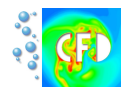
Hardwareorientierte Numerik für FEM-Frameworks

Markus Geveler

Institut für Angewandte Mathematik
TU Dortmund, Germany
markus.geveler@math.tu-dortmund.de

G2CG
Kaiserslautern, 18. April 2012

technische universität dortmund
fakultät für mathematik



Einleitung

Zwei ausgewählte Schlüsselaspekte des Wissenschaftlichen Rechnens

- *Alle Ebenen von Parallelität der (oft heterogenen) Hardware müssen berücksichtigt werden*
  - Vektorisierung auf einem core (SIMD)
  - multi-core / many-core (CPUs + GPUs)
  - verteilter Speicher
  - ILP, ...
  - heterogene Ressourcen (auf Knoten- und Chip-Ebene)
  - $\rightarrow$ Hardwareeffizienz
- *Alle Ebenen des numerischen Lösungsverfahrens müssen berücksichtigt werden*
  - Diskretisierung in Ort und Zeit
  - Stabilisierung, Linearisierung nicht-linearer Probleme
  - *Lösung der linearen Systeme*
  - *Komponenten dieser Löser*
  - $\rightarrow$ Numerische Effizienz

Motivation

Hardware- und Numerische Effizienz beeinflussen sich gegenseitig

- Beispiel: Vorkonditionierung in linearen Lösern
  - gute Vorkonditionierung $\rightarrow$ bessere Konvergenzrate
  - oft: gute Vorkonditionierung $\rightarrow$ höhere algorithmische Komplexität, schlechtere Parallelisierbarkeit

$\rightarrow$ simultane Berücksichtigung von Hardware- und Numerischer Effizienz unbedingt erforderlich!

Motivation

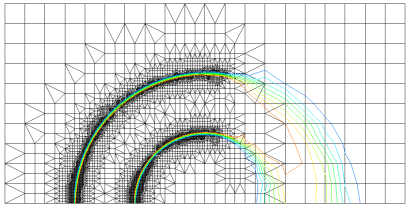
Heute am Beispiel:

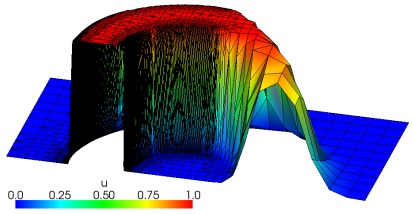
- geometrische Mehrgitterlöser zugeschnitten für FEM
- hardwareorientiert: GPUs vs. multi-core CPUs
- numerisch stark: Sparse Approximate Inverse -basierte Glätter
- flexibel: unstrukturierte Gitter

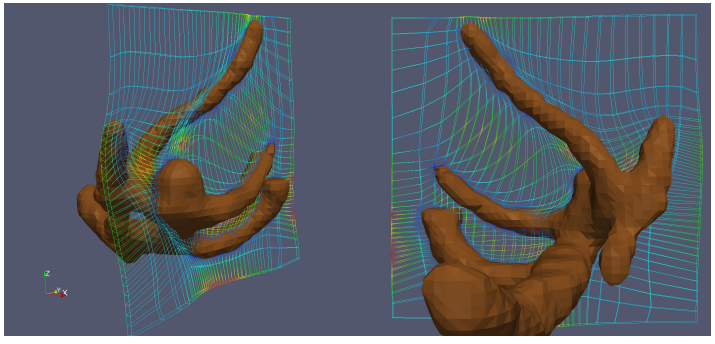
Motivation

FEM

- hochgradig genaue und flexible Methoden für PDEs:
  - Elemente hoher Ordnung (auch nicht-konform)
  - *beliebig unstrukturierte Gitter*
  - Gitteradaptivität
  - Pressure-Schur-Complement Vorkonditionierung
  - ...
- in Verbindung mit geometrischen Mehrgittermethoden:
  - Konvergenzraten unabhängig von der Gitterschrittweite
  - superlineare Konvergenzeffekte möglich ($\rightarrow$ Elemente hoher Ordnung)





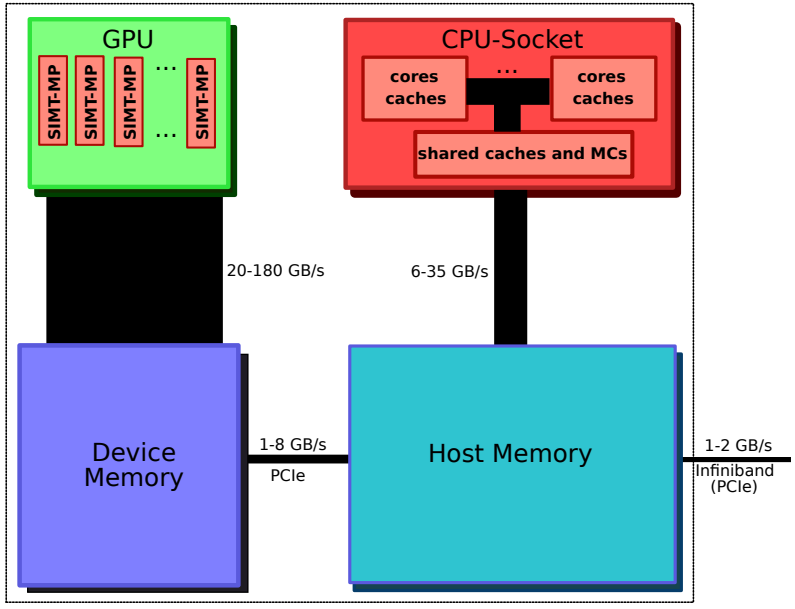


$\rightarrow$ **Finite Element (geometric) Multigrid (FE-gMG) Framework**

Motivation

GPUs

- Parallelisierungstechniken für FEM beschränken sich oft auf das Auswechseln von Basiskomponenten
- komplette Frameworks / komplexe Löser wenig thematisiert



