# Hochperformante Erdbebensimulationen[1]

Alexander Nikolas Breuer[2]

**Abstract:** Das Verständnis der Erdbebendynamik wird von hochauflösenden, gekoppelten Simulationen des Bruchprozesses und der seismischen Wellenausbreitung unterstützt. Für die benötigten hohen Auflösungen wird eine immense Menge an Höchstleistungsrechenressourcen verwendet, daher ist eine optimale Ausnutzung durch die Software unerlässlich. Getrieben durch aktuelle Entwicklungen in der Hardware erfordern die höheren Anforderungen an Parallelisierung und Datenlokalität häufig das Ersetzen ganzer Softwareteile, um gleichzeitig eine effiziente Numerik und Maschinenauslastung zu gewährleisten.

In dieser Dissertation präsentiere ich einen neuen Rechenkern für die seismische Simulationssoftware SeisSol. Der neue Kern maximiert den Wert und Durchsatz der Gleitkommaoperationen in der zugrundeliegenden ADER-DG Diskretisierungsmethode, um die Rechenzeit zum gewünschten Ergebnis zu minimieren. Beinhaltet sind automatisch optimierte Matrixkernel, hybride Parallelisierung von Vielkernarchitekturen bis hin zum kompletten Großrechner, sowie ein hochperformantes gruppiertes lokales Zeitschrittschema.

Der präsentierte Kern reduziert die Rechenzeit von SeisSol um einen substantiellen Faktor und skaliert bis hin zu mehr als einer Millionen Recheneinheiten. Durch den Kern wurde eine wegweisende Simulation des Landers-Erdbebens von 1992 auf einem kompletten Großrechner ermöglicht. Zum ersten Mal erlaubte diese Simulation die Analyse des damit verbundenen komplexen Bruchprozesses, welcher aus der nichtlinearen Interaktion des Reibungsprozesses gekoppelt an die seismische Wellenausbreitung resultiert, in einer komplizierten Geometrie.

## 1 Einleitung

**Erdbebengefährdung am Beispiel Kaliforniens:** Schätzungsweise 143 Millionen Menschen (46 % der Bevölkerung) leben, bezogen auf die Vereinigten Staaten, in Gebieten mit Potential für Erschütterungen (0.1 g) durch Erdbeben [Ja15]. Von diesen sind 44 Millionen durch sehr starke Erschütterungen (0.4 g) gefährdet [Ja15]. Mehr als 30 Millionen davon leben allein im Bundesstaat Kalifornien[3]. Die Metropolregionen San Francisco, Los Angeles und San Diego sind verantwortlich für 40 % des gemittelten, landesweiten Schadens durch Erdbeben [Fe08]. Während bereits die Zerstörung oder Beschädigung von Gebäuden durch seismische Wellen eine große Gefahr darstellt, können auch Sekundärfolgen großen Schaden anrichten. Beispiele sind gegeben durch Autounfälle, Folgen von Stromausfällen, Ausfälle in der Wasserversorgung, Tsunamis, Erdrutsche, Bodenverflüssigung, Schäden durch Verwerfungen, Dammbrüche, Feuer oder die Freisetzung von gefährlichen Materialien.

[1] Englischer Titel der Dissertation: “High Performance Earthquake Simulations”

[2] University of California, San Diego, anbreuer@ucsd.edu

[3] `https://web.archive.org/web/20160215065038/http://www.usgs.gov/blogs/features/usgs_top_story/nearly-half-of-americans-exposed-to-potentially-damaging-earthquakes/`

**Dynamische Bruchsimulationen:** Erdbeben sind ein Multiskalenphänomen. Seismische Wellen bewegen sich durch die komplette Erde und können noch tausende Kilometer entfernt von der Quelle gemessen werden. Im Gegensatz dazu ist die Entstehung von Erdbeben ein höchst lokaler Prozess mit kritischen Auflösungen im Meterbereich. Dieser nichtlineare Prozess ist ein wichtige Bestandteil in der seismische Gefahrenanalyse. Computersimulationen stellen daher ein unersetzbares Werkzeug dar, um die Ursachen und Effekte von Erdbeben zu simulieren. Die dynamische Simulation des Bruchprozesses stellt hohe Anforderungen an die Simulationsumgebungen. Realistische Darstellungen der Verwerfungen führen einen hohen Grad an geometrischer Komplexität ein, da die entsprechenden Verwerfungssysteme hochkomplex sind und als nichtplanare Flächen beschrieben werden (z.B. [Pl07]).

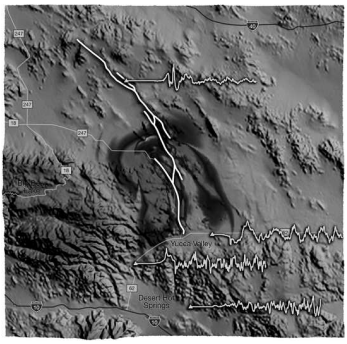


Abb. 1: Visualisierung der Geschwindigkeitsamplitude für das Landers-Erdbeben von 1992. Das komplexe Verwerfungssystem ist weiß. Quelle: [He14].

