Technische Universität Chemnitz-Zwickau

Mathias Meisel Arnd Meyer

Kommunikationstechnologien beim parallelen vorkonditionierten Schur-Komplement CG-Verfahren

Fakultät für Mathematik
TU Chemnitz-Zwickau
PSF 09107
D-09107 Chemnitz, FRG
(0371)-531-2657 (fax)

mathias.meisel@imech.tu-chemnitz.de
(0371)-531-4686

arnd.meyer@mathematik.tu-chemnitz.de
(0371)-531-2659

Preprint-Reihe der Chemnitzer DFG-Forschergruppe
"Scientific Parallel Computing"

SPC 95_19 **Juni 1995**

Diese Arbeit wurde gefördert durch die Deutsche Forschungsgemeinschaft (Vertrag Nr. Me1224).

Kommunikationstechnologien beim parallelen vorkonditionierten Schur-Komplement CG-Verfahren

Mathias Meisel   Arnd Meyer

19. Februar 1997

Abstract

Two alternative technologies of communication inside a parallelized Conjugate–Gradient algorithm are presented and compared to the well known hypercubecommunication. The amount of communication is diskussed in detail. A large range of numerical results corroborate the theoretical investigations.

Inhaltsverzeichnis

1 Einführung

Bei der Durchführung von Simulationsrechnungen zu Problemen der Festkörpermechanik , der Strömungsmechanik und auf vielen anderen Anwendungsgebieten der Mathematischen Physik sind häufig großdimensionale Gleichungssysteme $Kx = b$ mit schwach besetzter Systemmatrix K zu lösen. Sollen die physikalischen Gegebenheiten auch bei komplizierten Problemen mit hinreichender Genauigkeit erfaßt werden, müssen zur Diskretisierung (FEM , FDM , ...) oft Berechnungsgitter mit mehreren tausend Gitterpunkten benutzt werden. Dabei werden rasch die Grenzen der Leistungsfähigkeit herkömmlicher Computer erreicht. Zwar gestatten Betriebssysteme mit virtuellem Speicher wie UNIX , OS/2 und andere die Verarbeitung (zumindest theoretisch) beliebig großer Gleichungssysteme ohne daß das Auslagern und Einlesen von Matrixblöcken auf die bzw. von der Festplatte explizit vom Anwenderprogramm organisiert werden muß, jedoch werden die erforderlichen Rechenzeiten mit wachsender Systemdimension zunehmend unakzeptabel. Als Ausweg bietet sich hier der Einsatz von Mehrprozessorsystemen mit verteiltem Speicher an.

Ein natürlicher Parallelisierungszugang besteht darin, das Berechnungsgebiet Ω in Teilgebiete Ω_s zu zerlegen und diese Teilgebiete den zur Verfügung stehenden Prozessoren zuzuordnen ("Domain Decomposition", vgl. [2, 5]). Dies gestattet die Parallelisierung des gesamten Arithmetikaufwandes von der Netzgenerierung bis zur Lösung der Gleichungssysteme.

In [5] wurden eine parallele Realisierung des Schur–Komplement CG–Verfahrens ausführlich dargestellt und erste Erfahrungen mit diesem Solver innerhalb des FEM–Programmpakets FEAP ([3, 4]) bei der Lösung von Problemen der Festkörpermechanik präsentiert. Dabei war festzustellen, daß die Gesamteffizienz des Lösungsverfahrens mit wachsender Prozessoranzahl zunehmend durch die Effizienz der in jedem Iterationsschritt erforderlichen globalen Kommunikation zwischen den Prozessoren bestimmt wird.

Ziel dieses Artikels ist es, zwei gegenüber der den Rechenzeitangaben in [5] zugrunde liegenden Hypercubekommunikation verbesserte Kommunikationstechnologien vorzustellen und deren Zeitverhalten in verschiedenen Anwendungssituationen miteinander zu vergleichen.

Auf die parallele Netzgenerierung und FEM–Assemblierung soll hier nicht eingegangen werden. Hierfür sei auf [4] verwiesen.

In **Abschnitt 2** wird das Iterationsverfahren beschrieben, innerhalb dessen die Kommunikationsmethoden betrachtet werden.

Abschnitt 3 ist der Betrachtung der drei Kommunikationsmethoden gewidmet. Im abschließenden **Abschnitt 4** werden die Ergebnisse von Testrechnungen präsentiert.

2 Eine Version des parallelen CG–Verfahrens

In diesem Abschnitt sollen einige Bezeichnungen eingeführt und das benutzte Iterationsverfahren kurz beschrieben werden.

Der Einfachheit halber sei das Berechnungsgebiet Ω für die Simulationsrechnung auf $p = 2^\mu$ Prozessoren (μ-dimensionaler Hypercube) in p Teilgebiete Ω_s zerlegt ($\Omega_s \cap \Omega_t = \emptyset$ für $s \neq t$; $0 \leq s, t \leq p-1$) und jedem der Prozessoren $\mathcal{P}_s$ sei genau eines dieser Teilgebiete zugeordnet. Bei der Netzgenerierung seien die Gitterpunkte auf den Teilgebietsrändern $\Gamma_s := \partial\Omega_s$ so erzeugt worden, daß diese (lokalen) Randgitterpunkte hinsichtlich ihrer Anzahl und ihrer Koordinatenwerte auf den Koppelrändern $\Gamma_{st} := \overline{\Omega}_s \cap \overline{\Omega}_t$ in jeweils beiden Prozessoren $\mathcal{P}_s$ und $\mathcal{P}_t$ übereinstimmen (s. [5]). Ohne Beschränkung der Allgemeinheit seien die Gitterpunkte in $\overline{\Omega}_s$ lokal so numeriert, daß die auf Γ_s liegenden Punkte die Nummern 1 bis m_s^C und die m_s^I im Inneren von Ω_s liegenden Punkte die Nummern m_s^C+1 bis $m_s^C + m_s^I =: m_s$ erhalten. Dieser Numerierung folgend seien $n_s := n_s^I + n_s^C$ die Zahl der Freiheitsgrade in $\overline{\Omega}_s$, n die globale Anzahl der Freiheitsgrade im zu lösenden diskreten Problem, $n^I := \sum_{s=0}^{p-1} n_s^I$ die Gesamtzahl aller im Inneren der Teilgebiete Ω_s liegenden Freiheitsgrade und $n^C := n - n^I$ die Anzahl aller Koppel- bzw. Randfreiheitsgrade.

2.1 Verteilt gespeicherte Matrizen und Vektoren

Bei der FEM–Assemblierung wird die Steifigkeitsmatrix K in summarisch verteilter Speicherform erhalten:

$$K = \sum_{s=0}^{p-1} A_s^T K_s A_s \ .$$

Hier ist A_s eine Boolsche Matrix, die den Zusammenhang zwischen der lokalen Numerierung der Freiheitsgrade im Subdomain $\overline{\Omega}_s$ und der globalen Numerierung in $\overline{\Omega}$ herstellt.

Die Unterscheidung in Koppel- und innere Freiheitsgrade impliziert eine Strukturierung der entsprechenden Vektoren:

$$x = \begin{pmatrix} x^C \\ x^I \end{pmatrix} \quad , \quad x^I = \begin{pmatrix} x_0^I \\ \vdots \\ x_{p-1}^I \end{pmatrix} \ .$$

Dabei bezeichnet x_s^I den Vektor der im Inneren des Subdomains Ω_s liegenden Freiheitsgrade. Diese Struktur überträgt sich auch auf die Matrizen K, K_s und A_s:

$$\begin{gathered} K = \begin{pmatrix} K^C & K^{CI} \\ K^{IC} & K^I \end{pmatrix} \quad , \quad K^I = \operatorname{diag}\left(K_0^I \quad \ldots \quad K_{p-1}^I \right) \\ K^{IC} = \left((K_s^{IC} A_s^C) \right)_{s=0}^{p-1} \\ K^C = \sum_{s=0}^{p-1} (A_s^C)^T K_s^C A_s^C \quad , \quad K^{CI} = \sum_{s=0}^{p-1} (A_s^C)^T K_s^{CI} A_s^I \\ K_s = \begin{pmatrix} K_s^C & K_s^{CI} \\ K_s^{IC} & K_s^I \end{pmatrix} \quad , \quad A_s = \begin{pmatrix} A_s^C & \mathbf{0} \\ \mathbf{0} & A_s^I \end{pmatrix} . \end{gathered} \tag{2.1}$$

Für globale Vektoren werden zwei unterschiedliche Speicherformen betrachtet, die sich nur in der Behandlung der Koppelfreiheitsgrade unterscheiden. Durch

$$x^C = \sum_{s=0}^{p-1} (A_s^C)^T x_s^C \tag{2.2}$$

wird diejenige Speicherform beschrieben, bei der sich die Werte der Vektorfunktion x^C in den Koppelknoten durch die FEM-Assemblierung ihrer lokalen Anteile x_s^C ergeben (summarisch verteilte Speicherung). Andererseits beschreibt

$$x_s^C = A_s^C x^C \tag{2.3}$$

die Speicherart, bei der in $\mathcal{P}_s$ derjenige Teil von x^C gespeichert ist, der in $\overline{\Omega}_s$ liegenden Freiheitsgraden entspricht (kopienverteilte Speicherung).

Die im CG–Algorithmus (2.6) auftretenden Vektoren r, u und die rechte Seite b sind als nach (2.2) summarisch verteilt und die Vektoren w, q, ξ, g und die Lösung x als nach (2.3) kopienverteilt gespeichert zu betrachten.

2.2 Schur–Komplement CG

Mit der Schur-Komplement-Matrix

$$S = K^C - K^{CI}(K^I)^{-1}K^{IC}$$

ist die Blockfaktorisierung der Steifigkeitsmatrix

$$K = \begin{pmatrix} I & K^{CI}(K^I)^{-1} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} S & \mathbf{0} \\ \mathbf{0} & K^I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ (K^I)^{-1}K^{IC} & I \end{pmatrix}$$

Ausgangspunkt für die Konstruktion der Vorkonditionierungsmatrix

$$C = \begin{pmatrix} I & K^{CI}(K^I)^{-1} \\ \mathbf{0} & I \end{pmatrix} \begin{pmatrix} V^C & \mathbf{0} \\ \mathbf{0} & K^I \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ (K^I)^{-1}K^{IC} & I \end{pmatrix} ,$$

wobei für den auf den Koppelfreiheitsgraden wirkenden Vorkonditionierer für das Schur–Komplement S eine Blockdiagonalmatrix V^C mit den Eigenschaften (2.4) gewählt wurde (näheres hierzu siehe [5]):

$$V_s^C = A_s^C V^C (A_s^C)^T \qquad , \qquad (V_s^C)^{-1} = A_s^C (V^C)^{-1} (A_s^C)^T \, . \tag{2.4}$$

Für die Matrizen K_s^I von (2.1) sei durch

$$L_s^I \, (L_s^I)^T = K_s^I \tag{2.5}$$

eine Choleskyzerlegung definiert.

Die im Programmpaket FEAP implementierte Version des parallelisierten Schur–Komplement CG–Verfahrens wird aus der Sicht eines Prozessors $\mathcal{P}_s$ beschrieben. Eine ausführliche Darstellung und Herleitung des Formelsatzes findet sich in [5].

Zur Durchführung des Iterationsverfahrens mit dem Startvektor $x^{(0)}$ sind in jedem Prozessor $\mathcal{P}_s$ mit den Startwerten $\alpha^{(0)}\!:=1$, $\beta^{(0)}\!:=0$, $\xi_s^{(0)C}\!:=\mathbf{0}$, $u_s^{(0)C}\!:=\mathbf{0}$ und $i\!:=0$ die folgenden Operationen auszuführen:

$$
\begin{array}{rrcl}
(1) & r_s^{(0)I} & := & b_s^I - K_s^{IC} x_s^{(0)C} - K_s^I x_s^{(0)I} \\
(2) & & L_s^I & \text{nach (2.5)bestimmen} \\
(3) & \text{löse} & & L_s^I (L_s^I)^T h_s^{(0)I} = r_s^{(0)I} \\
(4) & x_s^{(1)I} & := & x_s^{(0)I} + h_s^{(0)I} \\
(5) & r_s^{(0)C} & := & K_s^C x_s^{(0)C} + K_s^{CI} x_s^{(1)I} - b_s^C \\
(6) & w_s^{(0)} & : & \text{wie in den Schritten (15)–(18) bestimmen} \\
(7) & q_s^{(0)C} & := & w_s^{(0)C} \\
(8) & h_s^{(0)C} & := & K_s^C w_s^{(0)C} + K_s^{CI} w_s^{(0)I} \\
(9) & \gamma_s^{(0)} & := & (w_s^{(0)C}, r_s^{(0)C}) \quad , \quad \tau_s^{(0)} := (w_s^{(0)C}, h_s^{(0)C}) \\
(10) & \gamma^{(0)} & := & \sum_{s=0}^{p-1} \gamma_s^{(0)} \quad , \quad \tau^{(0)} := \sum_{s=0}^{p-1} \tau_s^{(0)} \qquad \Longleftarrow \underline{\text{Kommunikation}} \\
\hline
(11) & u_s^{(i+1)C} & := & h_s^{(i)C} + \beta^{(i)} u_s^{(i)C} \\
(12) & \alpha^{(i+1)} & := & - \left[\frac{\tau^{(i)}}{\gamma^{(i)}} + \frac{\beta^{(i)}}{\alpha^{(i)}} \right]^{-1} \\
(13) & \xi_s^{(i+1)C} & := & \xi_s^{(i)C} + \alpha^{(i+1)} q_s^{(i)C} \\
(14) & r_s^{(i+1)C} & := & r_s^{(i)C} + \alpha^{(i+1)} u_s^{(i+1)C} \\
(15) & g_s^{(i+1)C} & := & r_s^{(i+1)C} + \sum_{t \neq s} A_s^C (A_t^C)^T r_t^{(i+1)C} \qquad \Longleftarrow \underline{\text{Kommunikation}} \qquad (2.6) \\
(16) & \text{löse} & & V_s^C w_s^{(i+1)C} = g_s^{(i+1)C} \\
(17) & h_s^{(i+1)I} & := & -K_s^{IC} w_s^{(i+1)C} \\
(18) & \text{löse} & & L_s^I (L_s^I)^T w_s^{(i+1)I} = h_s^{(i+1)I} \\
(19) & h_s^{(i+1)C} & := & K_s^C w_s^{(i+1)C} + K_s^{CI} w_s^{(i+1)I} \\
(20) & \gamma_s^{(i+1)} & := & (w_s^{(i+1)C}, r_s^{(i+1)C}) \quad , \quad \tau_s^{(i+1)} := (w_s^{(i+1)C}, h_s^{(i+1)C}) \\
(21) & \gamma^{(i+1)} & := & \sum_{s=0}^{p-1} \gamma_s^{(i+1)} \quad , \quad \tau^{(i+1)} := \sum_{s=0}^{p-1} \tau_s^{(i+1)} \qquad \Longleftarrow \underline{\text{Kommunikation}} \\
(22) & \beta^{(i+1)} & := & \gamma^{(i+1)} / \gamma^{(i)} \\
(23) & q_s^{(i+1)C} & := & w_s^{(i+1)C} + \beta^{(i+1)} q_s^{(i)C} \\
 & & & i := i+1 \quad \Longrightarrow (11) \quad \text{oder} \quad \Longrightarrow (24) \\
\hline
(24) & x_s^{(i)C} & := & x_s^{(0)C} + \xi_s^{(i)C} \\
(25) & h_s^{(i)I} & := & K_s^{IC} \xi_s^{(i)C} \\
(26) & \text{löse} & & L_s^I (L_s^I)^T \xi_s^{(i)I} = h_s^{(i)I} \\
(27) & x_s^{(i)I} & := & x_s^{(1)I} + \xi_s^{(i)I} \\
 & & & \text{STOP}
\end{array}
$$