- in FE-gMG:
  - strukturierte Gitter + zugehörige Operatoren für GPUs gut verstanden ($\rightarrow$ Dominik Göddeke, 1. FEAST Generation)
  - unstrukturierte Gitter + starke Glätter + GPUs / multi-core wenig thematisiert

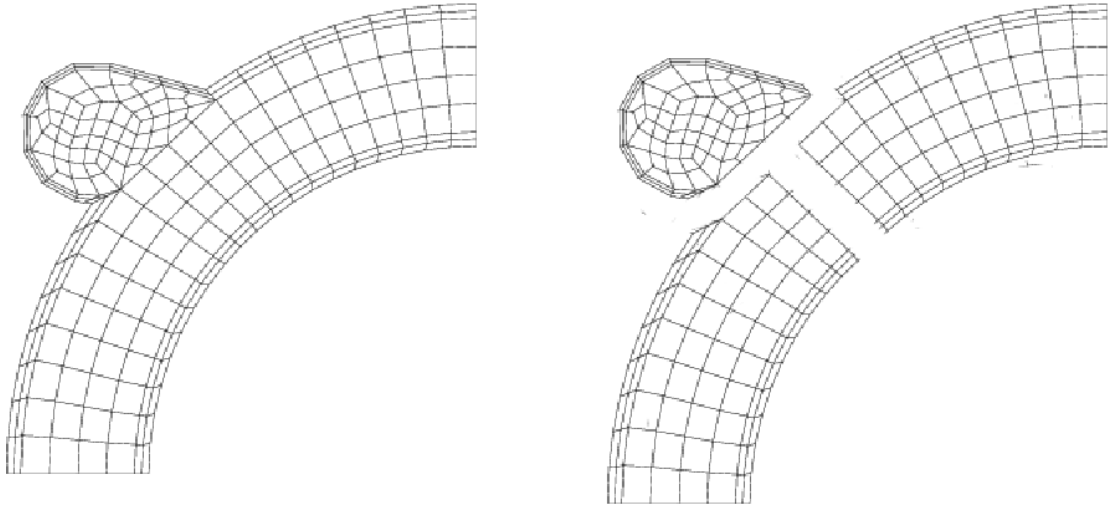
Motivation

FE-gMG - Performance-Aspekte

- FE-Raum höherer Ordnung $\rightarrow$ höhere numerische Qualität $\rightarrow$ höhere algor. Komplexität
- starke Glätter $\rightarrow$ höhere numerische Effizienz $\rightarrow$ höhere algor. Komplexität
- Abhängigkeit von Gitter, DOF-Nummerierung
- Abhängigkeit von Speicherformat, hardwarespezifischer Implementierung
- andere, z.B.: Zykluskontrolle

Einordnung von FE-gMG

Parallele lineare Löser für FEM



- starte mit konformem Grobgitter
- Zusammenfassen von Grobgitterzellen zu Matrixpatches für die Assemblierung lokaler Matrizen (strukturiert und unstrukturiert)
- Matrixpatches werden durch den Lastverteiler auf MPI-Prozesse verteilt (zunächst statisch, dann auf der Grundlage gesammelter Statistik-Daten)

