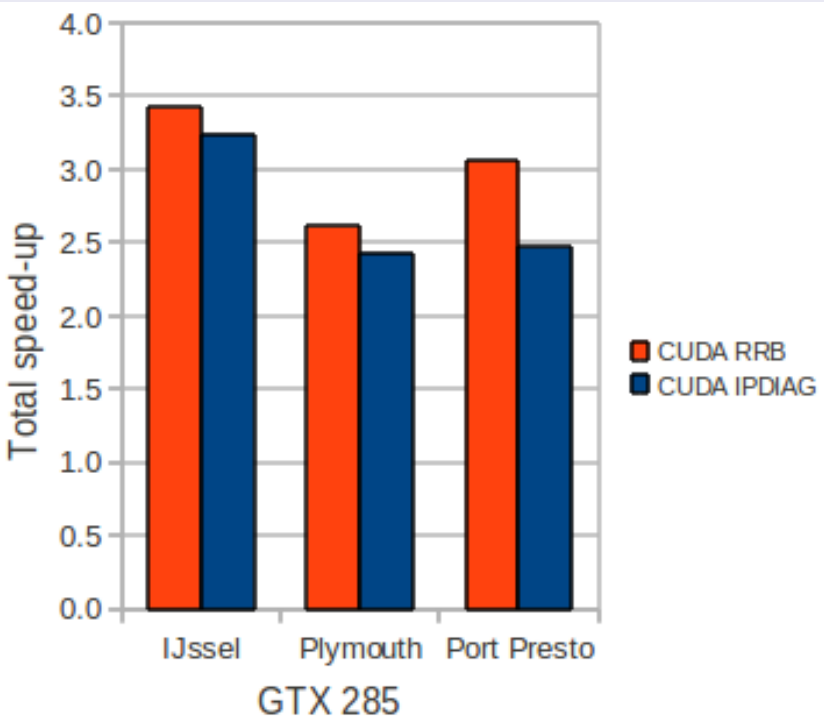
	Systeem 1: GTX 285			Systeem 2: GTX 580		
	C++ RRB	CUDA RRB	CUDA IPDIAG	C++ RRB	CUDA RRB	CUDA IPDIAG
100k	41.09	29.33	31.40	32.90	23.34	23.56
200k	83.64	48.09	50.47	92.49	37.46	37.90
500k	206.09	73.78	83.45	141.67	74.91	79.12
1M	509.41	190.69	207.16	313.18	142.75	151.80
1.5M	764.91	293.10	315.40	506.95	219.12	231.66