Der Bruch generiert seismische Wellen, welche sich durch das Volumen des simulierten Bereichs bewegen. Treffen diese Wellen wiederum auf eine Verwerfung, so können erneut überkritische Zustände entstehen, welche weitere Brüche mit entsprechenden Reibungsprozessen in Gang setzen. Folglich sind präzise Simulation von seismischer Wellenausbreitung und des dynamischen Bruchprozesses gleich wichtig. Die Softwareumgebung *SeisSol* ist Thema dieser Arbeit und benutzt die Finite Elemente Methode zur räumlich Diskretisierung. Das verwendete, unstetige Galerkin-Verfahren ermöglicht, zusammen mit der ADER-Diskretisierung in der Zeit, die genaue Darstellung von Verwerfungssystemen, Topographie und heterogenen Materialeigenschaften [Du06, Pe12, Pu09].

In dieser Arbeit präsentiere ich einen neuen Rechenkern für SeisSols rechenintensive Komponente, welche die seismische Wellenausbreitung simuliert. Mein Kern ist ausgelegt auf komplexe Geometrien und heterogene Materialparameter, berücksichtigt Regularitätsanforderungen moderner Höchstleistungsrechner bereits in der algorithmischen Designphase, und berücksichtigt alle Level an Parallelität in der Optimierung.

## 2 Algorithmische Grundlagen

SeisSol löst die elastischen Wellengleichungen. In Differentialform können die elastischen Wellengleichungen als ein lineares System hyperbolischer, partieller Differentialgleichungen mit variablen Koeffizienten formuliert werden:

$$q_t + A^x q_x + A^y q_y + A^z q_z = 0. \tag{1}$$

$q(\vec{x},t) = (u,v,w,\sigma_{xx},\sigma_{yy},\sigma_{zz},\sigma_{xy},\sigma_{xz},\sigma_{yz})^T$ ist der Vektor der elastischen Variablen. Die Partikelgeschwindigkeiten in kartesischer $x$-, $y$- und $z$-Richtung sind gegeben durch $u$, $v$ und $w$. $\sigma_{xx}$, $\sigma_{yy}$ und $\sigma_{zz}$ sind die drei Normalspannungen, und $\sigma_{xy}$, $\sigma_{xz}$, $\sigma_{yz}$ die drei Scherspannungen. $A^x(\vec{x})$, $A^y(\vec{x})$ und $A^z(\vec{x})$ sind die räumlich variablen Jacobi-Matrizen, welche

den Einfluss der Materialparameter auf (1) kodieren. SeisSol benutzt unstrukturierte Tetraedergitter für die räumliche Diskretisierung. In Kombination mit einem Satz von $B_O$ Basisfunktionen, erhalten wir die Freiheitsgrade $Q_k$ in jedem Element. Die Freiheitsgrade $Q_k$ (Größe $B_O \times 9$) fassen die $B_O$ modalen Koeffizienten der Basisfunktionen in Tetraeder $k$ für jede der neun elastischen Variablen zusammen. Typische Konvergenzordnungen in Produktionsläufen sind $O = 5$ oder $O = 6$ und resultieren in $B_5 = 35$ bzw. $B_6 = 56$ Basisfunktionen.

**Operatoren in aller Kürze:** Mein neuer Rechenkern für SeisSol fasst die Integratoren der ADER-DG-Maschinerie in Operatoren zusammen. Diese Operatoren setzen sich wiederum aus Matrixoperationen zusammen. Neben einer kompakten Zusammenfassung, ermöglicht uns diese Darstellung eine Trennung von Elementabhängigkeiten in den jeweiligen Zeitschritten. Dies ist wichtig, um korrekte und effiziente Parallelisierung auf allen Ebenen eines Höchstleistungsrechners zu erreichen.

Unser erster Operator, der **ADER-Operator**, betrachtet die elementlokalen Freiheitsgrade und gibt eine Vorhersage wie diese sich, im Bezug auf das Element, in der Zeit verhalten. Der Operator ist somit komplett elementlokal und hat keine Datenabhängigkeit zu benachbarten Elementen. Mit den Freiheitsgraden des Elements $k$ zum aktuellen Zeitschritt $t_k^{n_k}$ als Anfangswert, $\partial^0/\partial t^0 Q_k(t_0) = Q_k^{n_k}$, ist der ADER-Operator gegeben durch:

$$\frac{\partial^{d+1}}{\partial t^{d+1}} Q_k(t_0) = -\sum_{c=1}^{3} \hat{K}^{\xi_c} \left( \frac{\partial^j}{\partial t^j} Q_k(t_0) \right) A_k^{\xi_c}. \tag{2}$$