Scalable Recursive Clustering

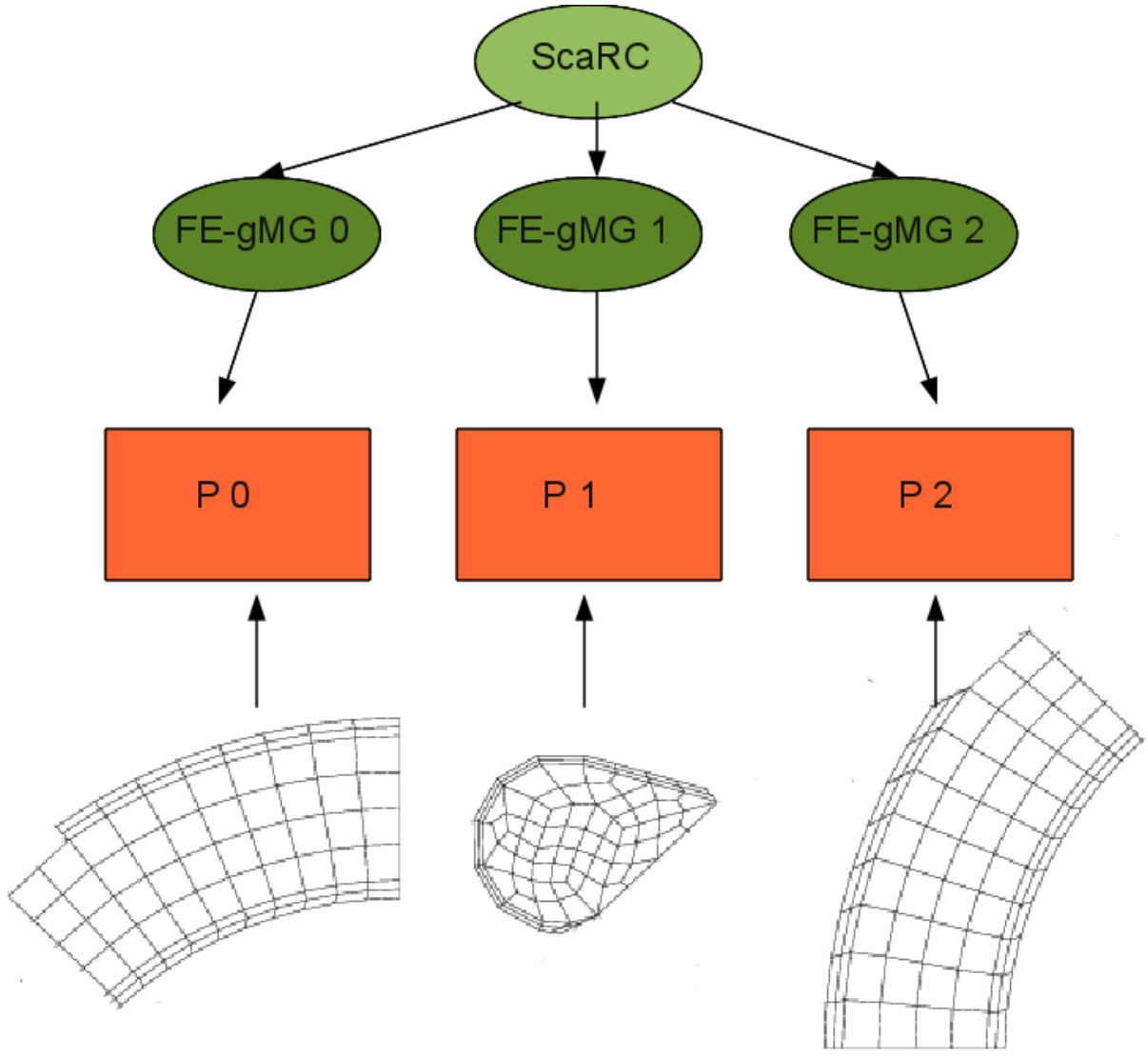
Einordnung von FE-gMG: ScaRC Löser

- auf globalem Problem wird ein datenparalleles Lösungsverfahren definiert
- als Glätter für den globalen Löser: Rekursion oder *blockweise lokale Löser* $\rightarrow$ *FE-gMG*
- Typ des Matrixpatches entscheidet über lokale Löserkomponenten innerhalb der Rekursion
- Anwendung des ScaRC-Glätters: globaler Defekt $\rightarrow$ lokale Löser (Rekursion oder FE-gMG) $\rightarrow$ globale Korrektur

Ziel: numerisch skalierbarer Löser

Scalable Recursive Clustering

Einordnung von FE-gMG: ScaRC Löser



FE-gMG

Idee: Ein performance-kritischer Kernel für gMG: SpMV

- Grobgitter Löser: Vorkonditionierte Krylov-Unterraum-Methoden
- Glätter: je nach Notwendigkeit basierend auf
  - vorkonditioniertem Richardson-Verfahren oder
  - Krylov-Unterraum-Methoden
- Defektberechnung

Was übrig bleibt...

- ein wenig BLAS-1 (dot-product, norm, scale, ...)
- wichtige Idee: *Gittertransfer* $\rightarrow$ kann auch auf SpMV zurückgeführt werden