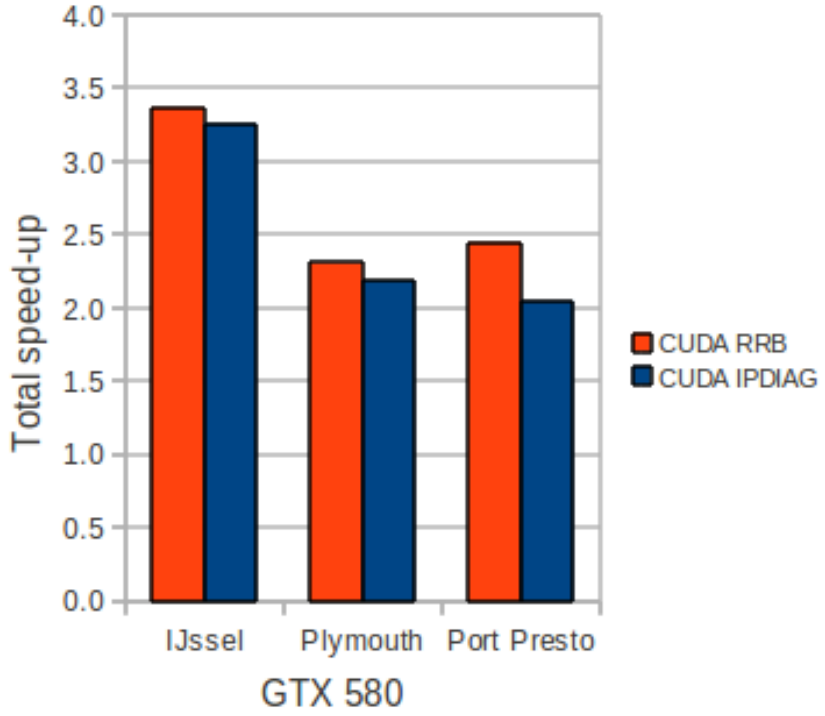
Solver speed up





Totale speed up

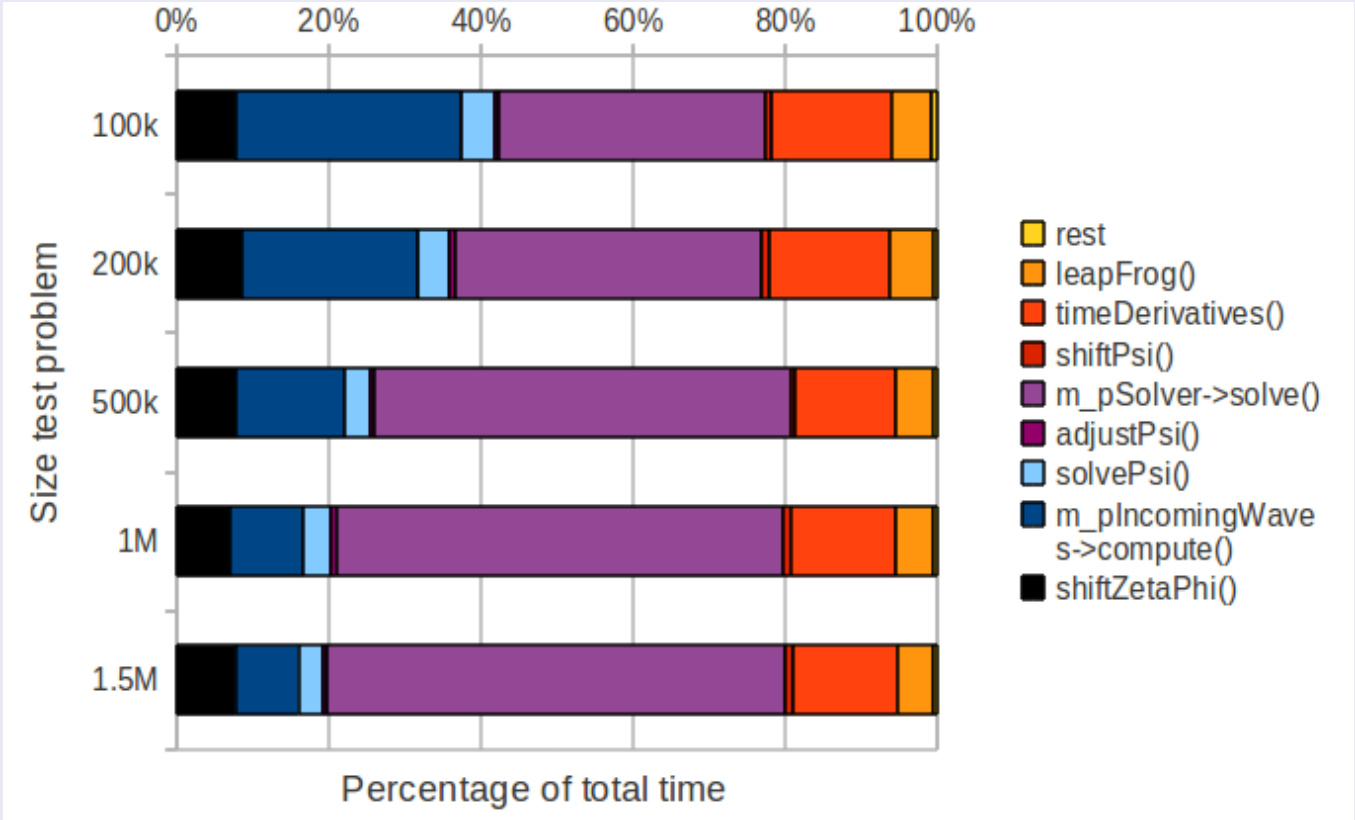




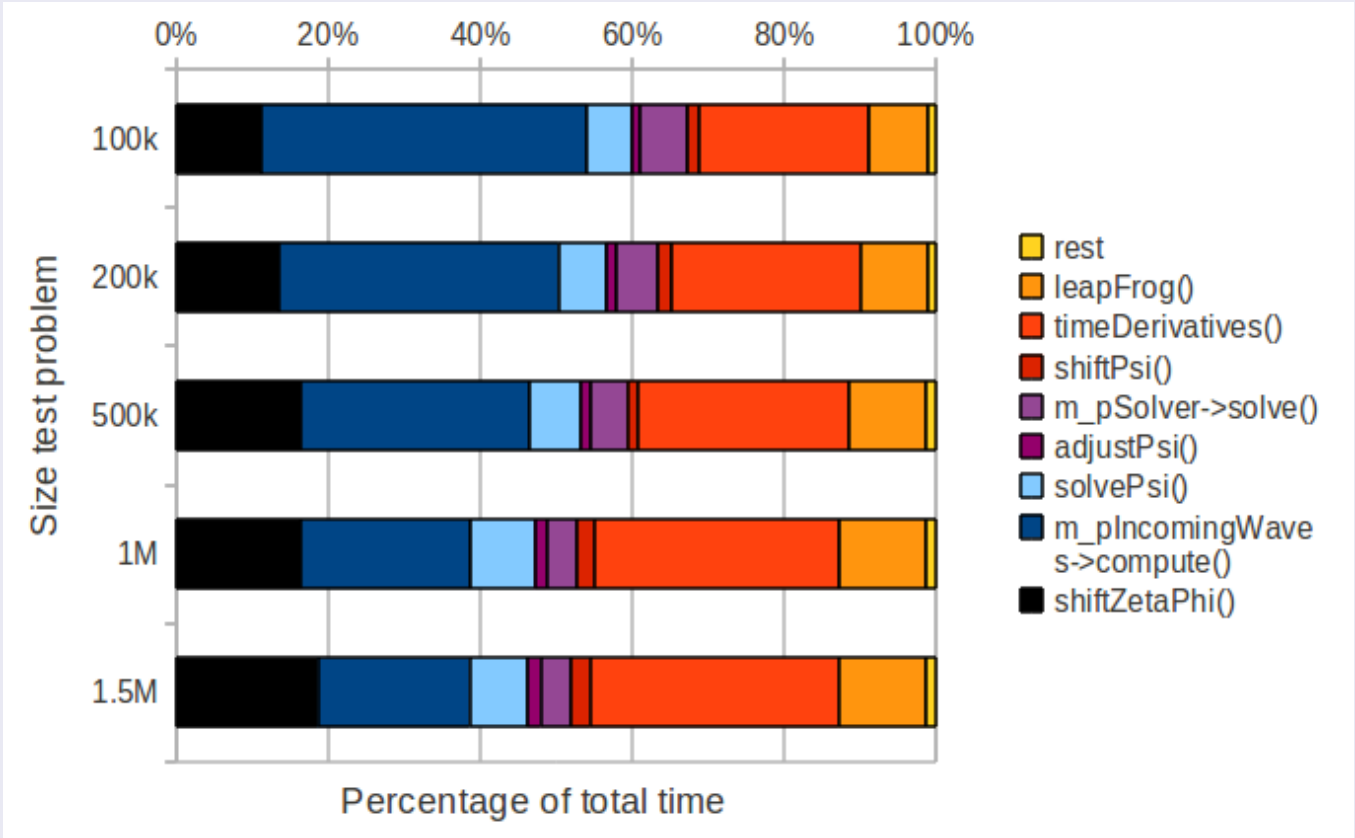
Profiel van de `lin_wacu` code

		100k	200k	500k	1M	1.5M
`shiftZetaPhi()`		3.2	6.4	16.1	32.0	48.8
`m_pIncomingWaves->compute()`		12.2	17.3	29.4	43.6	52.4
`waveBreaking()`		0	0	0	0	0
`boundaryZetaPhi()`		0	0	0	0.1	0.1
`solvePsi()`		1.7	3.0	6.6	16.4	20.1
`adjustPsi()`		0.3	0.6	1.3	3.2	4.3
`m_pSolver->solve()`	CUDA	1.8	2.6	4.7	7.8	10.6
	(C++)	(14.3)	(29.9)	(112.3)	(266.3)	(379.5)
`shiftPsi()`		0.4	0.8	1.3	4.3	6.5
`boundaryPsi()`		0	0	0	0	0
`timeDerivatives()`		6.4	11.7	27.2	63.0	85.8
`weaklyReflective()`		0.1	0.2	0.2	0.4	0.4
`smoothZetaPhi()`		0	0	0	0	0
`leapFrog()`		2.2	4.2	9.9	22.2	30.1
`m_pLPFil->filter(m zeta)`		0	0	0	0	0
`m_pLpFil->filter(m phi)`		0	0	0	0	0
"maxwaves"		0.2	0.3	1.1	2.0	2.9
Totaal	CUDA	28.5	47.1	97.8	195.0	262.0
	(C++)	(41.0)	(74.4)	(205.4)	(453.5)	(630.9)

Voorheen met de C++ RRB-solver