Bemerkung 2.1

Speichertechnisch kann im Verfahren (2.6) $\xi_s^I \equiv w_s^I \equiv r_s^I \equiv h_s^I$ *gesetzt werden. Da die Matrizen* K_s^I *in Schritt (2) choleskyzerlegt wurden, erfolgt die Lösung der Gleichungssysteme mit der Systemmatrix* $L_s^I(L_s^I)^T$ *in den Schritten (3), (18) und (26) durch einfaches Vorwärts– und anschließendes Rückwärtseinsetzen.*
Bei der Bildung der globalen Summen in den Schritten (10) und (21) wird statt zweier globaler Kommunikationen mit jeweils einer Zahl je Prozessor (γ *oder* τ*) eine globale Kommunikation mit zwei Zahlen je Prozessor ausgeführt. In jedem Prozessor liegt ein Paar reeller Zahlen (*γ_s *und* τ_s*) vor, aus denen jeweils die globale Summe (*$\sum_{s=0}^{p-1} \gamma_s$ *und* $\sum_{s=0}^{p-1} \tau_s$*) berechnet werden soll. Das Ergebnis wird in jedem Prozessor benötigt.*
Hierzu wird die in [1] dokumentierte Routine cube_dod benutzt.

3 Kommunikationstechnologien

Im Algorithmus (2.6) ist in Schritt (15) die FEM–Assemblierung eines globalen Vektors aus seinen lokalen Anteilen zu bilden:

$$g_s^C := r_s^C + \sum_{t \neq s} A_s^C (A_t^C)^T r_t^C \ . \tag{3.1}$$

Vom Ergebnis wird in jedem Prozessor nur der Teil gebraucht, der zu in $\overline{\Omega}_s$ liegenden Gitterpunkten gehört. Drei Möglichkeiten zur Realisierung dieser Aufgabenstellung sollen in diesem Abschnitt betrachtet werden.

3.1 Hypercubekommunikation

Die wohl unkomplizierteste Art der FEM–Assemblierung (3.1) eines Vektors r^C des Speichertyps (2.2) zu einem Vektor g^C des Speichertyps (2.3) besteht darin, daß jeder Prozessor $\mathcal{P}_s$ seinen Anteil r_s^C an alle anderen Prozessoren $\mathcal{P}_t$ $(s \neq t)$ sendet und anschließend jeder Prozessor aus allen so empfangenen Daten die zu Punkten des Koppelrandes seines Subdomains Ω_s gehörenden Werte heraussortiert und zu seinen eigenen Werten in diesen Punkten addiert.

Zum Versenden der Daten eignet sich gut die Routine cube_cat aus [1], die bei $p = 2^\mu$ Prozessoren in 2μ send–/receive–Operationen in jedem Prozessor einen Hilfsvektor H der Länge $l^C := \sum_{s=0}^{p-1} n_s^C$ fortlaufend mit den Daten aller Prozessoren füllt.

Der Assemblierungsschritt kann dabei so organisiert werden, daß in einem die Assemblierung vorbereitenden Schritt in jedem Prozessor $\mathcal{P}_s$ ein Integer–Vektor der Länge n_s^C mit global eindeutigen Nummern der Freiheitsgrade entsprechend der lokalen Numerierung gefüllt und dieser dann ebenfalls mittels cube_cat an alle anderen Prozessoren verschickt und in einem Hilfsvektor IH abgespeichert wird. Hieraus kann dann jeder Prozessor ermitteln, ob ein bestimmter Freiheitsgrad zu seinem Subdomain gehört und wenn ja, welche lokale Nummer er hat.

Als Nachteil dieser Technologie erweist sich, daß der Kommunikationsaufwand und die Länge der erforderlichen Hilfsvektoren H und IH mit der Zahl der Prozessoren gravierend anwächst. Zur Illustration dessen wird im folgenden die Arbeitsweise von cube_cat näher betrachtet:

Im Falle zweier Prozessoren ($\mu = 1$) sendet zuerst Prozessor $\mathcal{P}_0$ seine n_0^C Komponenten r_0^C an $\mathcal{P}_1$ (1.Takt). Hat $\mathcal{P}_1$ diese empfangen, schickt er seine n_1^C Komponenten r_1^C an $\mathcal{P}_0$ (2.Takt).

Der Gesamtkommunikationsaufwand[1] ist $\mathcal{O}(n_0^C + n_1^C) = \mathcal{O}(l^C)$, zuzüglich zwei Setup–Zeiten.

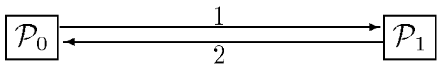

Abbildung 1: Hypercube der Dimension 1

Bei vier Prozessoren ($\mu = 2$) kommunizieren zuerst die beiden Prozessorpaare $\mathcal{P}_0$, $\mathcal{P}_1$ und $\mathcal{P}_2$,$\mathcal{P}_3$ jeweils wie im Fall $\mu = 1$ miteinander (Takte 1 und 2). Danach schicken gleichzeitig $\mathcal{P}_0$ die $n_0^C + n_1^C$ Komponenten (r_0^C, r_1^C) an $\mathcal{P}_2$ und $\mathcal{P}_1$ die $n_1^C + n_0^C$ Komponenten (r_1^C, r_0^C) an $\mathcal{P}_3$ (3.Takt). Abschließend senden gleichzeitig $\mathcal{P}_2$ die $n_2^C + n_3^C$ Komponenten (r_2^C, r_3^C) an $\mathcal{P}_0$ und $\mathcal{P}_3$ die $n_3^C + n_2^C$ Komponenten (r_3^C, r_2^C) an $\mathcal{P}_1$ (4.Takt).

Der Gesamtkommunikationsaufwand ist $\mathcal{O}(\max\{n_0^C + n_1^C, n_2^C + n_3^C\} + l^C)$ bzw. $\mathcal{O}(3l^C/2)$, wenn $n_0^C = n_1^C = n_2^C = n_3^C$. Hinzu kommen 4 Setup–Zeiten.

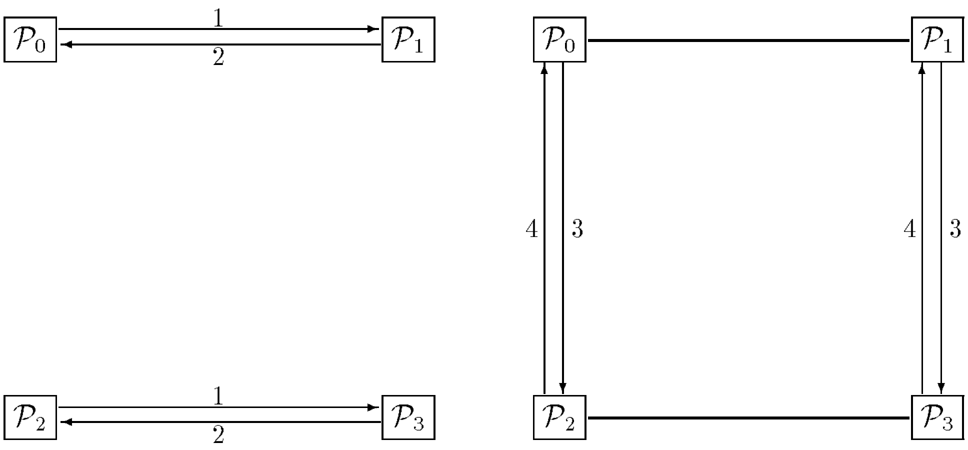

Abbildung 2: Hypercube der Dimension 2

Werden 8 Prozessoren benutzt ($\mu = 3$), finden zuerst innerhalb der Prozessorgruppen $\mathcal{P}_0$ bis $\mathcal{P}_3$ bzw. $\mathcal{P}_4$ bis $\mathcal{P}_7$ gleichzeitig all die Kommunikationen statt, die für den Hypercube der Dimension 2 bereits beschrieben wurden. Anschließend kommunizieren diese beiden Subhypercubes in zwei weiteren Takten miteinander:

5.Takt: $\mathcal{P}_0$ sendet $(r_0^C, r_1^C, r_2^C, r_3^C)$ an $\mathcal{P}_4$ $\quad$ $\mathcal{P}_1$ sendet $(r_1^C, r_0^C, r_3^C, r_2^C)$ an $\mathcal{P}_5$
$\mathcal{P}_2$ sendet $(r_2^C, r_3^C, r_0^C, r_1^C)$ an $\mathcal{P}_6$ $\quad$ $\mathcal{P}_3$ sendet $(r_3^C, r_2^C, r_1^C, r_0^C)$ an $\mathcal{P}_7$

[1]Hier und im weiteren soll darunter der Zeitaufwand für den Transport der Daten verstanden werden.

6.Takt: $\mathcal{P}_4$ sendet $(r_4^C, r_5^C, r_6^C, r_7^C)$ an $\mathcal{P}_0$ $\quad$ $\mathcal{P}_5$ sendet $(r_5^C, r_4^C, r_7^C, r_6^C)$ an $\mathcal{P}_1$
$\mathcal{P}_6$ sendet $(r_4^C, r_7^C, r_4^C, r_5^C)$ an $\mathcal{P}_2$ $\quad$ $\mathcal{P}_7$ sendet $(r_7^C, r_6^C, r_5^C, r_4^C)$ an $\mathcal{P}_3$
Gesamtkommunikationsaufwand: $\mathcal{O}(\max\{n_0^C + n_1^C, n_2^C + n_3^C, n_4^C + n_5^C, n_6^C + n_7^C\} + \max\{n_0^C + n_1^C + n_2^C + n_3^C, n_4^C + n_5^C + n_6^C + n_7^C\} + l^C)$ bzw. $\mathcal{O}(7l^C/4)$, wenn $n_0^C = \ldots = n_7^C$.
Hinzu kommen 6 Setup–Zeiten.

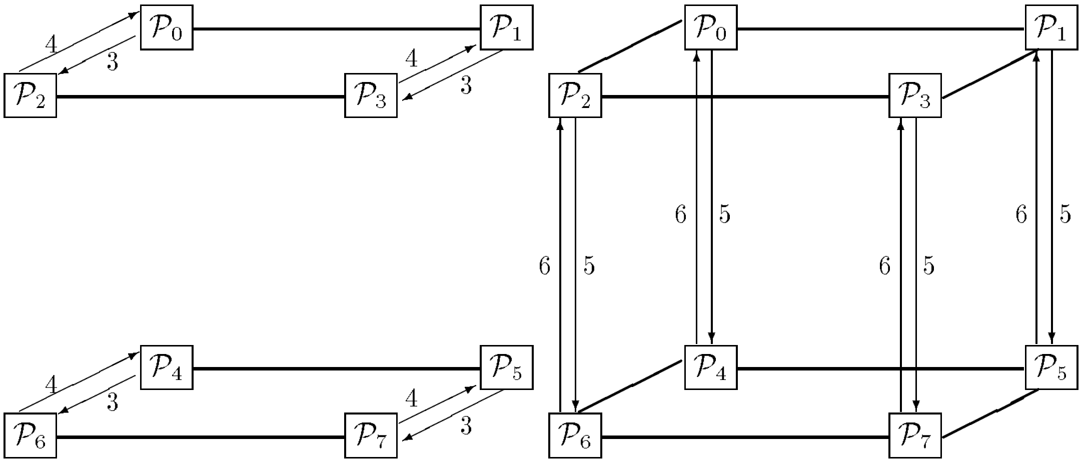

Abbildung 3: Hypercube der Dimension 3

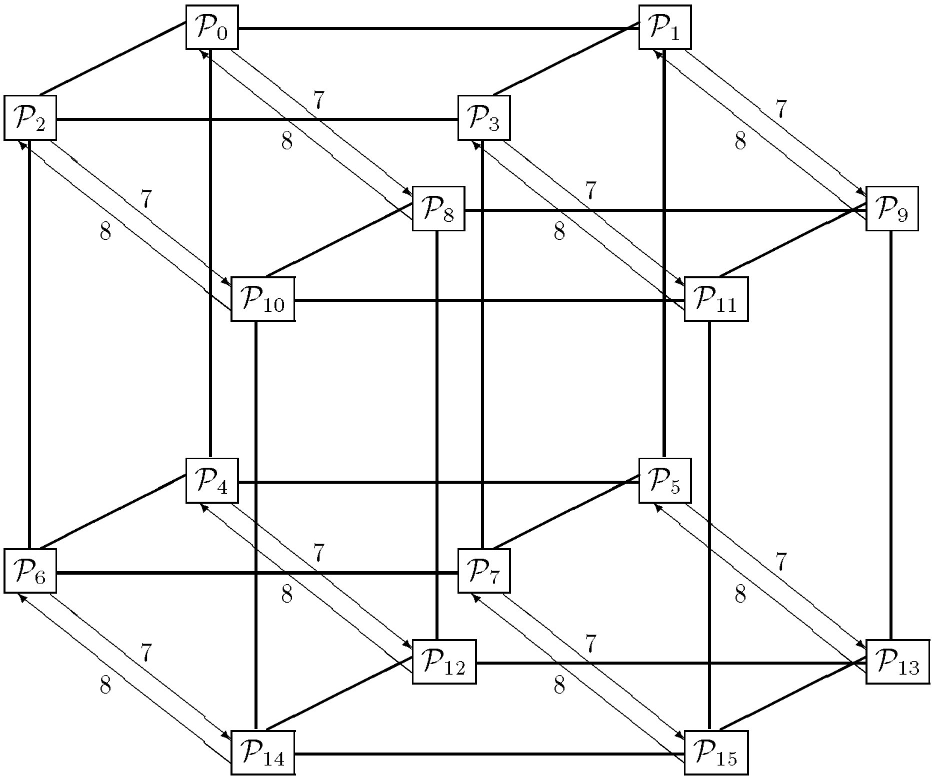

Abbildung 4: Hypercube der Dimension 4

Analog dazu werden bei einer cube_cat–Operation mit 16 Prozessoren ($\mu = 4$)

zunächst alle erforderlichen Kommunikationen in den beiden dreidimensionalen Hypercubes der Prozessoren $\mathcal{P}_0$ bis $\mathcal{P}_7$ bzw. $\mathcal{P}_8$ bis$\mathcal{P}_{15}$ ausgeführt, und in den beiden abschließenden Takten tauschen diese Subhypercubes der Dimension 3 ihre Daten aus. Wenn $n_0^C = \ldots = n_{15}^C$ (wenigstens näherungsweise) gilt, ist der Gesamtkommunikationsaufwand von der Ordnung $\mathcal{O}(15l^C/8)$. Hinzu kommen 8 Setup–Zeiten.