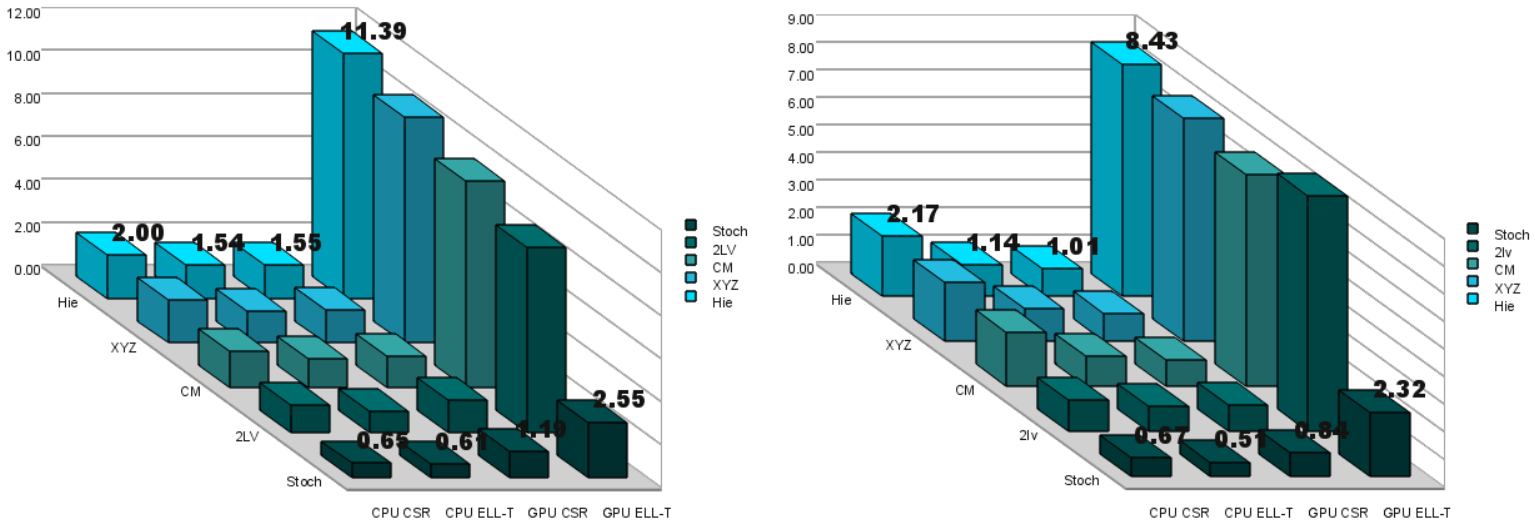
Vorteile

- Flexibilität der Implementierung (ausgewechselt werden nur *Matrizen*) $\rightarrow$ blackbox
- unabhängig von FE-Raum, Dimension
- performance-tuning an einer zentralen Stelle

SpMV: CSR vs. ELLPACK-R

Ausgangspunkt: SpMV kernel - Performance

- hier: Steifigkeitsmatrizen aus einem 3D-Poisson Problem, links Q_1, rechts Q_2 (mehr Nichtnulleinträge)
- Freiheitsgrad-Nummerierungstechnik: hell nach dunkel $\rightarrow$ mehr Matrixbandbreite



- $\rightarrow$ Portierung von CSR-SpMV auf GPUs katastrophal
- $\rightarrow$ im Folgenden: ELLPACK-R mit Erweiterung
- $\rightarrow$ im Hinterkopf behalten: Nummerierung der DOFs / Anzahl der nonzeros kritisch

Implementierung von SpMV

ELLPACK-R

- Grundidee: speichere sparse matrix S in zwei arrays `A` (non-zeros in column-major order) und `j` (Spalten-Index für jeden Eintrag in `A`)
- `A` hat (#Zeilen in S) $\times$ (maximale Anzahl non-zeros in Zeilen von S)
- kürzere Zeilen werden aufgefüllt
- zusätzliches array `rl` um die effective Anzahl von non-zeros in Zeile zu speichern (stoppe Berechnung auf einer Zeile richtig)

$$S = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \quad \Rightarrow \quad \texttt{A} = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \texttt{j} = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix} \texttt{rl} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 2 \end{bmatrix}$$

ELLPACK-R

Was gewinnt man mit ELLPACK-R?

$$y_i = \sum_{nz=0}^{rl_i} A_{i,nz} * x_{j_{nz}}$$

- vollständig reguläres Zugriffsmuster auf $\mathbf{y}$ und A
- GPU-Implementierung:
  - ein thread für jedes Element y_i
  - $\rightarrow$ Zugriffe auf die drei ELLPACK-R arrays und $\mathbf{y}$ vollständig coalesced (column-major)
  - $\rightarrow$ Zugriff auf $\mathbf{x}$: verwende texture-cache (FERMI: L2-cache)
  - $\rightarrow$ keine Synchronisation zwischen threads nötig
  - $\rightarrow$ keine branch-divergence
  - zusätzlich: Erweiterung des Produktes so, dass mehrere threads eine Zeile bearbeiten können (ELLPACK-T)
- *Zugriffsmuster auf* $\mathbf{x}$ *hängt stark von Besetzungsmuster von* A *ab* $\rightarrow$ Bandbreite durch DOF-Nummerierung

SpMV in gMG (1)

Nutzung von SpMV für Glätter, Grobgitterlöser

- vorkonditioniertes Richardson Verfahren:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \omega M(\mathbf{b} - A\mathbf{x}^k)$$

- CG oder BiCGStab Verfahren: Anwendung des Vorkonditionierers, Defekte, ...

Glätterkonstruktion

Wir brauchen: starke Glätter

- Beschränkung auf Jacobi katastrophal
- gute Vorkonditionierer oft schwer zu parallelisieren
- Idee: Vorkonditionierungsschritt im Glätter reduziert auf SpMV-Anwendung
- $\rightarrow$ *Sparse Approximate Inverse*-Techniken

SPAI

$$\| I - MA \|_F^2 \quad = \quad \sum_{k=1}^{n} \| e_k^{\mathrm{T}} - m_k^{\mathrm{T}} A \|_2^2 \quad = \quad \sum_{k=1}^{n} \| A^{\mathrm{T}} m_k - e_k \|_2^2$$

wobei e_k der k-te Einheitsvektor und m_k die k-te Zeile von M ist. $\rightarrow$ für n Spalten von $M \rightarrow n$ *least squares* Optimierungsprobleme:

$$min_{m_k} \| A^{\mathrm{T}} m_k - e_k \|_2, \;\; k = 1, \ldots n.$$

- verwende Besetzungsstruktur der Steifigkeitsmatrix als Muster für M

Glätterkonstruktion

starke Glätter: SAINV

- *Stabilised Approximate Inverse*
- berechne Faktorisierung $A^{-1} = ZD^{-1}Z^T$ wobei Z und D explizit berechnet werden: A-Biconjugation angewendet auf die Einheitsbasisvektoren
- Z wird unvollständig assembliert: Elemente unterhalb drop-tolerance werden ignoriert
- keine Strukturvorgabe möglich (im Gegensatz zu SPAI)
- $\rightarrow$ im Einzelfall: bessere Näherung an A^{-1}
- SAINV etwa so gut wie ILU(0)
- Problem: inhärent sequentiell