$\hat{K}^{\xi_c}$ sind die drei globalen, transponierten Steifigkeitsmatrizen (Größe $B_O \times B_O$) welche in der Vorverarbeitung mit der diagonalen, inversen Massenmatrix multipliziert wurden. Der Ausdruck *global* sagt, dass diese mit Hilfe eines eindeutigen Referenzelement definiert sind, der ADER-Operator also, unabhängig vom aktuellen Element, stets auf die gleichen Matrizen zurückgreift. Dies ist besonders mit Blick auf Datenlokalität in modernen Cache-Hierarchien eine wichtige Eigenschaft. Die drei elementlokalen Matrizen $A_k^{\xi_c}$ (Größe $9 \times 9$) sind Linearkombinationen der Jacobi-Matrizen [Du06]. Abb. 2 zeigt den ADER-Operator für eine Diskretisierung mit Ordnung $O = 5$, bzw. $B_5 = 35$ Basisfunktionen. Hier ist die Besetztheitsstruktur der Matrizen in grau dargestellt. Zeilen der Freiheitsgrade $Q_k^{n_k}$ bzw. von $\partial^d/\partial t^d Q_k(t_0)$ sind den Basisfunktionen zugeordnet, wobei die elastische Variablen als Spalten dargestellt sind. Orangefarbenen und grauen Blöcke zeigen eine besondere Eigenschaft der transponierten Steifigkeitsmatrizen auf. Orangefarbene Blöcke enthalten Nullwerte und erzeugen Nullblöcke in den jeweiligen Spalten des Ergebnisses von (2). Im Gegensatz dazu enthalten graue Blöcke Nichtnulleinträge, treffen aber im dargestellten Aufruf auf einen erzeugten Nullblock. Wir nutzen diese Eigenschaft aus, um a) die Zahl der Operationen im ADER-Operator zu reduzieren [Br14] und b) den Speicherbedarf im Falle einer Ausführung mit lokalen Zeitschritten zu reduzieren [Br15b].

Unser zweiter Operator ist der **Elementoperator**, welcher alle Aktualisierung eines Zeitschrittes enthält, die ausschließlich Daten des Elements selbst benötigen. Der Elementoperator setzt sich aus der Volumenintegration und einem Teil der Randintegration zusammen. Eingabe für den Elementoperator sind Linearkombinationen $T_k$ der $\partial^d/\partial t^d Q_k(t_0)$, welche

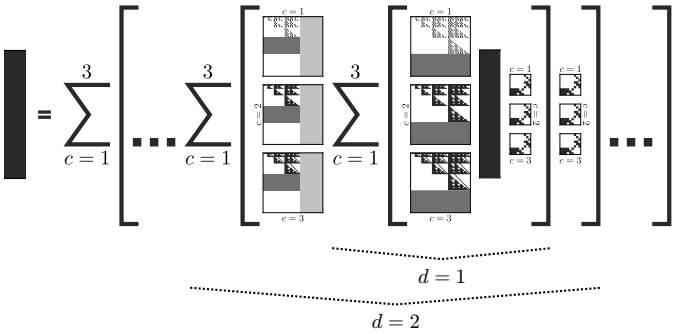



Abb. 2: Visualisierung des ADER-Operators (2). Die Besetztheitsstrukturen der Matrizen sind grau. Orangefarbene Blöcke erzeugen Nullblöcke, graue Blöcke treffen auf erzeugte Nullblöcke. Quelle: [Br15b]

im ADER-Operator (2) berechnet wurden:

$$Q_k^{*,n_k+1} = Q_k^{n_k} + \sum_{c=1}^{3} \tilde{K}^{\xi_c}(T_k) A_k^{\xi_c} - \sum_{i=1}^{4} \hat{F}^{-,i}(T_k) \hat{A}_{k,i}^{-}. \tag{3}$$

$\tilde{K}^{\xi_c}$ sind die drei globalen Steifigkeitsmatrizen (Größe $B_O \times B_O$), $\hat{F}^{-,i}$ vier globale Flussmatrizen (Größe $B_O \times B_O$). $\tilde{K}^{\xi_c}$ und $\hat{F}^{-,i}$ wurden in der Vorverarbeitung wieder mit der inversen Massenmatrix multipliziert. Die elementlokalen Matrizen $\hat{A}_{k,i}^{-}$ (Größe $9 \times 9$) lösen, bezogen auf die Seitenflächen der Tetraeder, lokale Riemann-Probleme. Analog zum ADER-Operator, zeigt Abb. 3 den Elementoperator für ein Verfahren mit Konvergenzordnung $O = 5$.