Allgemein ist der Kommunikationsaufwand für eine cube_cat–Operation im Hypercube der Dimension μ mit $p = 2^\mu$ Prozessoren bei näherungsweise gleichvielen Koppelfreiheitsgraden in jedem Prozessor von der Ordnung

$$\mathcal{O}\left(\frac{2(p-1)}{p}\ l^C\right) \approx \mathcal{O}\left(2\ (p-1)\ n_*^C\right) \qquad \text{zuzüglich} \quad 2\mu \quad \text{Setup–Zeiten,} \qquad (3.2)$$

wobei n_*^C die (durchschnittliche) Zahl der Koppelfreiheitsgrade in einem Prozessor bezeichnet.

3.2 Einsparungspotential beim Kommunikationsaufwand

Bereits in [5] wurde vermerkt, daß ein mit wachsender Prozessorzahl immer größer werdender Anteil des im vorangegangenen Abschnitt analysierten Kommunikationsaufwandes eigentlich überflüssig ist. Zur Illustration dessen werden folgendes streifenförmige Gebiet (etwa ein Stab oder Balken),

Ω_0	Ω_1	Ω_2	Ω_3	Ω_4	Ω_5	Ω_6	Ω_7

Ω_0	Ω_1	Ω_2	Ω_3	Ω_4	Ω_5	Ω_6	Ω_7	Ω_8	Ω_9	Ω_{10}	Ω_{11}	Ω_{12}	Ω_{13}	Ω_{14}	Ω_{15}

Abbildung 5: Streifengebiet auf 8 bzw. 16 Prozessoren

und das schon in [5] verwendete keilförmige Gebiet betrachtet:

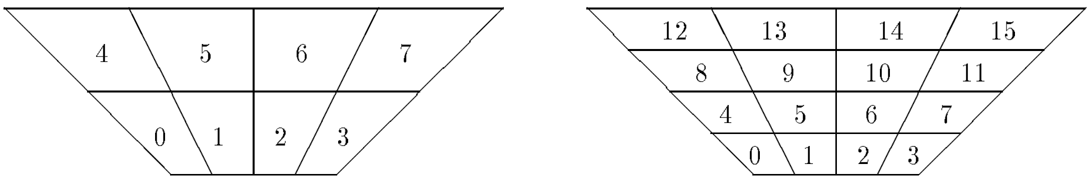

Abbildung 6: Keilgebiet auf 8 bzw. 16 Prozessoren

Für die Realisierung von (3.1) ist es offenbar ausreichend, wenn jeder Prozessor $\mathcal{P}_s$ mit genau denjenigen Prozessoren $\mathcal{P}_t$ Daten austauscht, deren Subdomain $\overline{\Omega}_t$ mit $\overline{\Omega}_s$ wenigstens einen Punkt gemeinsam hat.

Im Fall des Streifengebiets hat kein Prozessor mehr als 2 Nachbarn. Deshalb sind theoretisch 4 (bei zwei Prozessoren 2) Takte für eine globale Kommunikation ausreichend, und der damit verbundene Kommunikationsaufwand ist unabhängig von der Prozessoranzahl[2]

$$\mathcal{O}\left(2\, n_*^C\right) \qquad \text{zuzüglich} \quad 4 \quad \text{Setup–Zeiten.} \tag{3.3}$$

Dies entspricht bei 4, 8, 16, 32, 64 bzw. 128 Prozessoren einer Reduzierung auf 33%, 14%, 7%, 3%, 1.6% bzw. 0.8% des Hypercubekommunikationsaufwands (3.2). Im Fall des Keilgebietes ist der unumgängliche Kommunikationsaufwand von der Ordnung $\mathcal{O}\left(2\, n_*^C + 4\, n_\bullet^C\right)$ (bzw. $\mathcal{O}(n_*^C)$ bei $p=2$), wobei $n_\bullet^C$ die maximale Anzahl der Crosspointfreiheitsgrade eines Prozessors sei, jedoch ist der Anteil der Koppelfreiheitsgrade an der Gesamtzahl aller Freiheitsgrade größer als beim Streifengebiet, und aufgrund der größeren Zahl benachbarter Subdomains werden 16 (bei $p \geq 16$), 10 (bei $p=8$), 6 (bei $p=4$) bzw. 2 (bei $p=2$) Setup–Zeiten benötigt,

Andererseits ist die Hypercubekommunikationstechnologie weitestgehend universell: Weder an die Geometrie des Berechnungsgebiets Ω noch an dessen Zerlegung in die Teilgebiete Ω_s müssen besondere Anforderungen gestellt werden.

Für die in den beiden folgenden Abschnitten vorzustellenden Kommunikationstechnologien ist dagegen wenigstens vorauszusetzen, daß je zwei nichtdisjunkte Teilgebiete Ω_s und Ω_t $(s \neq t)$ entweder genau einen gemeinsamen Punkt (Crosspoint) oder eine ganze gemeinsame Kante (Koppelkante) besitzen. Unter dieser Voraussetzung können die jeweiligen virtuellen Prozessortopologien korrekt definiert werden.

Ferner ist bei einer direkten Kommunikation zwischen geometrisch benachbarten Prozessoren die Reihenfolge der Kommunikationsschritte in jedem Prozessor so festzulegen, daß der Prozessor $\mathcal{P}_s$ zu möglichst genau dem Zeitpunkt Daten von $\mathcal{P}_t$ empfangen will, zu dem $\mathcal{P}_t$ Daten an $\mathcal{P}_s$ schickt, weil sonst wegen der synchronen Kommunikationsweise unnötige Wartezeiten entstehen oder (im Extremfall) das Programm völlig zum Stehen kommt, weil alle Prozessoren im Wartezustand sind.[3] Die möglichen Einsparungen bei den Kommunikationszeiten erfordern also einen größeren Organisationsaufwand, der jedoch als einmaliger Aufwand vor dem Start des Iterationsverfahrens von geringerer Bedeutung ist als der in jedem Iterationsschritt erforderliche Kommunikationsaufwand.

3.3 Koppelkantenkommunikation

Die dem Inneren der Koppelkanten zuzuordnenden Daten werden jeweils nur von den beiden Prozessoren benötigt, deren Subdomains längs dieser Kante aneinandergrenzen, während die Crosspointdaten i.allg. mit mehreren Prozessoren, deren Anzahl von den geometrischen Verhältnissen der Partitionierung $\Omega = \cup_{s=0}^{p-1}\ \Omega_s$ abhängt, ausgetauscht werden müssen. Ein erster Schritt zur Reduzierung des Kommunikationsaufwandes kann demnach darin bestehen, nur die Crosspointdaten mittels Hypercubekommunikation auszutauschen und den Datenaustausch für die übrigen Koppelfreiheitsgrade auf direktem Wege zu organisieren. Dazu sind neben der virtuellen Prozessortopologie des Hypercubes weitere virtuelle Verbindungen ("links")

[2] bei 2 Prozessoren lediglich $\mathcal{O}(n_*^C)$ zzgl. 2 Setup–Zeiten.

[3] Dies ist beispielsweise dann der Fall, wenn alle Prozessoren gleichzeitig Daten senden wollen, aber kein Prozessor Daten empfangen will, oder wenn im Fall des Keilgebietes aus Abbildung 6 bei 8 Prozessoren $\mathcal{P}_0$ zuerst mit $\mathcal{P}_1$, $\mathcal{P}_1$ zuerst mit $\mathcal{P}_5$, $\mathcal{P}_5$ aber zuerst mit $\mathcal{P}_0$ kommunizieren will.

zwischen den geometrisch benachbarten Prozessoren zu definieren, über die der Austausch der Koppeldaten erfolgen kann.
Um die Wartezeiten bei der Koppelrandkommunikation möglichst gering zu halten, werden die Koppelränder aller Subdomains nach folgendem Algorithmus lokal neu numeriert, wobei die neuen Koppelkantennummern möglichst in beiden längs der jeweiligen Koppelkante benachbarten Subdomains übereinstimmen und so klein wie möglich sein sollen:

$\forall$ Subdomains $\Omega_s \qquad (s = 0, 1, \ldots)$:

$\forall$ Koppelkanten Γ_s^i von $\Omega_s \qquad (i = 1, 2, \ldots)$:

Sei Ω_t das Nachbarteilgebiet längs der i–ten Koppelkante.

Sei j die kleinste in Ω_s noch nicht vergebene Koppelkantennummer.

Wenn die Koppelkantennummer j in Ω_t noch nicht vergeben ist, dann k:=j. Weiter bei (1) .

Sei j die kleinste Koppelkantennummer, die weder in Ω_s noch in Ω_t bereits vergeben wurde.
Existiert ein solches j, dann k:=j. Weiter bei (1) .

Sei j die kleinste in Ω_s noch nicht vergebene Koppelkantennummer und k die kleinste in Ω_t noch nicht vergebene Koppelkantennummer.

(1) Die ehemals i–te Koppelkante in Ω_s erhält die neue Nummer j.
Die gleiche Koppelkante erhält in Ω_t die neue Nummer k.

Indem für jedes Koppelrandstück in beiden Prozessoren möglichst die gleiche Nummer festgelegt wird, werden die Wartezeiten innerhalb der Koppelrandkommunikation minimiert, wenn die Prozessoren in dieser Reihenfolge ihre Daten in den jeweiligen Koppelpunkten mit denen des entsprechenden Nachbarprozessors austauschen.

Für das Streifengebiet (Abb. 5) auf 8 Prozessoren liefert der Algorithmus die in Abbildung 7 dargestellte Numerierung:

1	1 2	2 1	1 2	2 1	1 2	2 1	1

Abbildung 7: Koppelkantennumerierung beim Streifengebiet

Dabei sind insgesamt 4 Kommunikationstakte für den Austausch der Koppelranddaten erforderlich. In den ersten beiden Takten erfolgt die Kommunikation über die mit "1" markierten Kanten und in den Takten 3 und 4 wird über die zweite Kantengruppe kommuniziert.

Als weiteres Beispiel zur Illustration der Koppelkantennumerierung wird in Abbildung 8 das Keilgebiet auf 16 Prozessoren betrachtet:

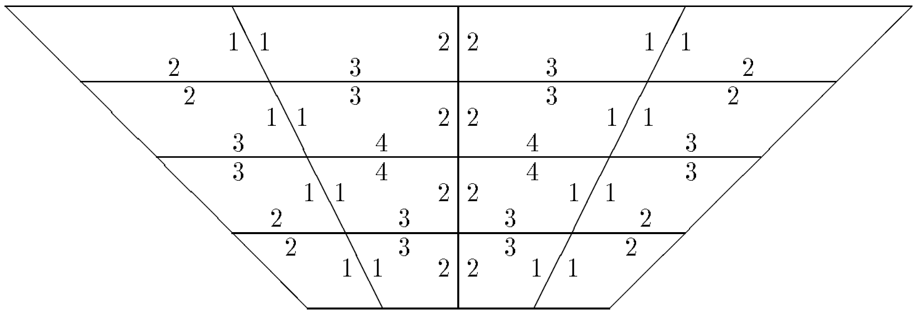

Abbildung 8: Koppelkantennumerierung beim Keilgebiet

Die Numerierung der Subdomains aus Abbildung 6 zugrundelegend ergibt sich folgende Reihenfolge der Kommunikationsaktionen:

$$
\begin{array}{lllll}
\underline{1.Takt:} & \mathcal{P}_0 \rightarrow \mathcal{P}_1 & \mathcal{P}_2 \rightarrow \mathcal{P}_3 & \mathcal{P}_4 \rightarrow \mathcal{P}_5 & \mathcal{P}_6 \rightarrow \mathcal{P}_7 \\
 & \mathcal{P}_8 \rightarrow \mathcal{P}_9 & \mathcal{P}_{10} \rightarrow \mathcal{P}_{11} & \mathcal{P}_{12} \rightarrow \mathcal{P}_{13} & \mathcal{P}_{14} \rightarrow \mathcal{P}_{15} \\
\underline{2.Takt:} & \mathcal{P}_0 \leftarrow \mathcal{P}_1 & \mathcal{P}_2 \leftarrow \mathcal{P}_3 & \mathcal{P}_4 \leftarrow \mathcal{P}_5 & \mathcal{P}_6 \leftarrow \mathcal{P}_7 \\
 & \mathcal{P}_8 \leftarrow \mathcal{P}_9 & \mathcal{P}_{10} \leftarrow \mathcal{P}_{11} & \mathcal{P}_{12} \leftarrow \mathcal{P}_{13} & \mathcal{P}_{14} \leftarrow \mathcal{P}_{15} \\
\underline{3.Takt:} & \mathcal{P}_0 \rightarrow \mathcal{P}_4 & \mathcal{P}_1 \rightarrow \mathcal{P}_2 & \mathcal{P}_3 \rightarrow \mathcal{P}_7 & \mathcal{P}_5 \rightarrow \mathcal{P}_6 \\
 & \mathcal{P}_8 \rightarrow \mathcal{P}_{12} & \mathcal{P}_9 \rightarrow \mathcal{P}_{10} & \mathcal{P}_{11} \rightarrow \mathcal{P}_{15} & \mathcal{P}_{13} \rightarrow \mathcal{P}_{14} \\
\underline{4.Takt:} & \mathcal{P}_0 \leftarrow \mathcal{P}_4 & \mathcal{P}_1 \leftarrow \mathcal{P}_2 & \mathcal{P}_3 \leftarrow \mathcal{P}_7 & \mathcal{P}_5 \leftarrow \mathcal{P}_6 \\
 & \mathcal{P}_8 \leftarrow \mathcal{P}_{12} & \mathcal{P}_9 \leftarrow \mathcal{P}_{10} & \mathcal{P}_{11} \leftarrow \mathcal{P}_{15} & \mathcal{P}_{13} \leftarrow \mathcal{P}_{14} \\
\underline{5.Takt:} & \mathcal{P}_1 \rightarrow \mathcal{P}_5 & \mathcal{P}_2 \rightarrow \mathcal{P}_6 & \mathcal{P}_4 \rightarrow \mathcal{P}_8 & \\
 & \mathcal{P}_7 \rightarrow \mathcal{P}_{11} & \mathcal{P}_9 \rightarrow \mathcal{P}_{13} & \mathcal{P}_{10} \rightarrow \mathcal{P}_{14} & \\
\underline{6.Takt:} & \mathcal{P}_1 \leftarrow \mathcal{P}_5 & \mathcal{P}_2 \leftarrow \mathcal{P}_6 & \mathcal{P}_4 \leftarrow \mathcal{P}_8 & \\
 & \mathcal{P}_7 \leftarrow \mathcal{P}_{11} & \mathcal{P}_9 \leftarrow \mathcal{P}_{13} & \mathcal{P}_{10} \leftarrow \mathcal{P}_{14} & \\
\underline{7.Takt:} & \mathcal{P}_5 \rightarrow \mathcal{P}_9 & \mathcal{P}_6 \rightarrow \mathcal{P}_{10} & & \\
\underline{8.Takt:} & \mathcal{P}_5 \leftarrow \mathcal{P}_9 & \mathcal{P}_6 \leftarrow \mathcal{P}_{10} & &
\end{array}
$$

Ein Vergleich der Grafik aus Abbildung 3 mit den folgenden Kommunikationsschemata zeigt, daß das Prinzip der Koppelkantenkommunikation unmittelbar auf die Lagebeziehungen der Subdomains eingeht, indem andere und insbesondere weniger Kommunikationswege benutzt werden:

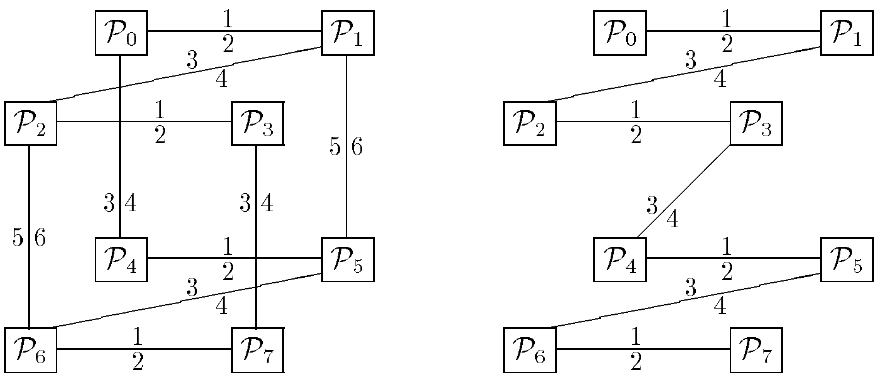

Abbildung 9: Keil– (links) und Streifengebiet (rechts) auf 8 Prozessoren

Nach (3.2) ist der Aufwand der Hypercubekommunikation für die Crosspointdaten von der Ordnung $\mathcal{O}\left(2(p-1)n_\bullet^C\right)$, wobei $n_\bullet^C$ die Zahl der Crosspointfreiheitsgrade je Prozessor sei. Entsprechend weniger Speicherplatz wird auch für die Kommunikationshilfsvektoren H und IH (s. Abschnitt 3.1) benötigt. H kann zugleich auch für die Koppelrandkommunikation benutzt werden, wenn seine Länge wenigstens gleich der maximalen Anzahl der Freiheitgrade ist, die im jeweiligen Prozessor auf einem Koppelrandstück lokalisiert sind.

Bezeichnet $\mathcal{N}$ die maximale Anzahl der Koppelkanten eines Subdomains, so ist der Kommunikationsaufwand für den Austausch aller Koppelranddaten von der Ordnung $\mathcal{O}\left(2(n_*^C - n_\bullet^C)\right)$ zuzüglich $2\mathcal{N}$ Setup–Zeiten. Zusammen mit dem Aufwand der Kommunikation für die Crosspointdaten ergibt sich damit bei dieser Kommunikationstechnologie insgesamt ein Aufwand von

$$\mathcal{O}\left((2p-4)n_\bullet^C + 2n_*^C\right) \qquad \text{zuzüglich} \;\; 2(\mathcal{N}+\mu) \;\; \text{Setup–Zeiten} \tag{3.4}$$

Gegenüber der Hypercubekommunikation bedeutet dies eine Einsparung (Differenz des Kommunikationsaufwands) der Größenordnung

$$\mathcal{O}\left((2p-4)\left(n_*^C - n_\bullet^C\right)\right) \;, \quad p > 1 \;,$$

die mit der Prozessorzahl p und der Feinheit des Netzes wächst und die $2\mathcal{N}$ zusätzliche Setup–Zeiten erfordert.

Ein Vergleich von (3.4) mit (3.3) zeigt die Reserven, die mit weiteren Verbesserungen des Kommunikationsprinzips noch erschlossen werden können. Als Aufwandsdifferenz ergibt sich $\mathcal{O}\left(2(p-4)n_\bullet^C\right)$, d.h., bei mehr als 4 Prozessoren sind weitere Einsparungen am Kommunikationsaufwand für die Crosspointdaten möglich, die aber nur bei relativ großen Prozessorzahlen ins Gewicht fallen, da $n_\bullet^C$ i.allg. eine kleine Zahl ist. Außerdem können im Fall des Streifengebiets 2μ und im Fall des Keilgebiets 2, 4 bzw. 8 Setup–Zeiten bei Verwendung von 4, 8 bzw. mehr als 8 Prozessoren eingespart werden.

3.4 Crosspointkommunikation

Da die Zahl der Prozessoren, deren Subdomain Ω_s einen bestimmten Crosspoint enthält, von der Geometrie der Gebietszerlegung $\Omega = \cup_{s=0}^{p-1}\Omega_s$ abhängt, ist es i.allg. schwierig, für die Crosspointdaten eine direkte Kommunikation nur der betroffenen Prozessoren zu benutzen.

Ist die Gebietszerlegung durch ein Tensorprodukt–Vierecksnetz definiert, gehört kein Crosspoint zu mehr als 4 Subdomains. Unter dieser Voraussetzung kann die Kommunikation für die Crosspointdaten auf einfache Weise durch eine dem Prinzip aus Abschnitt 3.3 ähnliche Technologie ersetzt werden.

Drei Klassen von Crosspoints sind zu unterscheiden:

Gehört ein Crosspoint zu genau zwei Subdomains (Q in Abb. 10), dann liegt er auf einer gemeinsamen Koppelkante dieser Teilgebiete und die Daten in diesem Punkt können zusammen mit den Koppelranddaten ausgetauscht werden.

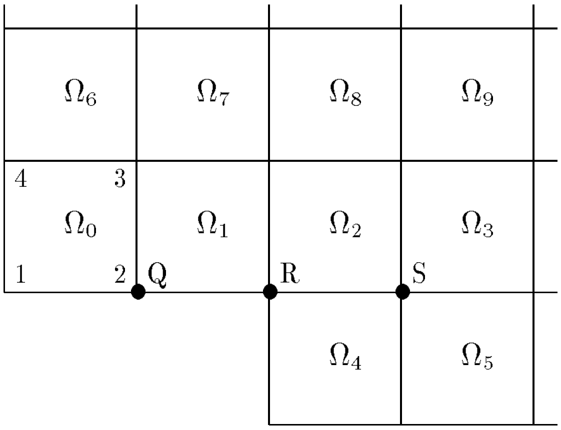

Abbildung 10: 3 Klassen von Crosspoints

Wenn drei Teilgebiete den gleichen Crosspoint besitzen (R), dann liegt er für zwei Paare von Subdomains (Ω_1 und Ω_2 bzw. Ω_2 und Ω_4) auf der jeweiligen gemeinsamen Koppelkante. Zwischen den entsprechenden Prozessorpaaren kann der Austausch der Crosspointdaten ebenfalls gemeinsam mit den Koppelfreiheitsgraden erfolgen. Für das dritte Prozessorpaar ($\mathcal{P}_1$ und $\mathcal{P}_4$) muß ein zusätzliches "Link" definiert werden, über das der Datenaustausch in einer separaten Kommunikation erfolgt.

Stoßen vier Teilgebiete in einem gemeinsamen Crosspoint (S) aneinander, kann der Datenaustausch zwischen den vier Prozessorpaaren ($\mathcal{P}_2$ und $\mathcal{P}_3$, $\mathcal{P}_4$ und $\mathcal{P}_5$, $\mathcal{P}_2$ und $\mathcal{P}_4$ sowie $\mathcal{P}_3$ und $\mathcal{P}_5$), deren Subdomains jeweils eine gemeinsame Koppelkante besitzen, wiederum gemeinsam mit den Koppelfreiheitsgraden erfolgen. Hier müssen zwei zusätzliche Kommunikationen über die zwischen den Prozessoren $\mathcal{P}_3$ und $\mathcal{P}_4$ bzw. $\mathcal{P}_2$ und $\mathcal{P}_5$ zu definierenden virtuellen Links ausgeführt werden.

Werden die Crosspoints in allen Subdomains lokal einheitlich (etwa so wie in Ω_0 aus Abbildung 10) numeriert, dann müssen bei diesen Kommunikationen keine zusätzlichen Informationen über Crosspointnummern, zu denen die ausgetauschten Daten gehören, mitgeliefert werden, weil stets die gleichen Crosspointnummern aufeinandertreffen.

Bei der vorausgesetzten Art der Gebietszerlegung hat ein Subdomain $\overline{\Omega}_s$ mit höchstens 8 anderen Teilgebieten $\overline{\Omega}_t$ einen nichtleeren Durchschnitt. Folglich muß ein Prozessor $\mathcal{P}_s$ mit höchstens 8 anderen Prozessoren $\mathcal{P}_t$ kommunizieren. Der Durchschnitt mit bis zu 4 dieser Teilgebiete ist jeweils eine gemeinsame Koppelkante. Die Kommunikation mit den entsprechenden Prozessoren kann wie in Abschnitt 3.3 beschrieben unter Einschluß der Daten für die beiden die Koppelkante begrenzenden Crosspoints erfolgen. Der Durchschnitt mit den übrigen Teilgebieten ist jeweils ein Crosspoint. Zu den entsprechenden (maximal 4) Prozessoren ist jeweils ein weiteres virtuelles Link zu definieren. Die Reihenfolge, in der diese Crosspointkommunikationen auszuführen sind, kann durch nochmaliges Anwenden des Algorithmus von Seite 11 bestimmt werden, wenn dort "Koppelkante" durch "Crosspoint" ersetzt wird.

Im Beispiel des Streifengebietes aus Abbildung 5 sind keine zusätzlichen Crosspointkommunikationen erforderlich, weil alle Crosspoints vom Typ "Q"(Abb. 10) sind.

Für das Keilgebiet wird folgende Crosspointnumerierung erhalten:

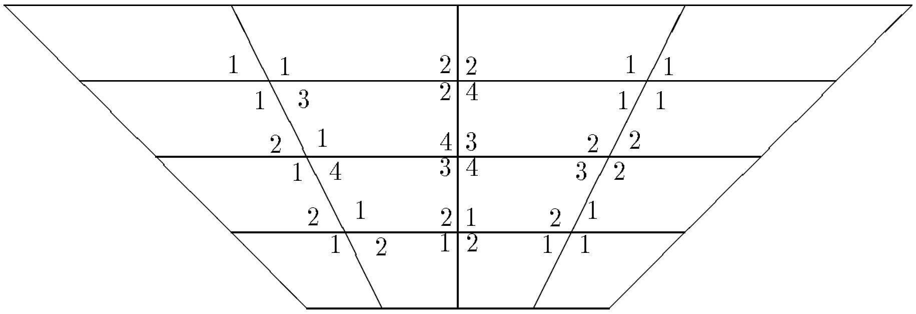

Abbildung 11: Crosspointnumerierung beim Keilgebiet

Da zuerst alle Koppelkantenkommunikationen und danach alle Crosspointkommunikationen ausgeführt werden sollen, bleiben die bereits auf Seite 12 aufgelisteten Takte 1 bis 6 unverändert. In den Takten 7 und 8 beginnen die Prozessoren, die bereits alle Koppelkantenkommunikationen ausgeführt haben, mit der Crosspointkommunikation, die in den anschließenden Takten 9 bis 16 ihren Abschluß findet (in Klammern: die letzten Aktionen der Koppelkantenkommunikation).