Nu met de CUDA RRB-solver



Conclusies

Conclusies

✓ De CUDA RRB-solver is 30× sneller dan de C++ versie

(hoe groter het probleem des te groter de speed up)

Conclusies

- ✓ De CUDA RRB-solver is 30× sneller dan de C++ versie

  (hoe groter het probleem des te groter de speed up)
- ✓ De solver is niet langer de bottle-neck in de `lin_wacu` code

Conclusies

- ✓ De `CUDA` RRB-solver is 30× sneller dan de `C++` versie

  (hoe groter het probleem des te groter de speed up)
- ✓ De solver is niet langer de bottle-neck in de `lin_wacu` code
- ✓ De implementatie van de `CUDA` RRB-solver is erg efficiënt

  (throughput tot wel 250 GB/s, veel hoger kan niet)

Conclusies

- ✓ De CUDA RRB-solver is 30× sneller dan de C++ versie

  (hoe groter het probleem des te groter de speed up)
- ✓ De solver is niet langer de bottle-neck in de `lin_wacu` code
- ✓ De implementatie van de CUDA RRB-solver is erg efficiënt

  (throughput tot wel 250 GB/s, veel hoger kan niet)

Aanbevelingen

Conclusies

- ✓ De CUDA RRB-solver is 30× sneller dan de C++ versie

  (hoe groter het probleem des te groter de speed up)
- ✓ De solver is niet langer de bottle-neck in de `lin_wacu` code
- ✓ De implementatie van de CUDA RRB-solver is erg efficiënt

  (throughput tot wel 250 GB/s, veel hoger kan niet)

Aanbevelingen

- ✓ Optimaliseer de rest van de `lin_wacu` code

  (dan krijg je wellicht echt 30× snellere software)

Conclusies

- ✓ De `CUDA` RRB-solver is 30× sneller dan de `C++` versie

  (hoe groter het probleem des te groter de speed up)
- ✓ De solver is niet langer de bottle-neck in de `lin_wacu` code
- ✓ De implementatie van de `CUDA` RRB-solver is erg efficiënt

  (throughput tot wel 250 GB/s, veel hoger kan niet)

Aanbevelingen

- ✓ Optimaliseer de rest van de `lin_wacu` code

  (dan krijg je wellicht echt 30× snellere software)
- ✓ Onderzoek hoe nog grotere problemen kunnen worden opgelost

  (speciale Poisson solver, niet-equidistant grid, multiple CPU/GPU)

Zijn er vragen?

Voeg hier een filmpje in