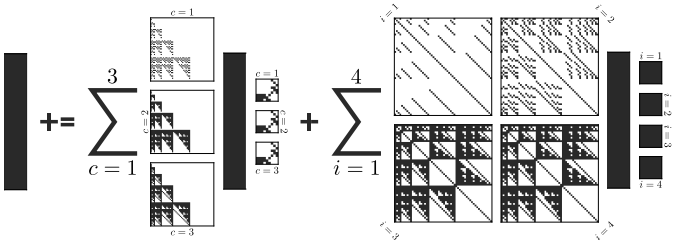



Abb. 3: Visualisierung des Elementoperators (3). Die Besetztheitsstrukturen sind grau. Quelle: [Br15b]

Der dritte und letzte Operator ist gegeben durch den **Nachbaroperator**, welcher Datenabhängigkeiten bezogen auf Nachbarelementen auflöst. Das ADER-DG-Verfahren ist in dieser Hinsicht optimal, da nur Daten von benachbarten Elementen benötigt werden, welche über Flächen an das aktuelle Element angrenzen. In unserem Fall hängt der Elementoperator von Tetraeder $k$ also nur von den vier, über Dreiecksflächen benachbarten Tetraedern $k_i,\ i \in 1,\ldots,4$ ab. Der Nachbaroperator vervollständigt die Aktualisierung der Freiheitsgrade und wir erhalten als Ergebnis die Freiheitsgrade von Element $k$ zum nächsten

Zeitschritt:

$$Q_k^{n_k+1} = Q_k^{*,n_k+1} - \sum_{i=1}^{4} \hat{F}^{+,i,j_k,h_k}\left(T_{k_i}\right)\hat{A}_{k,i}^{+}. \tag{4}$$

$\hat{F}^{+,i,j,h}$, $j \in 1,\ldots,4$, $k \in 1,\ldots,3$ sind 48 globale Flussmatrizen (Größe $B_O \times B_O$), welche in der Vorverarbeitung wieder mit der inversen Massenmatrix multipliziert wurden. Die Wahl der Indizes $j_k$ und $h_k$ hängt von den Knoten ab, welche zwei benachbarte Tetraeder im Bezug auf ihre Abbildung auf das Referenzelement teilen. Die elementlokalen Matrizen $\hat{A}_{k,i}^{+}$ (Größe $9 \times 9$) lösen, bezogen auf die Seitenflächen der Tetraeder, lokale Riemann-Probleme. Abb. 4 visualisiert den Nachbaroperator für ein Verfahren der Konvergenzordnung $O = 5$.

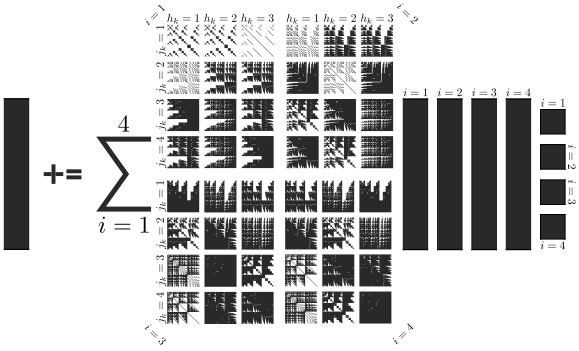



Abb. 4: Visualisierung des Nachbaroperators (4). Die Besetztheitsstrukturen sind grau. Quelle: [Br15b]

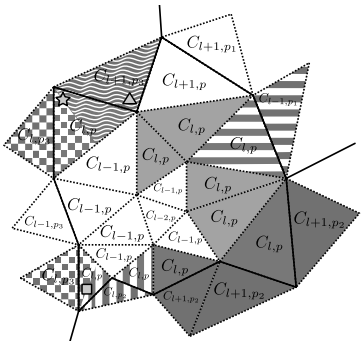
**Lokale Zeitschritte:** Die Wahl eines numerisch stabilen Zeitschrittes für das Schema in (3) und (4) hängt von der CFL-Bedingung ab. Die CFL-Bedingung berücksichtigt die Geschwindigkeit mit der Information durch ein Element propagiert und limitiert dementsprechend den Zeitschritt. Der maximale, elementlokale Zeitschritt $\Delta t_k^{\mathrm{CFL}}$ ist limitiert durch die maximale Wellengeschwindigkeit, den Inkugeldurchmesser und die Konvergenzordnung des Verfahrens. Wir erlauben für jedes Element einen maximalen Zeitschritt $\Delta t_k^{\mathrm{CFL}}$, welcher 50 % der Stabilitätsbedingungen von Runge-Kutta-Verfahren entspricht [Du06]. Wie in [Du07] vorgestellt, erlaubt das ADER-Verfahren die Aktualisierung jedes Elementes in (3) und (4) mit dem lokalen Zeitschrittlimit, $\Delta t_k = \Delta t_k^{\mathrm{CFL}}$. Jedoch führt dieser Ansatz zu einem sehr heterogenen Rechenschema und limitiert effiziente Implementierungen. Im Gegensatz dazu schränkt mein lokales Zeitschrittverfahren die möglichen Beziehungen zweier benachbarter Elemente auf zwei grundlegende Fälle ein.