$\underline{7.Takt:}$	$(\mathcal{P}_5 \rightarrow \mathcal{P}_9)$	$(\mathcal{P}_6 \rightarrow \mathcal{P}_{10})$	$\mathcal{P}_2 \rightarrow \mathcal{P}_7$	$\mathcal{P}_8 \rightarrow \mathcal{P}_{13}$	$\mathcal{P}_{11} \rightarrow \mathcal{P}_{14}$
$\underline{8.Takt:}$	$(\mathcal{P}_5 \leftarrow \mathcal{P}_9)$	$(\mathcal{P}_6 \leftarrow \mathcal{P}_{10})$	$\mathcal{P}_2 \leftarrow \mathcal{P}_7$	$\mathcal{P}_8 \leftarrow \mathcal{P}_{13}$	$\mathcal{P}_{11} \leftarrow \mathcal{P}_{14}$
$\underline{9.Takt:}$	$\mathcal{P}_0 \rightarrow \mathcal{P}_5$	$\mathcal{P}_1 \rightarrow \mathcal{P}_6$	$\mathcal{P}_4 \rightarrow \mathcal{P}_9$	$\mathcal{P}_{10} \rightarrow \mathcal{P}_{15}$	
$\underline{10.Takt:}$	$\mathcal{P}_0 \leftarrow \mathcal{P}_5$	$\mathcal{P}_1 \leftarrow \mathcal{P}_6$	$\mathcal{P}_4 \leftarrow \mathcal{P}_9$	$\mathcal{P}_{10} \leftarrow \mathcal{P}_{15}$	
$\underline{11.Takt:}$	$\mathcal{P}_1 \rightarrow \mathcal{P}_4$	$\mathcal{P}_2 \rightarrow \mathcal{P}_5$	$\mathcal{P}_3 \rightarrow \mathcal{P}_6$	$\mathcal{P}_7 \rightarrow \mathcal{P}_{10}$	$\mathcal{P}_9 \rightarrow \mathcal{P}_{14}$
$\underline{12.Takt:}$	$\mathcal{P}_1 \leftarrow \mathcal{P}_4$	$\mathcal{P}_2 \leftarrow \mathcal{P}_5$	$\mathcal{P}_3 \leftarrow \mathcal{P}_6$	$\mathcal{P}_7 \leftarrow \mathcal{P}_{10}$	$\mathcal{P}_9 \leftarrow \mathcal{P}_{14}$
$\underline{13.Takt:}$	$\mathcal{P}_5 \rightarrow \mathcal{P}_{10}$	$\mathcal{P}_6 \rightarrow \mathcal{P}_{11}$	$\mathcal{P}_9 \rightarrow \mathcal{P}_{12}$		
$\underline{14.Takt:}$	$\mathcal{P}_5 \leftarrow \mathcal{P}_{10}$	$\mathcal{P}_6 \leftarrow \mathcal{P}_{11}$	$\mathcal{P}_9 \leftarrow \mathcal{P}_{12}$		
$\underline{15.Takt:}$	$\mathcal{P}_5 \rightarrow \mathcal{P}_8$	$\mathcal{P}_6 \rightarrow \mathcal{P}_9$	$\mathcal{P}_{10} \rightarrow \mathcal{P}_{13}$		
$\underline{16.Takt:}$	$\mathcal{P}_5 \leftarrow \mathcal{P}_8$	$\mathcal{P}_6 \leftarrow \mathcal{P}_9$	$\mathcal{P}_{10} \leftarrow \mathcal{P}_{13}$		

Da beispielsweise der Prozessor $\mathcal{P}_5$ an jedem der 16 Takte beteiligt ist, kann die Zahl der Takte nicht durch Änderung der Kommunikationsreihenfolge reduziert werden.

Sowohl für das Streifen- als auch für das Keilgebiet ergibt sich als Gesamtkommunikationsaufwand exakt der in Abschnitt 3.2 angegebene theoretisch unumgängliche Aufwand.

4 Einige numerische Resultate

Über dem Gebiet aus Abbildung 6 wird das bereits in [5] zugrunde gelegte technische Problem betrachtet. Ein keilförmiger, beidseitig längs der Hälfte seiner schrägen Kanten fest eingespannter und ansonsten frei beweglicher Körper wird mit einer flächenverteilten Last nach unten gedrückt.

Der Körper aus Abbildung 5 sei an seinem linken Rand fest eingespannt und wie in Abbildung 12 belastet.

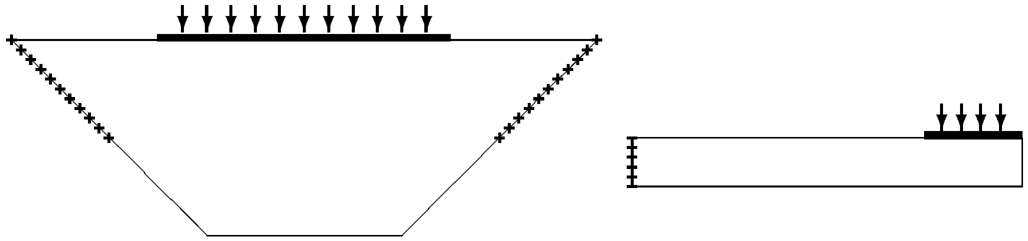

Abbildung 12: Randbedingungen und Lasten

Die Rechnungen wurden auf Vierecksnetzen mit etwa gleichen Kantenlängen in beiden Richtungen ausgeführt. Die CG–Iteration wurde beendet, wenn die Zahl $\gamma^{(i+1)}$ von Schritt (21) in (2.6) die Ungleichung $\gamma^{(i+1)} \leq 10^{-12}\,\gamma^{(0)}$ mit $\gamma^{(0)}$ von Schritt (10) aus (2.6) erfüllt. Die angegebenen Rechenzeiten für die Lösung der Gleichungssysteme wurden auf Transputersystemen der Firma PARSYTEC (Prozessortyp T800 bei bis zu 16 und T805 bei 32 bis 128 Prozessoren) ermittelt.

4.1 Abhängigkeit der Iterationszahlen von der Zahl der Prozessoren und der Zahl der Unbekannten

In [5] wurde die Abhängigkeit der Iterationszahlen von der Zahl der zur Lösung benutzten Prozessoren bei konstanter Dimension n des Gleichungssystems für das Keilgebiet aus Abbildung 6 untersucht. Siehe auch Tabelle 11.
Einige Daten für das Streifengebiet (Abb. 5) sind in Tabelle 1 zusammengestellt:

Tabelle 1: Zusammenhang von Iterationszahlen und Prozessoranzahl

n	1	2	4	8	16	32	64	128	# Proc.
192	103	99	94	98	115				
384	132	127	131	139	151				
640	112	111	113	123	145	186			
960	144	141	144	159	175				
1344		141	146	164	180	239			
2304			126	147	173	231	391		
3520			160	183	209	277			
4992				189	216	285	491		# Iter.
8704				169	199	266	460	910	
13440					243	323	555		
19200					251	331	564	1126	
33792						297	514	1040	
52480							617	1405	
75264							635	1348	
102144							603	1233	
133120								1191	

Bei dieser Randwertaufgabe wirken die vorgegebenen Kräfte jeweils nur im letzten Subdomain Ω_{p-1}, Randbedingungen sind nur in Ω_0 vorgegeben und die einzelnen Subdomains sind stets nur paarweise über je eine gemeinsame Kante miteinander verbunden. Daher ist die Existenz der Randkräfte nach frühestens p Iterationsschritten auch im Prozessor $\mathcal{P}_0$ bekannt. Diese Besonderheiten der betrachteten Randwertaufgabe sind die Hauptursachen für das meist drastische Anwachsen der Iterationszahlen mit der Prozessorzahl.

Andererseits wachsen die Iterationszahlen bei konstanter Prozessorzahl mit der Zahl der Freiheitsgrade wesentlich moderater an. So stieg bei 16 Prozessoren die Iterationszahl auf das 2.2–fache, wenn die Zahl der Freiheitsgrade auf das Hundertfache gesteigert wurde.

4.2 Zeitverhalten der Kommunikationstechnologien

In diesem Abschnitt soll das Laufzeitverhalten der drei Kommunikationstechnologien auf verschiedenen Prozessorzahlen und bei unterschiedlichen Gleichunssystemdimensionen (n) miteinander verglichen werden. In den mit T_a überschriebenen Spalten der folgenden Tabellen ist die für arithmetische Operationen verbrauchte Zeit und in den mit T_g markierten Spalten die benötigte Gesamtrechenzeit angegeben. Die Differenz beider Werte entspricht dem Kommunikationsaufwand. Ein hochgestelltes H verweist auf die Hypercube– , ein K auf die Koppelrand– und ein C auf die Crosspointkommunikation. Die Zeitangaben erfolgen in Sekunden.

Am Ende des Abschnitts 3.3 war festgestellt worden, daß sich der Aufwand der reinen Hypercubekommunikation von dem der Koppelkantenkommunikation mit Hypercubekommunikation nur für die Crosspointdaten in der Größenordnung

$$\mathcal{O}\left((2p-4)(n_*^C - n_\bullet^C)\right) \qquad (p \geq 2) \tag{4.1}$$

unterscheidet und daß die dritte betrachtete Kommunikationstechnologie, die den Hypercube auch für die Crosspointdaten nicht benutzt, weitere Einsparungen der Ordnung

$$\mathcal{O}\left(2(p-4)n_\bullet^C\right) \qquad (p \geq 4) \tag{4.2}$$

ermöglicht. Bei der Bewertung des Zeitverhaltens ist zu berücksichtigen, daß von den Einsparungen (4.1) der Zeitbedarf von $2\mathcal{N}$ ($\mathcal{N}$: Maximale Koppelkantenzahl eines Subdomain) zusätzlichen Setup–Operationen abzuziehen ist, daß bei (4.2) zusätzlich 2μ Setup–Zeiten beim Streifengebiet und 2,4 bzw. 8 Setup–Zeiten beim Keilgebiet auf 4, 8 bzw. mehr als 8 Prozessoren eingespart werden und daß diese Aufwandsunterschiede in jedem Iterationsschritt wirken.

In Tabelle 2 sind die auf einem Prozessor gemessenen Zeiten zusammengestellt. Da hier keinerlei Kommunikation auftritt, ist $T_a^{\cdot} = T_g^{\cdot} =: T^{\cdot}$.
Die für die drei Programmversionen gemessenen Zeiten unterscheiden sich nur sehr geringfügig voneinander, jedoch mit der eindeutigen Tendenz $T^H \geq T^K \geq T^C$ obwohl keine Kommunikation stattfindet. Dies hat seine Ursache in dem bei jeder Version veränderten Operationsablauf bei der Ausführung von Schritt (15) aus (2.6), den das Programm auch bei p=1 formal durchlaufen muß. Bezüglich der mittels Hypercubekommunikation ausgetauschten Daten ist ein Suchprozeß in der Liste aller empfangenen Daten erforderlich, der durch schneller abarbeitbare Kopieroperationen ersetzt werden kann, wenn direkte Kommunikationswege benutzt werden. Dieser

Tabelle 2: Zeitmessungen auf einem Prozessor

Gebiet	n	T^H	T^K	T^C
Streifen-gebiet	192	7.08	6.79	6.71
	384	32.30	31.73	31.48
	640	68.61	68.30	68.03
	960	193.92	193.08	193.13
Keil-gebiet	142	1.17	1.06	1.06
	310	4.88	4.61	4.68
	542	8.80	8.67	8.67
	838	25.03	24.86	24.84
	1198	42.51	42.32	42.32

Effekt verbessert zusätzlich zu den Einsparungen im Kommunikationsaufwand das Laufzeitverhalten der drei Programmversionen.

Für die Rechenzeiten in Tabelle 3 gelten diese Anmerkungen in gleicher Weise, denn nach (4.1) und (4.2) ist aus den veränderten Kommunikationstechnologien bei 2 Prozessoren kein Laufzeitunterschied zu erwarten.

Tabelle 3: Zeitmessungen auf zwei Prozessoren

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
Streifen-gebiet	192	3.33	3.55	3.16	3.33	3.14	3.26
	384	12.38	12.77	11.80	12.06	11.16	11.36
	640	19.97	20.40	19.80	20.07	20.04	20.27
	960	53.12	53.83	52.40	52.85	52.13	52.52
	1344	88.78	89.64	88.22	88.75	88.22	88.69
	1792	141.57	142.57	140.89	141.50	140.76	141.28
Keil-gebiet	142	0.94	0.97	0.90	0.93	0.72	0.73
	310	2.33	2.39	2.33	2.37	2.07	2.10
	542	3.41	3.48	3.29	3.34	3.25	3.28
	838	8.93	9.03	9.06	9.13	8.70	8.75
	1198	13.92	14.05	13.77	13.86	13.76	13.81
	1622	21.99	22.16	21.77	21.86	21.79	21.86
	2110	27.05	27.20	26.94	27.02	26.89	26.95
	2662	51.55	51.77	51.17	51.29	51.13	51.22

Werden 4 Prozessoren benutzt, ist nach (4.1) ein meßbarer Zeitunterschied zwischen der Hypercube– und der Koppelkantenkommunikation zu erwarten, während sich letztere nach (4.2) kaum von der Crosspointkommunikationsversion unterscheiden dürfte.

Beim Streifengebiet variiert die Zeitersparnis $T_g^H - T_g^K$ zwischen einer halben und 4 Sekunden und $T_g^K - T_g^C$ liegt meist in der Nähe von 0.25 Sekunden, was der theoretischen Voraussage zumindest nicht widerspricht. Beim Keilgebiet fallen die Unterschiede deutlich geringer aus, was insbesondere an den wesentlich geringeren Iterationszahlen liegt.

Tabelle 4: Zeitmessungen auf vier Prozessoren

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
	192	4.52	4.84	2.14	2.40	1.98	2.17
	384	5.96	6.58	5.87	6.31	5.71	6.04
	640	9.36	10.02	7.36	7.78	7.29	7.61
	960	21.19	22.19	17.56	18.15	17.55	18.01
Streifen-	1344	28.59	29.83	27.91	28.63	26.61	27.20
gebiet	1792	41.15	42.55	40.47	41.25	40.43	41.07
	2304	51.48	52.85	50.72	51.47	50.62	51.24
	2880	90.76	92.60	90.08	91.09	90.01	90.81
	3520	129.64	131.68	128.93	130.10	128.88	129.84
	4224	180.91	183.21	180.07	181.35	180.02	181.03
	142	0.56	0.62	0.52	0.58	0.58	0.62
	310	1.33	1.44	1.29	1.37	1.44	1.49
	542	1.80	1.91	1.74	1.81	1.83	1.89
	838	4.12	4.30	4.08	4.18	4.25	4.34
Keil-	1622	9.13	9.40	9.04	9.16	9.01	9.13
gebiet	2110	10.99	11.24	10.85	10.96	10.77	10.88
	2662	18.56	18.89	18.49	18.63	18.42	18.57
	3278	25.88	26.26	25.71	25.87	25.66	25.83
	3958	34.71	35.15	34.60	34.78	34.49	34.70
	4702	43.73	44.21	43.56	43.77	43.40	43.62

Tabelle 5 enthält einige der auf 8 Prozessoren ermittelten Rechenzeiten. Nach (4.1) und (4.2) sind ein monotones Anwachsen der Zeitersparnis $T_g^H - T_g^K$ mit der Problemgröße n und etwa konstante Unterschiede zwischen T_g^K und T_g^C zu erwarten: Die Folge der $T_g^H - T_g^K$–Werte wächst bei beiden betrachteten Gebieten monoton von 0.14 auf 4.91 (Streifengebiet) bzw. von 0.04 auf 1.81 Sekunden (Keilgebiet), was die These "$\mathcal{O}\left(12n_*^C\right)$" unterstützt. Die Differenzen $T_g^K - T_g^C$ bewegen sich eher unregelmäßig zwischen 0.21 und 1.50 bzw. zwischen –0.04 und 0.19 Sekunden.