SpMV in gMG (2)

Nutzung von SpMV für Glätter, Grobgitterlöser: haben wir gesehen

Jetzt: Nutzung von SpMV für den Gittertransfer:

- zwei konforme FE-Räume V_{2h} und V_h
- mit Lagrange-Basis: Interpolation (Gittertransfer) kann durch SpMV ausgedrückt werden

Prolongationsmatrix

$$(P_{2h}^h)_{ij} = \varphi_{2h}^{(j)}(\xi_h^{(i)})$$

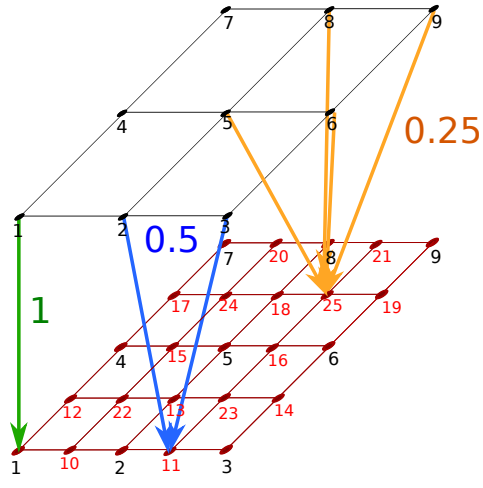
Restriktionsmatrix

$$R_h^{2h} = (P_{2h}^h)^T$$

Transferoperatoren

Beispiel: 2D Q_1

- Knotenpunkte $\xi_h^{(i)}$ von V_h sind identisch mit Vertices $v_h^{(i)}$ des Gitters Ω_h
- Jeder Vertex $v_h^{(i)}$ von Ω_h korrrespondiert entweder mit einem Vertex in Ω_{2h} oder mit dem Mittelpunkt einer Kante eines Quads in Ω_{2h}
- n_{2h}^v #Vertices gesamt, n_{2h}^e #Kanten und n_{2h}^q #Quads von $\Omega_{2h} \Rightarrow$ $n_h^v = n_{2h}^v + n_{2h}^e + n_{2h}^q$ #Vertices in Ω_h



Transferoperatoren

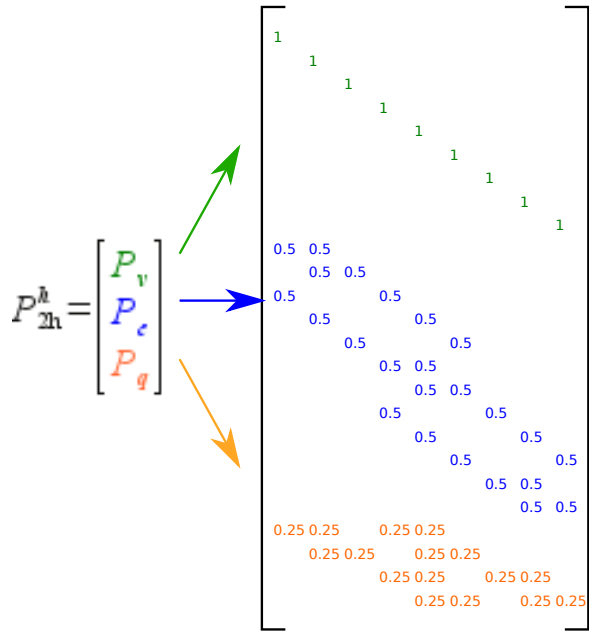
Beispiel: 2D Q_1

- Ausgangspunkt: 2lv Sortierung der Vertices $v_h^{(i)}$ von Ω_h und damit der Basisfunktionen $\varphi_h^{(i)}$ des V_h
- $\Rightarrow$ die $n_h^v \times n_{2h}^v$ Prolongationsmatrix hat Blockstruktur:

$$P_{2h}^h = \begin{bmatrix} P_v \\ P_e \\ P_q \end{bmatrix},$$

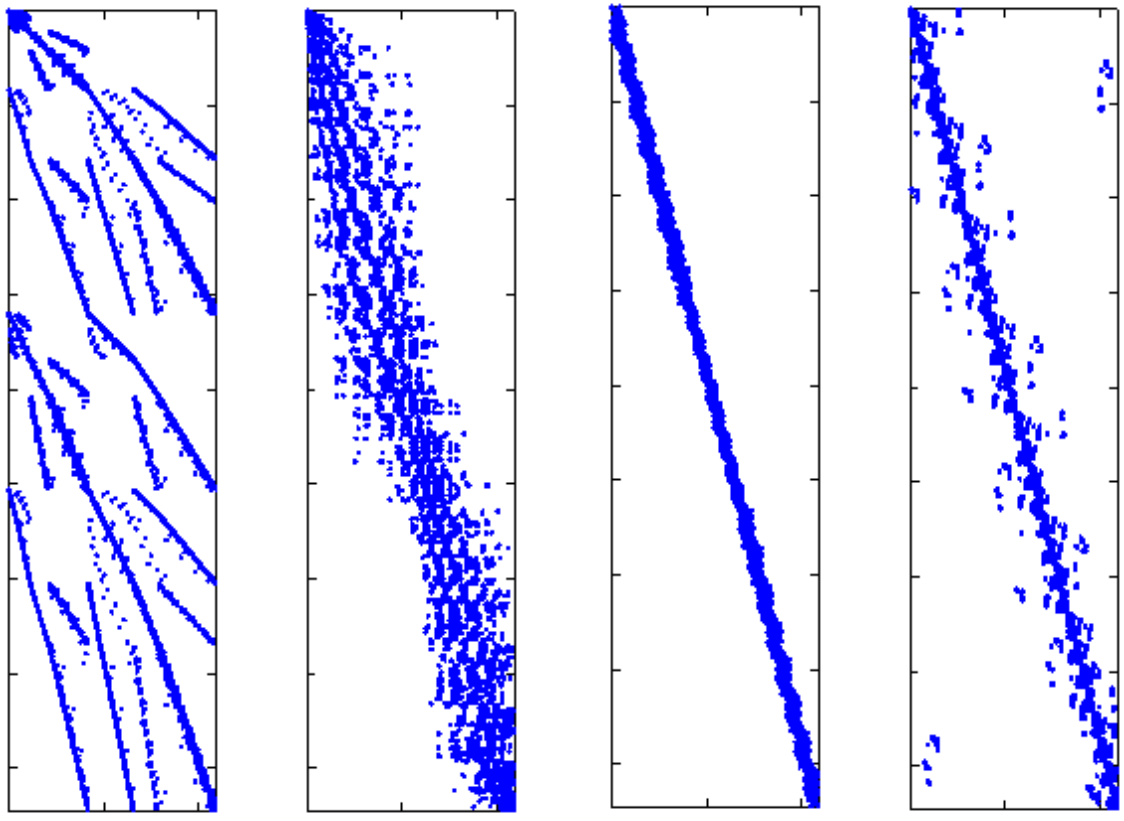
(P_v ist $n_{2h}^v \times n_{2h}^v$ Identitätsmatrix, P_e sind $n_{2h}^e \times n_{2h}^v$ bzw. P_q $n_{2h}^e \times n_{2h}^v$ Matrizen.)

- jede Zeile i in P_e repräsentiert den Vertex in Ω_h, der mit dem Mitteplunkt der Kante i in Ω_{2h} korrespondiert. P_q analog für Quad-Mittelpunkte



Transferoperatoren

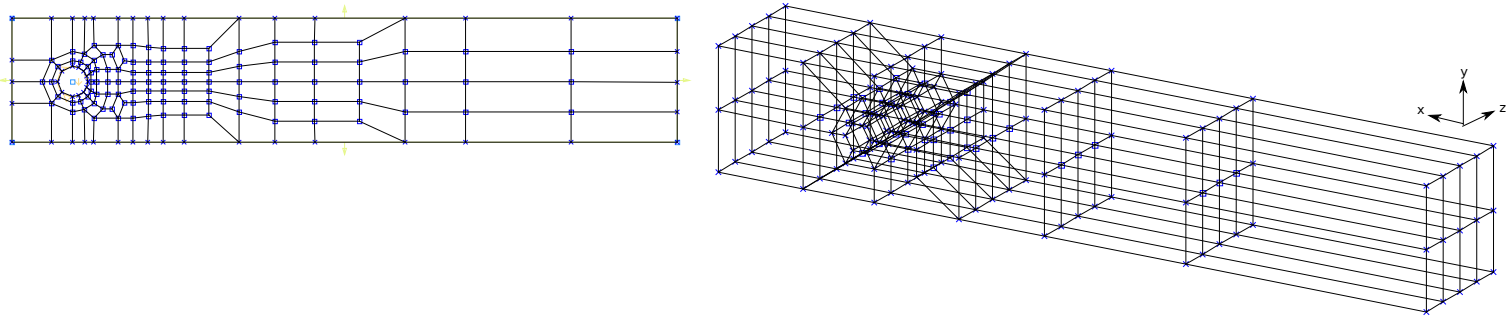
Prolongationsmatrix - Beispiele



- DOF Nummerierungstechnik $\rightarrow$ Performance

Ergebnisse

Beispiel



- populäres Gitter, unstrukturiert, Poisson Problem
- 2D und 3D, Q_1 und Q_2 FE, CPU (Core i7 980X, 6 threads) und GPU (Tesla C2070)
- starke Glättung mit Approximate Inverse Techniken (SPAI, SAINV)

$$\begin{cases} -\Delta u = 1, & \mathbf{x} \in \Omega \\ u = 0, & \mathbf{x} \in \Gamma_1 \\ u = 1, & \mathbf{x} \in \Gamma_2 \end{cases}$$

- verschiedene FE-Räume
- verschiedene DOF numbering Techniken

Ergebnisse

FE-gMG: Ergebnisse (2D)

Q1	CPU								GPU									
	Jacobi		SPAI			SAINV			Jacobi			SPAI				SAINV			
sort	time	#iter	time	#iter	speedup jac	time	#iter	speedup jac	time	#iter	speedup cpu	time	#iter	speedup jac	speedup cpu	time	#iter	speedup jac	speedup cpu
2lv	4.04	13	2.54	5	1.59	3.59	6	1.12	1.06	13	3.82	0.56	5	1.88	4.53	1.19	6	0.89	3.01
CM	3.65	13	2.19	5	1.66	3.29	6	1.11	1.03	13	3.55	0.72	5	1.43	3.05	0.82	6	1.26	4.03
XYZ	3.48	13	2.06	5	1.69	4.44	9	0.78	0.98	13	3.53	0.51	5	1.93	4.04	1.03	9	0.96	4.32
Stoch	4.04	13	2.57	5	1.57	3.19	5	1.27	1.74	13	2.33	1.04	5	1.66	2.46	1.29	5	1.35	2.47
Hie	3.49	13	2.07	5	1.69	3.07	6	1.14	0.97	13	3.59	0.50	5	1.94	4.14	0.77	6	1.26	3.98