Nehmen wir ein gegebenes Tetraeder $k$ mit benachbartem Element $k_i$ und den dazugehörigen Zeitschritten $\Delta t_k$ und $\Delta t_{k_i}$ an. Wir unterscheiden zwischen den beiden Fällen $\Delta t_k = \frac{1}{r} \cdot \Delta t_{k_i}$, $r \in \mathbb{N}^+$ und $\Delta t_k = r \cdot \Delta t_{k_i}$, $r \in \mathbb{N}^+$. Die Beziehung $\Delta t_k = \Delta t_{k_i}$ $(r = 1)$ ist in beiden Fällen enthalten und die Entscheidung, welcher Fall verwendet wird, hängt von der Zeitschrittkonfiguration der Nachbarelemente ab. Unsere Limitierung der Zeitschrittunterschiede auf $r \in \mathbb{N}^+$ anstelle von beliebigen Raten $r \in \mathbb{R}^+$ ist nicht von dem ADER-Verfahren erfordert und geschieht aus Effizienzgründen. Hierbei reduzieren wir die algorithmische Komplexität des lokalen Zeitschrittverfahrens und berücksichtigen bereits im

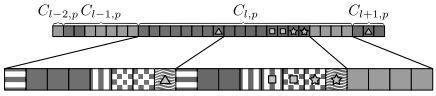
Design die effiziente Ausführung auf einem Höchstleistungsrechner. Dazu opfern wir hier und im Folgenden einen Teil numerischen Optimalität für einen deutlich erhöhten Durchsatz an Gleitkommaoperation in der finalen Implementierung.

## 3 Speicherlayout

Mein Rechenkern beinhaltet ein Speicherlayout für alle Datenstrukturen, das sich mittels einer Sortierung der Elemente an die Rechenoperationen anschmiegt und Regularitätsanforderungen von modernen Großrechnern gerecht wird. Wir fassen alle Elemente, welche gemäß des vorigen Kapitels eine gemeinsame Zeitschrittlänge haben, in einer Gruppe $C_l$ zusammen. Folglich überdecken die $L$ disjunkten Gruppen $C_l$, $l \in 1,\ldots,L$ das komplette Intervall der möglichen Zeitschritte und haben jeweils einen Zeitschrittunterschied, gegeben durch die Rate $r$ in den beiden möglichen Zeitschrittbeziehungen. In jeder Partition $p$ bzgl. des verteilten Speichers eines Höchstleistungsrechner sortieren wir alle Elemente gemäß ihre Zugehörigkeit zu den Gruppen $C_l$, erhalten die Partition-Gruppen-Kombination $C_{l,p}$, und garantieren somit linearen Speicherzugriff, wenn wir über eine Gruppe mittels ADER- (2) und Elementoperator (3) iterieren. Der Nachbaroperator (4) folgt für die Zugriffe auf $T_{k_i}$ dem unstrukturierten Gitter und uneingeschränkte Linearität ist nicht gegeben. Jedoch sind auch hier die Zugriffe auf die Freiheitsgrade des Elements selbst linear.




(a) Nur Elemente von $C_{l,p}$ sind eingefärbt.




(b) Sortierung der Elemente im Speicher.

Abb. 5: Exemplarisches Speicherlayout für eine Partition $p$. Quelle: [Br16, Br15b]

In der nächsten Ebene sortieren wir die Gruppen $C_l$ in jeder Partition $p$ gemäß den Kommunikationsstrukturen. Wir unterscheiden zwischen Elementen, die an Kommunikation im Bezug auf den verteilten Speicher involviert sind und inneren Elemente, die bzgl. eines Zeitschrittes unabhängig von entfernten Daten sind. Innerhalb der Elemente, welche an Kommunikation beteiligt sind, sortieren wir weiter nach Regionen. Somit speichern wir Daten einer Nachricht, die über das Netzwerk verschickt wird, linear im Speicher und müssen diese nicht zunächst aus dem Speicher zusammensuchen.