Tabelle 5: Zeitmessungen auf acht Prozessoren

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
	192	1.23	1.78	1.21	1.64	1.16	1.39
	384	2.93	3.95	2.88	3.56	2.69	3.10
	640	3.56	4.67	3.48	4.12	3.34	3.73
	960	8.41	10.12	8.35	9.28	7.50	8.10
	1344	11.03	13.08	11.17	12.17	10.71	11.37
	1792	15.22	17.66	14.86	16.00	14.82	15.63
Streifen-	2304	18.14	20.50	18.13	19.08	17.67	18.42
gebiet	2880	32.47	35.60	32.67	34.00	31.55	32.50
	3520	43.19	46.80	42.41	43.86	42.52	43.65
	4224	57.49	61.53	56.77	58.47	56.68	57.91
	4992	70.77	74.95	69.99	71.63	69.91	71.10
	6720	105.98	110.49	105.08	106.51	105.00	106.20
	8704	142.73	147.33	141.76	143.10	141.69	142.56
	9792	199.29	204.65	198.14	199.74	198.06	199.07

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
Keil-gebiet	142	0.52	0.68	0.51	0.64	0.49	0.59
	310	0.98	1.24	0.95	1.12	0.93	1.05
	542	1.26	1.58	1.22	1.40	1.18	1.31
	838	2.36	2.80	2.28	2.50	2.25	2.43
	1198	3.18	3.73	3.08	3.32	3.06	3.26
	2110	4.93	5.60	4.75	5.00	4.74	4.97
	2662	8.05	8.89	7.79	8.09	7.86	8.13
	3278	10.25	11.19	9.81	10.13	10.01	10.31
	4702	15.69	16.87	15.45	15.83	15.32	15.67
	6382	23.08	24.46	22.85	23.26	22.66	23.07
	8318	29.82	31.25	29.48	29.89	29.46	29.88
	10510	49.30	51.11	48.93	49.44	48.90	49.43
	11702	56.85	58.77	56.44	56.96	56.41	56.97

Bei Verwendung von 16 Prozessoren ist $T_g^H - T_g^K = \mathcal{O}\left(28 n_*^C\right)$ zu erwarten, d.h., die Laufzeitunterschiede sollten monoton mit n wachsen und größer sein als bei 8 Prozessoren. Die Differenzen $T_g^K - T_g^C$ sollten ebenfalls größer sein als im Fall p=8:

Tabelle 6: Zeitmessungen auf 16 Prozessoren

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
Streifen-gebiet	192	1.99	2.99	1.46	2.24	0.84	1.14
	384	2.18	3.93	2.32	3.43	1.80	2.27
	640	2.58	4.67	2.74	3.85	2.24	2.76
	960	5.77	8.81	4.88	6.31	4.68	5.39
	1344	5.78	9.40	5.57	7.11	5.24	6.05
	1792	7.42	11.69	7.24	8.93	6.60	7.54
	2304	9.93	14.42	8.97	10.61	8.32	9.28
	2880	15.01	20.75	13.77	15.60	13.62	14.76
	3520	19.98	26.61	17.41	19.59	17.28	18.57
	4224	22.80	30.42	22.72	25.15	21.84	23.20
	4992	27.85	36.02	25.93	28.43	25.70	27.15
	6720	37.15	46.39	36.15	38.74	35.99	37.63
	8704	48.31	57.71	47.23	49.60	47.09	48.46
	9792	69.65	80.78	68.36	71.08	68.20	69.69
	10944	81.58	93.52	80.17	82.78	80.01	81.59
	13440	112.92	127.45	111.22	113.96	111.05	112.85
	16192	152.84	168.74	150.87	153.56	150.68	152.55
	19200	188.96	205.48	186.83	189.76	186.66	188.53
Keil-gebiet	142	0.39	0.73	0.53	0.81	0.49	0.69
	310	0.83	1.36	0.78	1.12	1.00	1.24
	542	1.07	1.74	0.94	1.28	0.91	1.16
	838	1.83	2.74	1.66	2.07	1.63	1.94
	1198	2.68	3.81	2.12	2.56	2.08	2.43
	1622	2.76	4.14	2.75	3.23	2.71	3.14
	2110	3.29	4.96	3.08	3.52	3.09	3.49
	2662	6.12	8.05	4.98	5.49	5.05	5.51
	3278	6.59	8.57	6.32	6.88	6.24	6.75
	3958	8.84	11.09	7.98	8.57	7.78	8.34
	4702	9.51	11.92	9.30	9.90	9.22	9.80

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
Keil-gebiet	6382	14.03	16.86	13.55	14.19	13.31	13.95
	8318	18.41	21.55	18.31	18.97	17.95	18.67
	10510	28.77	32.49	28.22	28.97	28.04	28.87
	11702	33.06	37.04	32.57	33.36	32.47	33.34
	12958	39.56	43.95	38.94	39.79	39.02	39.96
	15662	53.96	59.08	53.39	54.36	53.29	54.41
	18622	67.78	73.32	67.03	68.07	67.10	68.31

Die aus den gemessenen Werten ermittelten Zeitunterschiede $T_g^H - T_g^K$ wachsen bei beiden Testaufgaben wie theoretisch erwartet und sind etwa doppelt so groß wie bei 8 Prozessoren (zwischen 0.75 und 16 Sekunden beim Streifengebiet und bis zu 5 Sekunden beim Keilgebiet). Die Werte für $T_g^K - T_g^C$ sind beim Keilgebiet vergleichsweise gering. Beim Streifengebiet liegen sie etwa konstant bei ca. 1 Sekunde und sind damit ungefähr doppelt so groß wie bei 8 Prozessoren.

Für Rechnungen auf 32 Prozessoren wurde $T_g^H - T_g^K = \mathcal{O}\left(60n_*^C\right)$ vorausgesagt:

Tabelle 7: Zeitmessungen auf 32 Prozessoren

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
Streifen-gebiet	640	2.73	8.18	2.47	5.35	2.22	3.16
	1344	5.98	15.82	5.30	9.22	4.94	6.38
	2304	7.56	19.90	6.74	10.71	6.40	7.95
	3520	15.04	32.71	13.79	18.97	13.37	15.40
	4992	19.57	41.13	17.94	23.40	17.52	19.86
	6720	24.08	48.37	22.20	27.25	21.79	23.89
	8704	29.19	55.56	27.06	32.07	26.68	28.93
	10944	49.81	83.25	47.03	52.73	46.60	49.04
	13440	64.06	102.93	60.74	66.95	60.27	63.16
	16192	81.21	126.07	77.34	83.61	76.86	79.89
	19200	95.55	143.06	91.37	97.89	90.92	93.89
	25984	129.47	181.58	124.81	131.08	124.38	127.28
	33792	170.63	226.87	165.54	171.48	165.14	167.88
	42624	284.88	356.68	278.26	285.19	277.82	281.03
	49842	–	–	324.73	331.95	324.29	327.61
Keil-gebiet	542	1.57	3.30	1.13	2.01	1.08	1.48
	1198	3.26	6.18	1.95	3.05	1.92	2.45
	3278	5.50	10.90	4.85	6.20	4.97	5.72
	6382	10.05	17.66	8.45	9.89	7.59	8.54
	12958	21.12	33.18	20.16	21.89	20.37	21.62
	15662	26.74	40.66	25.68	27.53	26.05	27.41
	18622	32.08	47.10	31.23	33.13	30.63	32.03
	25310	43.88	60.55	42.24	44.12	42.11	43.59
	33022	60.66	79.51	58.89	60.84	58.96	60.60
	41758	95.28	118.09	93.04	95.25	93.07	94.96
	46510	–	–	108.82	111.11	108.70	110.70

Die ermittelten Zeitunterschiede $T_g^H - T_g^K$ wachsen monoton, sind knapp dreimal so groß wie bei 16 Prozessoren und betragen 20 bis 30% der Zeiten T_g^H, werden also sowohl mit wachsender Prozessorzahl als auch mit wachsenden Freiheitsgradzahlen zunehmend signifikanter. Die Werte für $T_g^K - T_g^C$ sind für das Streifengebiet monoton wachsend, was jedoch ausschließlich durch das Anwachsen der Iterationszahlen bedingt sein dürfte, und etwa doppelt so groß wie bei 16 Prozessoren. Beim Keilgebiet ist keine klare Tendenz für $T_g^K - T_g^C$ feststellbar.

Bei Rechnungen auf 64 Prozessoren müßten sich T_g^H und T_g^K in der Größenordnung $\mathcal{O}\left(124 n_*^C\right)$ unterscheiden:

Tabelle 8: Zeitmessungen auf 64 Prozessoren

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
	2304	12.35	66.83	13.68	28.79	9.11	12.44
	4992	28.21	125.45	20.47	40.24	19.54	24.73
	13440	70.73	248.76	52.90	77.24	50.93	57.91
Streifen-	25984	109.46	346.00	83.23	105.17	81.48	89.99
gebiet	42624	213.35	540.84	175.59	199.54	173.68	180.53
	52480	273.05	660.40	228.27	254.17	226.25	234.17
	63360	346.82	796.96	294.36	322.01	292.24	300.73
	75264	402.95	879.81	347.20	374.49	345.19	353.93
	88192	455.17	945.56	397.54	423.58	395.80	404.02
	102144	–	–	464.49	490.28	462.67	470.86
	542	1.33	6.19	1.03	3.85	0.87	1.51
	1198	2.06	9.87	1.95	5.12	1.48	2.27
	4702	5.67	23.50	4.53	8.44	4.54	5.74
	8318	8.67	32.76	6.84	10.90	6.69	8.27
	15662	20.55	58.27	16.79	21.77	16.51	18.62
Keil-	25310	31.85	77.30	27.07	31.95	26.78	29.07
gebiet	33022	43.04	95.26	37.43	42.38	37.04	39.55
	41758	66.30	129.35	59.25	64.75	59.02	61.91
	46510	76.24	143.70	68.64	74.26	68.36	71.39
	51518	89.74	163.69	81.42	87.30	81.07	84.31
	62302	119.66	205.28	109.81	116.11	109.53	113.27
	74110	148.09	240.18	137.41	143.82	137.17	141.04
	80398	–	–	147.28	153.46	147.04	150.79

Die in Tabelle 8 angegebenen Rechenzeiten sind in den Abbildungen 14 und 13 grafisch dargestellt:

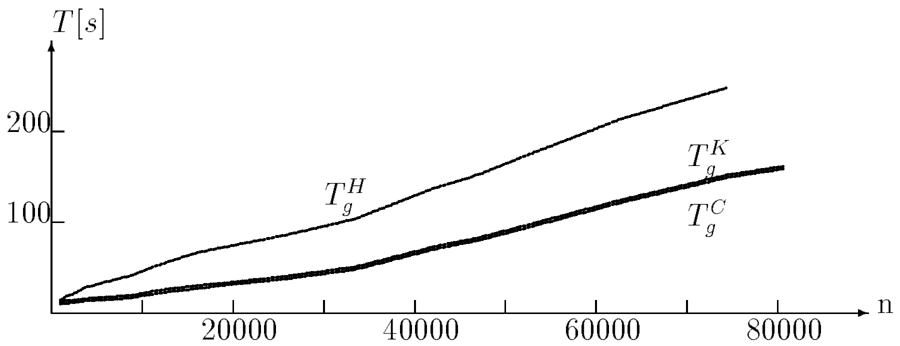

Abbildung 13: Rechenzeiten bei 64 Prozessoren (Keilgebiet)

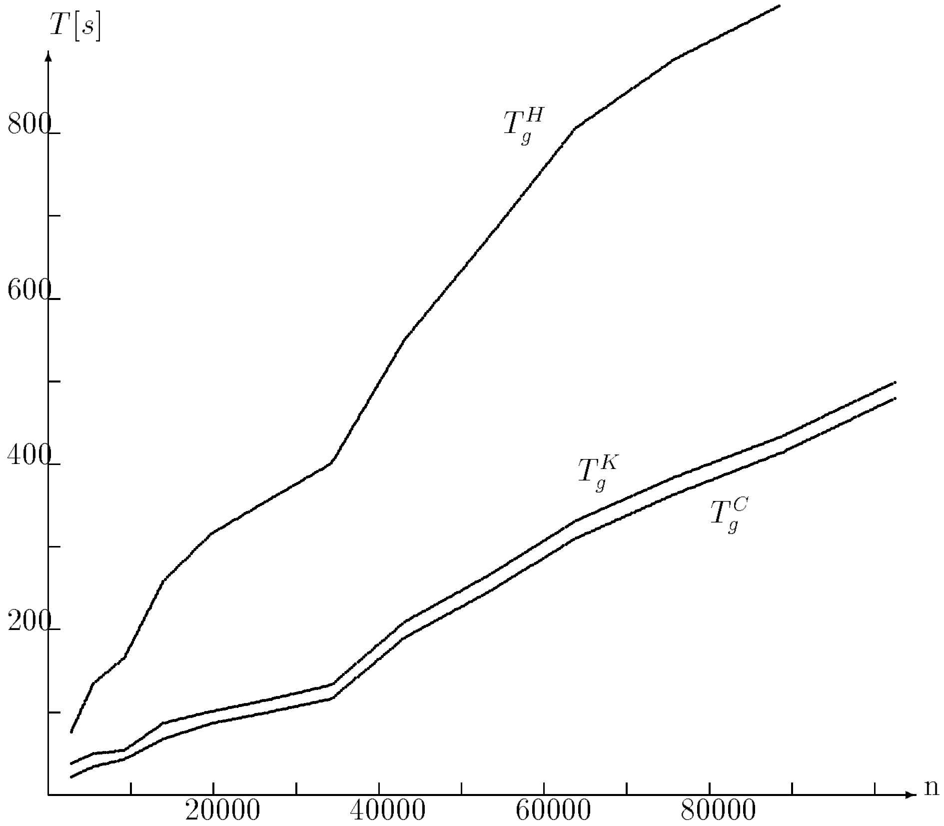

Abbildung 14: Rechenzeiten bei 64 Prozessoren (Streifengebiet)

Mit Zeiteinsparungen zwischen 14 und 20 Sekunden bzw. 4–40% der Zeit T_g^K beim Streifengebiet und ca. 2.5 Sekunden bzw. 2–50% von T_g^K beim Keilgebiet wird bei 64 und mehr Prozessoren auch die Zeitersparnis $T_g^K - T_g^C$ zunehmend attraktiv. Die Zeitdifferenzen $T_g^H - T_g^K$ erreichen 50–70% der Zeit T_g^H und betragen bis zu 520 Sekunden beim Streifen– und bis zu 96 Sekunden beim Keilgebiet. Beide Zeitdifferenzen betragen das Doppelte bis Fünffache der auf 32 Prozessoren erreichten Werte.