Q2	CPU								GPU									
	Jacobi		SPAI			SAINV			Jacobi			SPAI				SAINV			
sort	time	#iter	time	#iter	speedup jac	time	#iter	speedup jac	time	#iter	speedup cpu	time	#iter	speedup jac	speedup cpu	time	#iter	speedup jac	speedup cpu
2lv	13.19	22	4.87	5	2.71	10.24	9	1.29	2.27	22	5.80	0.93	5	2.44	5.22	2.04	9	1.12	5.02
CM	11.40	22	4.40	5	2.59	10.58	12	1.08	2.50	22	4.56	1.02	5	2.44	4.30	2.20	12	1.13	4.80
XYZ	11.29	22	4.21	5	2.68	9.58	12	1.18	2.41	22	4.69	0.99	5	2.44	4.26	2.70	12	0.89	3.55
Stoch	12.92	22	5.14	5	2.51	4.64	9	2.79	4.78	22	2.70	2.04	5	2.35	2.52	1.57	9	3.05	2.96
Hie	11.25	22	4.24	5	2.66	8.68	9	1.30	2.44	22	4.60	1.00	5	2.43	4.22	1.88	9	1.30	4.61

Ergebnisse

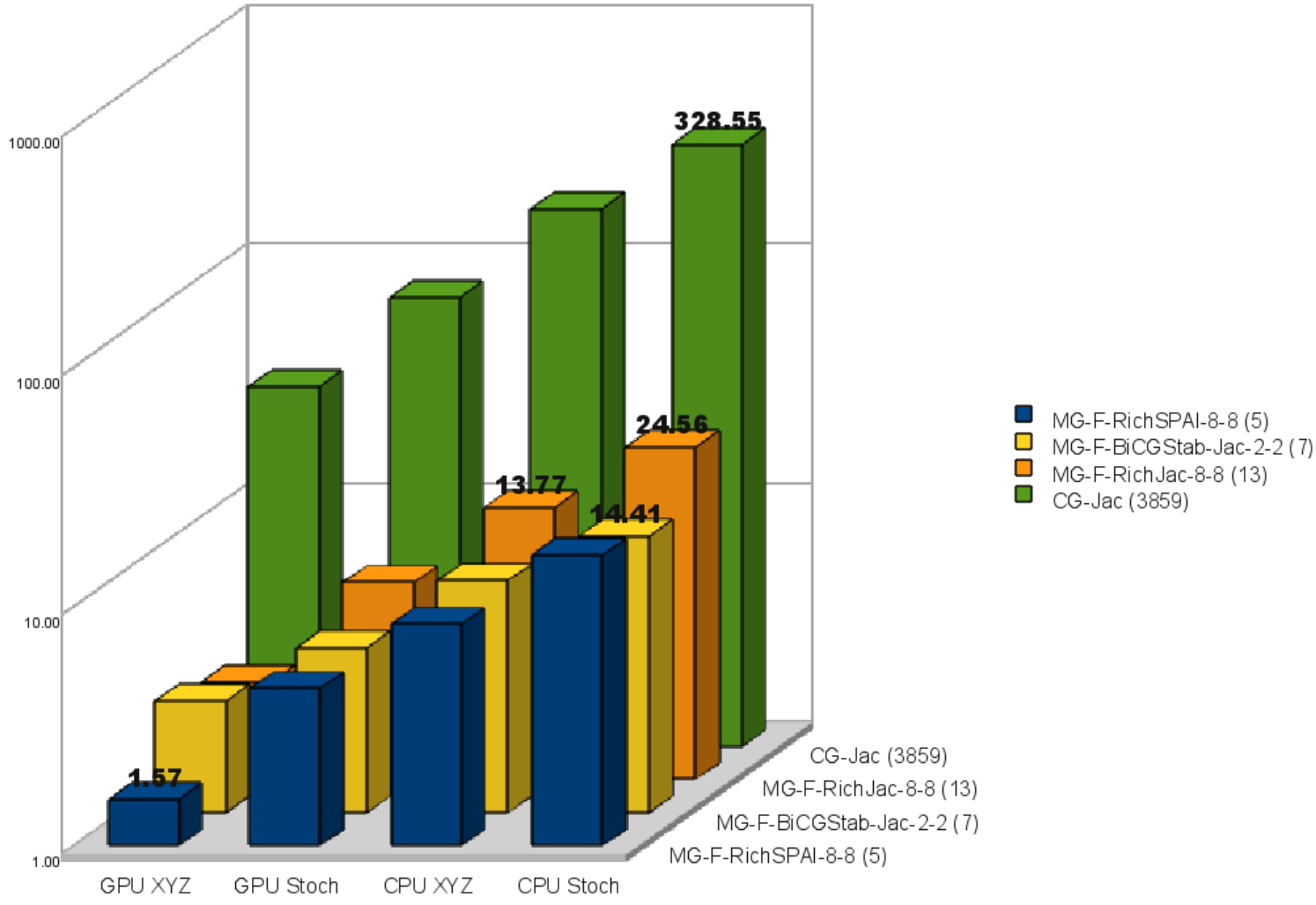
FE-gMG: Ergebnisse (3D)