Abb. 5 illustriert für ein kleines zweidimensionales Beispiel diese Sortierung der Elemente im Speicher. Partition $p$ grenzt an Partitionen $p_1$, $p_2$ und $p_3$ an. Partitionsgrenzen sind in Abb. 5a mittels durchgezogener Linien dargestellt. Die Zeitschrittgruppen $C_{l-2}$, $C_{l-1}$, $C_l$ und $C_{l+1}$ sind in Partition $p$ vertreten, wobei das Speicherlayout von $C_{l,p}$ hervorgehoben ist. Die vier inneren Elemente sind grau, Elemente die Daten zu benachbarten Partitionen senden sind blau und Element-

daten, welche von Nachbarn empfangen werden, in Orange dargestellt. Die Muster in der Farbgebung unterscheiden die einzelnen Kommunikationsregionen. So kommuniziert beispielsweise die Gruppe $C_l$ in Partition $p$ und $p_3$ mittels der gekachelten Elemente. Die Elemente mit einem gelben Dreieck, Stern und Quadrat sind Sonderfälle, da hier einzelne Elemente an mehreren Regionen beteiligt sind und von uns dupliziert werden. Der obere Teil von Abb. 5b zeigt die Einordnung von Elementen in Zeitschrittgruppen und Kommunikationsbereiche. Zusätzlich zeigt der untere Teil die Anordnung der einzelnen Regionen im Speicher. Hier empfangen wir beispielsweise die beiden orangefarbene, gekachelten Elemente in jedem Zeitschritt von $C_{l,p}$. Analog stellen die beiden blauen, gekachelten Bereiche eine ausgehende Nachricht dar.

Auf der untersten Ebene, welche den Gleitkommaoperationen am nächsten ist, füllt unser Speicherlayout die Freiheitsgrade $Q_k$ und die Ergebnisse des ADER-Operators (2) mit Nullen in den Spalten auf (siehe auch Abb. 2, 3, 4). Dies versichert Alignment der Daten auf 16 Byte (SSE3), 32 Byte (AVX, AVX2), bzw. 64 Byte Grenzen (KNC ISA, AVX-512) und erlaubt die Verwendung von schnellen aligned-load und aligned-store Operationen in den Matrixoperatoren [Br15a, Br15b].

## 4 Parallelisierung

**Einzelner Kern:** Auf der Ebene eines einzelnen Kerns greift die Parallelisierung auf die LIBXSMM-Bibliothek für die Ausführung der Matrixoperationen im ADER-, Element- und Nachbaroperator zurück. LIBXSMM ermöglicht die Generierung von Code, der speziell auf die kleinen Matrix-Matrix-Multiplikationen in den Operatoren und die jeweilige Zielarchitektur zugeschnitten ist. Zu diesem Zweck teilen wir dem Codegenerator die Struktur der Operatoren (siehe Abb. 2, 3, 4) mit und integrieren die speziellen Codeteile zur Compilezeit. Ein zusätzlicher Schritt selektiert individuell zwischen Matrixoperatoren, die die Besetztheit der Matrizen ausnutzen und solchen, die auf dicht besetzten Blöcken rechnen. Ähnlich zu den Optimierungen im Zeitschrittverfahren, wählen wir den optimalen Mittelweg zwischen Minimierung der benötigten Gleitkommaoperationen und der Maximierung des Durchsatzes. Die Codegenerierung war zunächst ein eigenständiger Teil von SeisSol [Br13, Br14, He14] und wurde später in LIBXSMM integriert [He15, Br15a, Br16].

**Gemeinsamer Speicher:** Das strikte Speicherlayout aus Kap. 3 ermöglicht eine einfache, aber hocheffiziente Parallelisierung für Rechenknoten mit gemeinsamem Speicher. Die Berechnung des ADER- und Elementoperators erfolgt in einer ersten Schleife, und die Berechnung des Nachbaroperators in einer zweiten Schleife. Hier teilen wir die zu berechnenden Elemente bei beiden Schleifeneintritten statisch auf die verfügbaren Recheneinheiten auf und versichern somit implizit, dass jedes Element einem ihm zugewiesenen Kern hat. Dieser Struktur folgen wir auch bei der Initialisierung der Datenstrukturen. Das Ergebnis ist eine Zuweisung von Speicher in Hardware, der dem berechnenden Kern am nächsten liegt. So maximieren wird die verfügbare Bandbreite und können, zusammen mit Prefetches in Hardware und Software, für hohe Konvergenzraten die Speicherzugriffe hinter Berechnungen verstecken.