Ist das Berechnungsgebiet Ω auf 128 Prozessoren verteilt, dann kann nach (4.1) und (4.2) eine erneute Verdoppelung der Zeitunterschiede zwischen den drei Kommunikationstechniken gegenüber den auf 64 Prozessoren festgestellten Einsparungen erwartet werden. Tatsächlich sind die ermittelten Unterschiede siebenmal so groß. Ein Grund dafür ist sicher in den Prozessorbelastungen, die bei Verwendung der Hypercubekommunikation durch das "Durchrouten" der Daten entlang der physischen Prozessorlinks entstehen, zu sehen. Beim Streifengebiet liegen die Zeitdifferenzen $T_g^H - T_g^K$ mit 740 bis 3600 Sekunden bei 80 bis 90% der jeweiligen Rechenzeiten T_g^H. Für das Keilgebiet konnten je nach Problemgröße 25 bis 500 Sekunden Unterschied festgestellt werden, was 50 bis 80% der Zeiten T_g^H entspricht.
Beim Vergleich der Koppelkanten- mit der Crosspointkommunikation auf 128 Prozessoren ergeben sich Zeitunterschiede zwischen 100 und 160 Sekunden beim Streifen- und 13 bis 20 Sekunden beim Keilgebiet. Das sind 10 bis 70% des Zeitbedarfs der CG–Iteration mit Koppelkantenkommunikation.

Tabelle 9: Zeitmessungen auf 128 Prozessoren

Gebiet	n	T_a^H	T_g^H	T_a^K	T_g^K	T_a^C	T_g^C
Streifengebiet	8704	63.63	886.41	41.04	143.17	33.72	42.62
	19200	142.53	1659.32	93.62	221.99	84.41	98.34
	33792	176.98	2042.43	113.94	234.37	105.52	120.09
	52480	391.00	3521.20	269.22	432.31	258.25	277.41
	75264	473.90	4073.50	329.13	487.39	318.38	339.18
	102144	–	–	385.93	531.77	376.11	396.76
	133120	–	–	475.68	617.91	466.31	487.29
	168192	–	–	789.57	945.78	779.65	803.35
	207360	–	–	1101.04	1281.76	1089.86	1119.34
Keilgebiet	2110	2.93	40.99	2.25	16.01	1.79	3.07
	4702	5.69	73.43	4.16	20.77	3.51	5.25
	8318	7.68	98.49	5.26	22.18	4.65	6.65
	12958	13.66	143.71	9.73	29.32	10.04	12.59
	18622	17.54	177.95	12.22	32.45	12.23	15.10
	25310	21.25	202.66	14.92	34.69	14.29	17.79
	41758	40.59	288.89	30.00	51.59	29.42	33.40
	62302	64.03	396.80	48.13	72.09	47.41	52.07
	100798	99.71	496.12	78.40	101.19	77.73	82.67
	131582	132.71	574.85	107.53	130.09	106.78	112.32
	148510	178.51	680.06	146.82	171.16	145.98	152.08
	166462	–	–	171.52	196.16	170.66	176.80
	185438	–	–	199.52	224.78	198.68	205.13

Die Rechenzeiten aus Tabelle 9 sind in den Abbildungen 15 und 16 grafisch dargestellt:

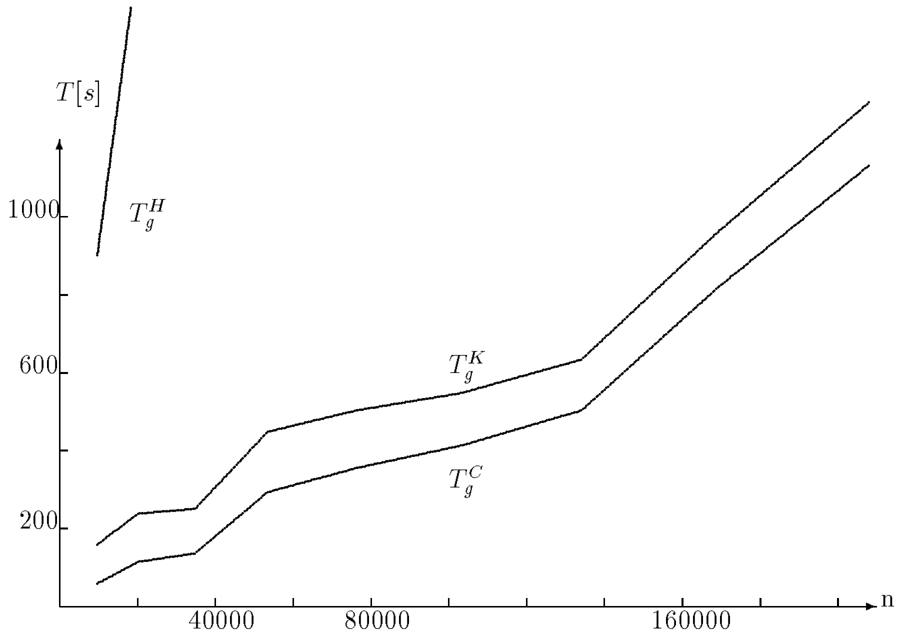

Abbildung 15: Rechenzeiten bei 128 Prozessoren (Streifengebiet)

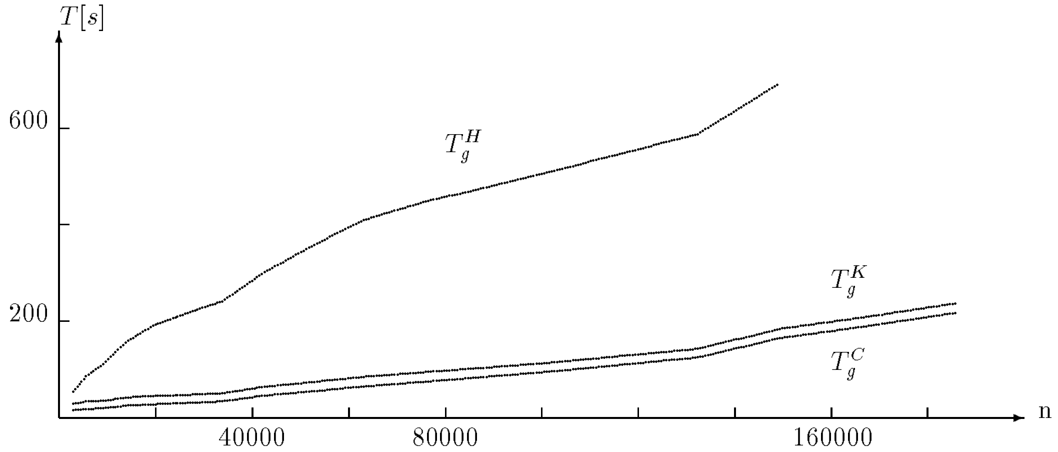

Abbildung 16: Rechenzeiten bei 128 Prozessoren (Keilgebiet)

Zusammenfassung

Die in Abschnitt 3 vorausgesagten Unterschiede im Laufzeitverhalten der drei betrachteten Kommunikationsmodelle konnten anhand der beiden Testaufgaben aus Abbildung 12 praktisch verifiziert werden. Besonders deutlich werden die Unterschiede bei Gebieten Ω, bei denen die Teilgebiete Ω_s nur wenige benachbarte Subdomains haben, wie das beim Streifengebiet der Fall war. Erwartungsgemäß sind die beobachteten Unterschiede zwischen der Koppelkanten– und der Crosspointkommunikation wesentlich geringer als zwischen der Hypercube- und der Koppelkantenkommunikation. Dennoch ist die Crosspointkommunikation insbesondere bei Prozessorzahlen $p \geq 64$ zu bevorzugen.

4.3 Speed Up

In [5] wurden bereits Speed Up–Werte für das Keilgebiet bei Verwendung der Hypercubekommunikation bestimmt und festgestellt, daß ab 32 Prozessoren aufgrund des großen Kommunikationsaufwandes keine akzeptablen Werte mehr erreicht werden konnten. Hier soll untersucht werden, wie sich die beiden anderen Kommunikationstypen auf die Effizienz auswirken.

Aus den in den Tabellen 2–9 angegebenen Rechenzeiten $T_g^{\cdot}$ werden die Speed–Up Werte $S^{\cdot}$ berechnet, indem das k–fache der auf k Prozessoren ermittelten Zeiten durch die entsprechenden auf mehr als k Prozessoren erhaltenen Zeiten dividiert wird, wobei k die kleinste Prozessoranzahl ist, auf der die Aufgabe mit der entsprechenden Zahl von Freiheitsgraden realisiert werden konnte. Werden diese Speed–Up–Werte noch durch die Zahl der benutzten Prozessoren dividiert, ergeben sich die Effizienzen $E^{\cdot}$.
Die so ermittelten Zahlen sind in den Tabellen 10 und 11 zusammengestellt:

Tabelle 10: Speed Up (Streifengebiet)

n	p	#It	T_g^H	T_g^K	T_g^C	S^H	S^K	S^C	E^H	E^K	E^C
192	1	103	7.1	6.8	6.7	–	–	–	–	–	–
	2	99	3.6	3.3	3.3	1.97	2.06	2.03	0.99	1.03	1.02
	4	94	4.8	2.4	2.2	1.48	2.83	3.05	0.37	0.71	0.76
	8	98	1.8	1.6	1.4	3.94	4.25	4.79	0.49	0.53	0.60
	16	115	3.0	2.2	1.1	2.37	3.09	6.09	0.15	0.19	0.38
384	1	132	32.3	31.7	31.5	–	–	–	–	–	–
	2	127	12.8	12.1	11.4	2.52	2.62	2.76	1.26	1.31	1.38
	4	131	6.6	6.3	6.0	4.89	5.03	5.25	1.22	1.26	1.31
	8	139	4.0	3.6	3.1	8.07	8.81	10.16	1.01	1.10	1.27
	16	151	3.9	3.4	2.3	8.28	9.32	13.70	0.52	0.58	0.86
640	1	112	68.6	68.3	68.0	–	–	–	–	–	–
	2	111	20.4	20.1	20.3	3.36	3.40	3.35	1.68	1.70	1.67
	4	113	10.0	7.8	7.6	6.86	8.76	8.95	1.71	2.19	2.24
	8	123	4.7	4.1	3.7	14.60	16.66	18.38	1.82	2.08	2.30
	16	145	4.7	3.8	2.8	14.60	17.97	24.29	0.91	1.12	1.52
	32	186	8.2	5.4	3.2	8.37	12.65	21.25	0.26	0.40	0.66
960	1	144	193.9	193.1	193.1	–	–	–	–	–	–
	2	141	53.8	52.8	52.5	3.60	3.66	3.68	1.80	1.83	1.84
	4	144	22.2	18.2	18.0	8.73	10.61	10.73	2.18	2.65	2.68
	8	159	10.1	9.3	8.1	19.20	20.76	23.84	2.40	2.60	2.98
	16	175	8.8	6.3	5.4	22.03	30.65	35.76	1.38	1.92	2.23
1344	2	141	89.6	88.8	88.7	–	–	–	–	–	–
	4	146	29.8	28.6	27.2	6.01	6.21	6.52	1.50	1.55	1.63
	8	164	13.1	12.2	11.4	13.68	14.56	15.56	1.71	1.82	1.95
	16	180	9.4	7.1	6.0	19.06	25.01	29.57	1.19	1.56	1.85
	32	239	15.8	9.2	6.4	11.34	19.30	27.72	0.35	0.60	0.87
2304	4	126	52.8	51.5	51.2	–	–	–	–	–	–
	8	147	20.5	19.1	18.4	10.30	10.79	11.13	1.29	1.35	1.39
	16	173	14.4	10.6	9.3	14.67	19.43	22.02	0.92	1.21	1.38
	32	231	19.9	10.7	8.0	10.61	19.25	25.60	0.33	0.60	0.80
	64	391	66.8	28.8	12.4	3.16	7.15	16.52	0.05	0.11	0.26
3520	4	160	131.7	130.1	129.8	–	–	–	–	–	–
	8	183	46.8	43.9	43.6	11.26	11.85	11.91	1.41	1.48	1.49
	16	209	26.6	19.6	18.6	19.80	26.55	27.91	1.24	1.66	1.74
	32	277	32.7	19.0	15.4	16.11	27.39	33.71	0.50	0.86	1.05
4992	8	189	75.0	71.6	71.1	–	–	–	–	–	–
	16	216	36.0	28.4	27.2	16.67	20.17	20.91	1.04	1.26	1.31
	32	285	41.1	23.4	19.9	14.60	24.48	28.58	0.46	0.76	0.89
	64	491	125.4	40.2	24.7	4.78	14.25	23.03	0.07	0.22	0.36
8704	8	169	147.3	143.1	142.6	–	–	–	–	–	–
	16	199	57.7	49.6	48.5	20.42	23.08	23.52	1.28	1.44	1.47
	32	266	55.6	32.1	28.9	21.19	35.66	39.47	0.66	1.11	1.23
	64	460	155.8	44.4	33.5	7.56	25.78	34.05	0.12	0.40	0.53
	128	910	886.4	143.2	42.6	1.33	7.99	26.78	0.01	0.06	0.21
13440	16	243	127.4	114.0	112.8	–	–	–	–	–	–
	32	323	102.9	67.0	63.2	19.81	27.22	28.56	0.62	0.85	0.89
	64	555	248.8	77.2	57.9	8.19	23.63	31.17	0.13	0.37	0.49

n	p	#It	T_g^H	T_g^K	T_g^C	S^H	S^K	S^C	E^H	E^K	E^C
	16	251	205.5	189.8	188.5	–	–	–	–	–	–
19200	32	331	143.1	97.9	93.9	22.98	31.02	32.12	0.72	0.97	1.00
	64	564	306.0	91.2	76.8	10.75	33.30	39.27	0.17	0.52	0.61
	128	1126	1659.3	222.0	98.3	1.98	13.68	30.68	0.02	0.11	0.24
	32	297	226.9	171.5	167.9	–	–	–	–	–	–
33792	64	514	391.3	123.5	106.6	18.56	44.44	50.40	0.29	0.69	0.79
	128	1040	2042.4	234.4	120.1	3.56	23.41	44.74	0.03	0.18	0.35
52480	64	617	660.4	254.2	234.2	–	–	–	–	–	–
	128	1405	3521.2	432.3	277.4	12.00	37.63	54.03	0.09	0.29	0.42
75264	64	635	879.8	374.5	353.9	–	–	–	–	–	–
	128	1348	4073.5	487.4	339.2	13.82	49.18	66.77	0.11	0.38	0.52
102144	64	603	–	490.3	470.9	–	–	–	–	–	–
	128	1233	–	531.8	396.8	–	59.01	75.95	–	0.46	0.59