Q1	CPU								GPU								
	Jacobi		SPAI			SAINV			Jacobi			SPAI				SAINV			
sort	time	#iter	time	#iter	speedup jac	time	#iter	speedup jac	time	#iter	speedup cpu	time	#iter	speedup jac	speedup cpu	time	#iter	speedup jac	speedup cpu
2lv	2.43	26	1.08	7	2.25	1.03	9	2.37	0.66	26	3.71	0.27	7	2.39	3.94	0.28	9	2.32	3.63
CM	2.34	26	1.02	7	2.30	0.98	9	2.37	0.66	26	3.53	0.28	7	2.39	3.67	0.29	9	2.26	3.36
Stoch	2.63	26	1.18	7	2.23	1.28	10	2.06	0.75	26	3.48	0.33	7	2.32	3.61	0.38	10	1.98	3.35

Q2	CPU								GPU								
	Jacobi		SPAI			SAINV			Jacobi			SPAI				SAINV			
sort	time	#iter	time	#iter	speedup jac	time	#iter	speedup jac	time	#iter	speedup cpu	time	#iter	speedup jac	speedup cpu	time	#iter	speedup jac	speedup cpu
2lv	9.86	42	3.09	8	3.19	2.44	10	4.04	2.01	42	4.90	0.58	8	3.44	5.29	0.56	10	3.60	4.37
CM	7.46	42	2.50	8	2.99	2.41	12	3.10	2.31	42	3.23	0.70	8	3.32	3.59	0.73	12	3.18	3.32
Stoch	8.89	42	3.14	8	2.83	2.90	12	3.07	2.92	42	3.04	0.92	8	3.18	3.41	0.92	12	3.19	3.16

Ergebnisse

FE-gMG: Ergebnisse



Ergebnisse

FE-gMG

- völlig unstrukturierte (Teil-) Gitter möglich
- erstmals vollständiger geometrischer MG auf GPU (unstrukturiert)
- hohes Erweiterbarkeitspotential Glätter/Vorkonditionierer: SPAI, SAINV, ILU, poly.
- Nummerierung der Freiheitsgrade ist kritisch

Die Kombination von zugeschnittenen numerischen Methoden und Hardwareorientierung liefert bis zu 3 Gr.-Ordnungen speedup. Daran hat die numerische Effizienz den Hauptanteil.

Der nächste Schritt

Assemblierung und GPUs

- Verschiedene Matrizen und Erzeugungsalgorithmen in FE-gMG
  - FEM-spezifisch (abhängig von Gitter/Nummerierungsart, PDE): Steifigkeit, Masse
  - gMG-spezifisch (abhängig von Gitter/Nummerierungsart): Transfermatrizen
  - Glätterspezifisch (Abhängig von Systemmatrizen, Algorithmus): Approximate Inverses
- potentiell sehr unterschiedliche Eigenschaften (Bandbreite, #non-zeros / Zeile, #non-zeros, ...)
- Speichertechniken sind unterschiedlich stark für unterschiedliche Architekturen
- Speichertechniken sind unterschiedlich stark für verschiedene Aufgaben
  - SpMV: ELLPACK-T sehr gut für GPUs
  - Assemblierung: andere Formate gebraucht $\rightarrow$ wahlfreier Zugriff

Der nächste Schritt

Assemblierung und GPUs

- FEM-Matrizen: Potentiell viele Möglichkeiten auf GPUs: Beispiel CSR-basiert
- Ziel: minimiere globale Speichertransaktionen, nutze Speicherhierarchien aus, balanciere Redundanz und Effizienz
  - 1 Thread pro Zeile? (extra-lookups in die Spalten, stark abhängig von konkretem Speicherformat)
  - 1 Thread pro non-zero? (Schlechte Balance z.B.: zwischen Haupt- und Nebendiagonalen)
  - 1 Thread pro Element? (race-conditions bei den updates)

$\rightarrow$ **in Arbeit**

Der nächste Schritt

Assemblierung und GPUs

- Sparse Approximate Inverse: Potentiell viele Erzeugungsalgorithmen
  - SPAI: gut parallelisierbar
  - SAINV: manchmal numerisch besser aber inhärent sequentiell
  - ndere Approximierte Inverse: Polynomielle Vorkonditionierer?
  - temporale Überlappung mit Erzeugung der Systemmatrizen?
- Transferoperatoren
  - Blockweiser Ansatz?

$\rightarrow$ **in Arbeit**

Fazit

Zusammenfassung

- FE-gMG ist ein Beispiel für simultane Berücksichtigung von Numerischer und Hardwareeffizienz
- hohe Flexibilität unstrukturierter Gitter
- Ausnutzung von GPUs auch für sehr komplexe Anwendungen

Zukünftige / derzeitige Arbeiten

- Assemblierung
- Optimierung: Verluste in Transferoperatoren
- Performancemodellierung in ScaRC / Verbesserung numerischer Skalierbarkeit

Danksagung

Unterstützt durch das BMBF, *HPC Software für skalierbare Parallelrechner: SKALB Projekt 01IH08003D*.

Unterstützt durch die DFG, *SFB 708 TP B1 & SPP 1423, TU 102/32-2*.

Dank an das FEAST-Team.

Dank an alle Entwickler von HONEI (`www.honei.org`).

Dank an das LiDO-Team @ DOWIHR.