**Verteilter Speicher:** Die Parallelisierung des Rechenkerns für Systeme mit verteiltem Speicher geschieht mithilfe des Message Passing Interface (MPI). Auf modernen Vielkernarchitekturen reservieren wir einen Kern pro Rechenknoten exklusiv für die Kommunikation mittels MPI. Neben der Verwaltung der Sende- und Empfangsoperationen ist dieser Kern für die Progression der Nachrichten zuständig. Dies ist wichtig, um echte asynchrone Kommunikation zu erreichen, da die reine Verwendung von nicht-blockierender Kommunikation in aktuellen MPI-Implementierung keine überlappende Kommunikation sicherstellt. Wir kombinieren diesen Ansatz mit einer dynamischen Berechnung von Arbeitspaketen. Jede Zeitschrittgruppe $C_l$ in einer Partition $p$ berechnet einen Zeitschritt mittels vier Arbeitspaketen. Das erste Arbeitspaket $\mathrm{WP1}_{l,p}$ berechnet den ADER- und Elementoperator für Zellen, die Daten für benachbarte Partitionen zu Verfügung stellen müssen. $\mathrm{WP2}_{l,p}$ berechnet ADER- und Elementoperator für von Kommunikation unabhängige Zellen. $\mathrm{WP3}_{l,p}$ berechnet den Nachbaroperator für Elemente, die Daten von benachbarten Partitionen benötigen und $\mathrm{WP4}_{l,p}$ analog für von Kommunikation unabhängige Elemente. Wir maximieren überlappende Kommunikation und Berechnung indem wir die in Kommunikation involvierten Paketen $\mathrm{WP1}_{l,p}$ und $\mathrm{WP3}_{l,p}$ priorisieren. Sollte jedoch eines der Pakete noch auf eine Sende- oder Empfangsoperation warten, springen wir sofort zum nächsten Paket. Die von Kommunikation unabhängigen Arbeitspakete stellen wegen SeisSols kompakter Partitionierung mittels des dualen Graphens den rechenintensiven Teil dar. Hier berechnet der Rechenkern jeweils ein Paket und springt anschließend zurück zu den kommunikationsabhängigen Paketen. Unter den internen Arbeitspaketen $\mathrm{WP2}_{l,p}$ und $\mathrm{WP4}_{l,p}$ priorisieren wir nach der Zeitschrittgröße und nehmen an, dass kleine Zeitschritte den kritischen Pfad unserer Simulation darstellen.

## 5 Ergebnisse

Der präsentierte Rechenkern lief erfolgreich auf den kompletten Großrechnern SuperMUC Phase 1 (LRZ, Deutschland), SuperMUC Phase 2 (LRZ, Deutschland), Stampede (TACC, USA) und dem halben Tianhe-2 (NUDT, China). Auf allen Maschinen konnte petascale Leistung ($10^{15}$ Rechenoperationen pro Sekunde mit doppelter Genauigkeit) erreicht werden. Die Maximalleistung liegt hier bei 8.6 PFLOPS auf 8,192 Knoten von Tianhe-2 (1+ Mio. Kerne). Im Folgenden fokussieren wir uns auf zwei Simulationen, die auf den 3,072 Knoten (86,016 Kerne) von SuperMUC Phase 2 gelaufen sind.

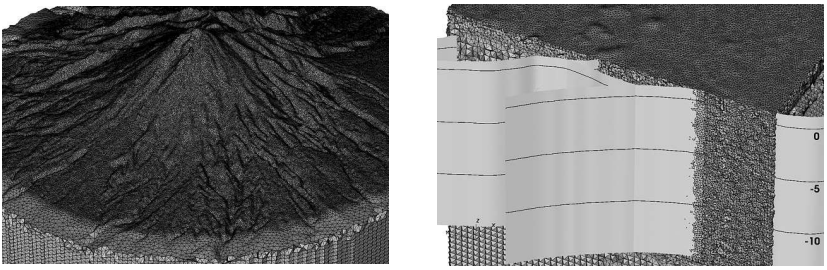



Abb. 6: Gitter der Mount Merapi und Landers Konfiguration.

Die erste Anwendung unseres Rechenkerns simuliert die Ausbreitung von seismischen Wellen im Vulkan **Mount Merapi**. Wir verwenden ein Gitter mit 99,831,401 Elementen und sechste Ordnung im ADER-DG-Verfahren (50,315,026,104 Freiheitsgrade). Das Gitter ist in der linken Seite von Abb. 6 gezeigt und die Seitenflächen sind anhand der Oberflächentopographie ausgerichtet. Wir verwenden unser gruppierte Zeitschrittverfahren mit einer Rate von $r = 2$ zwischen den Gruppen. Ausführender Höchstleistungsrechner ist die komplette zweite Phase von SuperMUC. Die Simulation erreichte die gewünschten 10 simulierten Sekunden nach einer Stunde und 6.6 Minuten. Im Schnitt (inkl. Setup und I/O) lief die Simulation mit 1.3 PFLOPS in Hardware, wobei 53 % der Operationen Nichtnulloperationen waren. Bezogen auf die Leistung der Phase 2 im HPL-Benchmark, dem Benchmark der TOP500-Liste[4], entspricht dies einer HPL-Effizienz von 46 % in Hardware.

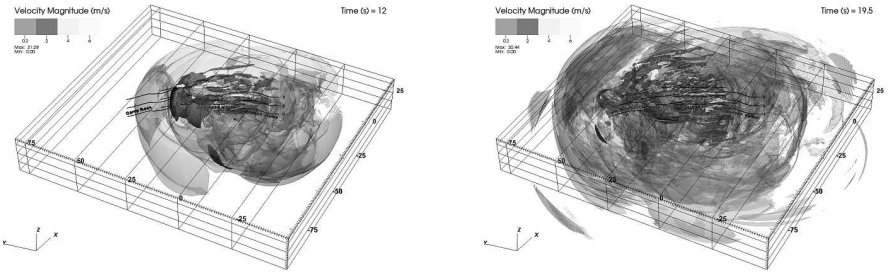



Abb. 7: Wellenausbreitung des auf 86,016 Kernen simulierten Landers Erdbebens von 1992 nach 12 s und 19.5 s. Quelle: [Br15b]

Unsere zweite Anwendung ist ein Produktionslauf, der das **Landers Erdbeben** von 1992 simuliert. Dieser Lauf verwendete ein Gitter mit 191,098,540 Elementen, sechste Ordnung (96,313,664,160 Freiheitsgrade) und den gleichen Zeitschritt in allen Elementen ($r = \infty$). Die Seitenflächen der Tetraeder sind anhand der Topographie und des Verwerfungssystems ausgerichnet. Abb. 6 zeigt auf der rechten Seite das Gitter mit dem eingebetteten Verwerfungssystem. Auf den Verwerfungen geben Konturlinien die Tiefe in Kilometern an (0 km, -5 km, -10 km). Da die Simulation in der Testphase von SuperMUC Phase 2 lief, musste sie mehrfach von Checkpoints neu gestartet werden. Der längste, ununterbrochene Teil lief für eine Stunde und 52 Minuten und schaltete die Simulation um 11.75 s vorwärts. Hier wurde eine durchschnittliche Leistung (inkl. I/O) von 1.4 PFLOPS in Hardware erzielt. Abb. 7 zeigt die komplexe seismische Wellenausbreitung dieser Simulation an verschiedenen Zeitpunkten.

## Literaturverzeichnis

[Br13] Breuer, Alexander et al.: Accelerating SeisSol by Generating Vectorized Code for Sparse Matrix Operators. In: Parallel Computing - Accelerating Computational Science and Engineering (CSE). Advances in Parallel Computing. IOS Press, 2013.

[4] `http://www.top500.org/`

[Br14] Breuer, Alexander et al.: Sustained Petascale Performance of Seismic Simulations with SeisSol on SuperMUC. In: Supercomputing. Lecture Notes in Computer Science. Springer, 2014.

[Br15a] Breuer, Alexander et al.: High-Order ADER-DG Minimizes Energy- and Time-to-Solution of SeisSol. In: High Performance Computing. Lecture Notes in Computer Science. Springer, 2015.

[Br15b] Breuer, Alexander Nikolas: High Performance Earthquake Simulations. Dissertation, Technische Universität München, München, 2015.

[Br16] Breuer, Alexander et al.: Petascale Local Time Stepping for the ADER-DG Finite Element Method. In: Parallel and Distributed Processing Symposium (IPDPS), to appear. 2016.

[Du06] Dumbser, Michael et al.: An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes - II. The three-dimensional isotropic case. Geophysical Journal International, 2006.

[Du07] Dumbser, Michael et al.: An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes - V. Local time stepping and p-adaptivity. Geophysical Journal International, 2007.

[Fe08] Federal Emergency Management Agency (FEMA): Hazus-MH Estimated Annualized Earthquake Losses for the United States, FEMA 366s. Mitigation Division, Washington, D.C, 2008.

[He14] Heinecke, Alexander et al.: Petascale High Order Dynamic Rupture Earthquake Simulations on Heterogeneous Supercomputers. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2014.

[He15] Heinecke, Alexander et al.: LIBXSMM: A High Performance Library for Small Matrix Multiplications. In: SC15, poster. 2015.

[Ja15] Jaiswal, Kishor S. et al.: Earthquake Shaking Hazard Estimates and Exposure Changes in the Conterminous United States. Earthquake Spectra, 31(S1):S201–S220, 2015.

[Pe12] Pelties, Christian et al.: Three-dimensional dynamic rupture simulation with a high-order discontinuous Galerkin method on unstructured tetrahedral meshes. Journal of Geophysical Research: Solid Earth, 2012.

[Pl07] Plesch, Andreas et al.: Community Fault Model (CFM) for Southern California. Bulletin of the Seismological Society of America, 2007.

[Pu09] de la Puente, Josep et al.: Dynamic Rupture Modeling on Unstructured Meshes Using a Discontinuous Galerkin Method. Journal of Geophysical Research: Solid Earth, 2009.

**Alexander Breuer** promovierte 2015 am Lehrstuhl für Wissenschaftliches Rechnen der Fakultät für Informatik der Technischen Universität München und ist seitdem an der University of California, San Diego. Alexanders Forschung fokussiert sich auf die Lösung hyperbolischer Differentialgleichungen und deckt Optimierungen in der kompletten Simulationspipeline ab. Dies beinhaltet Optimierungen auf dem Level eines Knoten, sowie homogene und heterogene Parallelisierung bis hin zum kompletten Höchstleistungsrechner.