In Abbildung 17 sind die auf 32, 64 und 128 Prozessoren erreichten Effizienzen grafisch dargestellt:

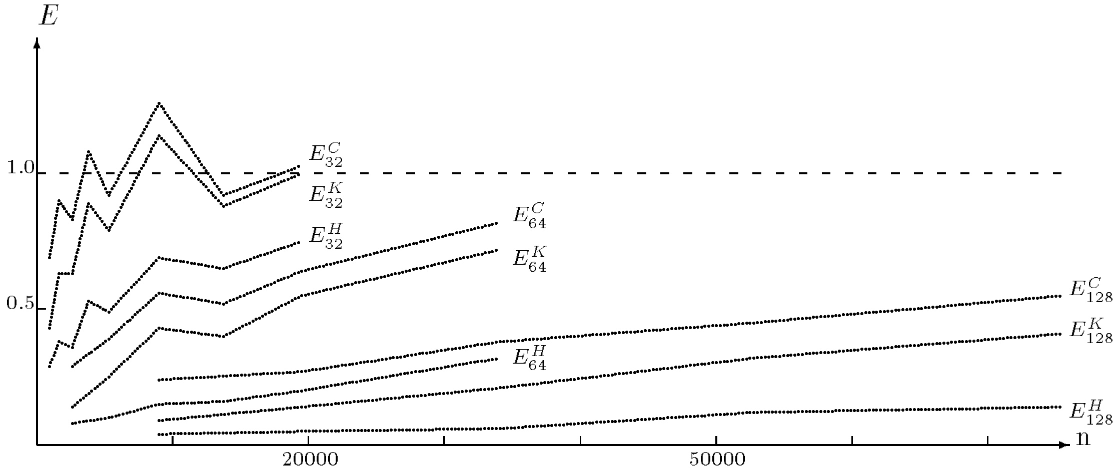

Abbildung 17: Effizienzen E^H, E^K und E^C (Streifengebiet,p=32,p=64 u. p=128)

Der Grafik der Effizienzen auf 16 Prozessoren in Abbildung 18 ist ein anderer Maßstab als in Abbildung 17 zugrunde gelegt:

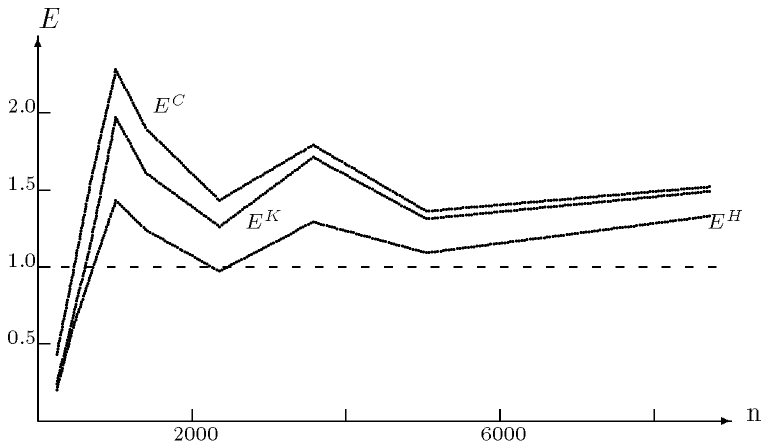

Abbildung 18: Effizienzen E^H, E^K und E^C (Streifengebiet, p=16)

Schließlich sind, wiederum mit verändertem Maßstab, in Abbildung 19 die Effizienzen auf 8 Prozessoren dargestellt. Die obere Linie entspricht E^C, die mittlere E^K und die untere E^H:

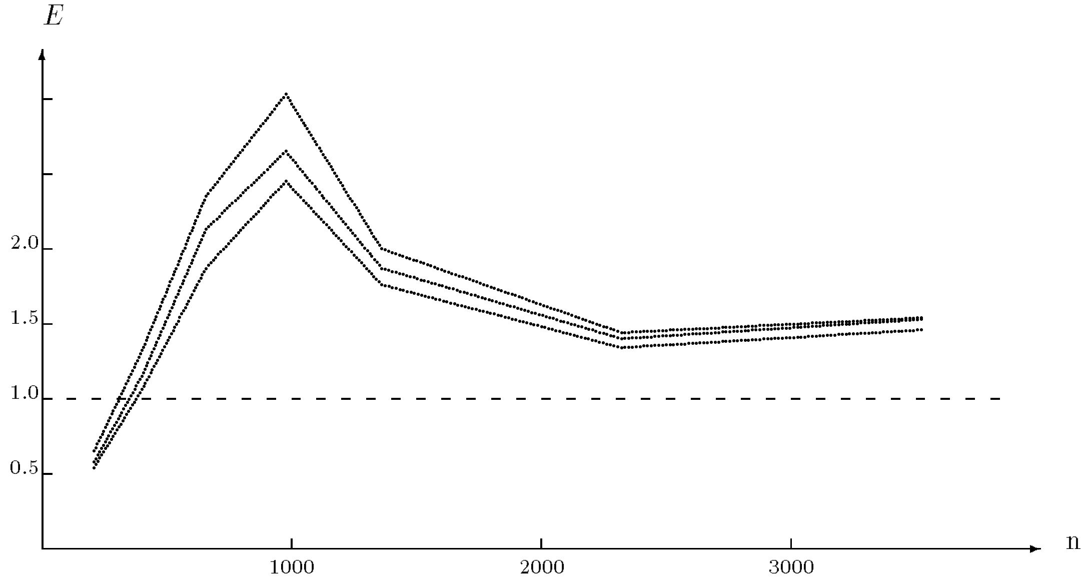

Abbildung 19: Effizienzen E^H, E^K und E^C (Streifengebiet, p=8)

Bei den Rechnungen zur zweiten betrachteten Testaufgabe ergaben sich folgende Speed Up–Werte:

Tabelle 11: Speed Up (Keilgebiet)

n	p	#It	T_g^H	T_g^K	T_g^C	S^H	S^K	S^C	E^H	E^K	E^C
142	1	21	1.2	1.1	1.1	–	–	–	–	–	–
	2	25	1.0	0.9	0.7	1.20	1.22	1.57	0.60	0.61	0.79
	4	26	0.6	0.6	0.6	2.00	1.83	1.83	0.50	0.46	0.46
	8	33	0.7	0.6	0.6	1.71	1.83	1.83	0.21	0.23	0.23
	16	41	0.7	0.8	0.7	1.71	1.38	1.57	0.11	0.09	0.10
310	1	35	4.9	4.6	4.7	–	–	–	–	–	–
	2	38	2.4	2.4	2.1	2.04	1.92	2.24	1.02	0.96	1.12
	4	37	1.4	1.4	1.5	3.50	3.29	3.13	0.88	0.82	0.78
	8	41	1.2	1.1	1.0	4.08	4.18	4.70	0.51	0.52	0.59
	16	50	1.4	1.1	1.2	3.50	4.18	3.92	0.22	0.26	0.24
542	1	31	8.8	8.7	8.7	–	–	–	–	–	–
	2	35	3.5	3.3	3.3	2.51	2.64	2.64	1.26	1.32	1.32
	4	32	1.9	1.8	1.9	4.63	4.83	4.58	1.16	1.21	1.14
	8	41	1.6	1.4	1.3	5.50	6.21	6.69	0.69	0.78	0.84
	16	50	1.7	1.3	1.2	5.18	6.69	7.25	0.32	0.42	0.45
	32	62	3.3	2.0	1.5	2.67	4.35	5.80	0.08	0.14	0.18
	64	81	6.2	3.8	1.5	1.42	2.29	5.80	0.02	0.04	0.09

n	p	#It	T_g^H	T_g^K	T_g^C	S^H	S^K	S^C	E^H	E^K	E^C
1198	1	49	42.5	42.3	42.3	–	–	–	–	–	–
	2	55	14.5	13.9	13.8	2.93	3.04	3.07	1.47	1.52	1.53
	4	44	6.3	6.2	6.0	6.75	6.82	7.05	1.69	1.71	1.76
	8	52	3.8	3.3	3.3	11.18	12.82	12.82	1.40	1.60	1.60
	16	61	3.8	2.6	2.4	11.18	16.27	17.62	0.70	1.02	1.10
	32	75	6.2	3.0	2.4	6.85	14.10	17.62	0.21	0.44	0.55
	64	91	9.9	5.1	2.3	4.29	8.29	18.39	0.07	0.13	0.29
2110	2	51	27.2	27.2	27.0	–	–	–	–	–	–
	4	39	11.2	11.0	10.9	4.86	4.95	4.95	1.21	1.24	1.24
	8	50	5.6	5.0	5.0	9.71	10.88	10.80	1.21	1.36	1.35
	16	59	5.0	3.5	3.5	10.88	15.54	15.43	0.68	0.97	0.96
	32	75	7.1	4.0	3.2	7.66	13.60	16.88	0.24	0.43	0.53
	64	95	13.3	5.6	2.8	4.09	9.71	19.29	0.06	0.15	0.30
	128	122	41.0	16.0	3.1	1.33	3.40	17.42	0.01	0.03	0.14
3278	4	49	26.3	25.9	25.8	–	–	–	–	–	–
	8	58	11.2	10.1	10.3	9.39	10.26	10.02	1.17	1.28	1.25
	16	69	8.6	6.9	6.8	12.23	15.01	15.18	0.76	0.94	0.95
	32	88	10.9	6.2	5.7	9.65	16.71	18.11	0.30	0.52	0.57
	64	106	19.7	7.4	4.7	5.34	14.00	21.96	0.08	0.22	0.34
4702	4	50	44.2	43.8	43.6	–	–	–	–	–	–
	8	61	16.9	15.8	15.7	10.46	11.09	11.11	1.31	1.39	1.39
	16	71	11.9	9.9	9.8	14.86	17.70	17.80	0.93	1.11	1.11
	32	91	13.6	7.7	7.0	13.00	22.75	24.91	0.41	0.71	0.78
	64	110	23.5	8.4	5.7	7.52	20.86	30.60	0.12	0.33	0.48
	128	147	73.4	20.8	5.2	2.41	8.42	33.54	0.02	0.07	0.26
8318	8	56	31.2	29.9	29.9	–	–	–	–	–	–
	16	70	21.6	19.0	18.7	11.56	12.59	12.79	0.72	0.79	0.80
	32	88	18.9	11.1	11.0	13.21	21.55	21.75	0.41	0.67	0.68
	64	112	32.8	10.9	8.3	7.61	21.94	28.82	0.12	0.34	0.45
	128	149	98.5	22.2	6.6	2.53	10.77	36.24	0.02	0.08	0.28
10510	8	64	51.1	49.4	49.4	–	–	–	–	–	–
	16	74	32.5	29.0	28.9	12.58	13.63	13.67	0.79	0.85	0.85
	32	96	26.8	17.4	16.6	15.25	22.71	23.81	0.48	0.71	0.74
	64	120	41.5	15.1	12.2	9.85	26.17	32.39	0.15	0.41	0.51
12958	16	79	44.0	39.8	40.0	–	–	–	–	–	–
	32	102	33.2	21.9	21.6	21.20	29.08	29.63	0.66	0.91	0.93
	64	127	49.6	18.8	15.5	14.19	33.87	41.29	0.22	0.53	0.65
	128	172	143.7	29.3	12.6	4.90	21.73	50.79	0.04	0.17	0.40
18622	16	83	73.3	68.1	68.3	–	–	–	–	–	–
	32	106	47.1	33.1	32.0	24.90	32.92	34.15	0.78	1.03	1.07
	64	131	64.2	24.6	22.4	18.27	44.29	48.79	0.29	0.69	0.76
	128	177	178.0	32.4	15.1	6.59	33.63	72.37	0.05	0.26	0.57
33022	32	100	79.5	60.8	60.6	–	–	–	–	–	–
	64	128	95.3	42.4	39.6	26.70	45.92	48.97	0.42	0.72	0.77
	128	169	228.8	37.9	21.4	11.12	51.37	90.62	0.09	0.40	0.71
41758	32	108	118.1	95.2	95.0	–	–	–	–	–	–
	64	138	129.4	64.8	61.9	29.21	47.01	49.11	0.46	0.73	0.77
	128	185	288.9	51.6	33.4	13.08	59.04	91.02	0.10	0.46	0.71
62302	64	154	205.3	116.1	113.3	–	–	–	–	–	–
	128	204	396.8	72.1	52.1	33.11	103.1	139.2	0.26	0.81	1.09
74110	64	152	240.2	143.8	141.0	–	–	–	–	–	–
	128	201	432.2	81.0	61.3	35.57	113.6	147.2	0.28	0.89	1.15

In den Abbildungen 20, 21 und 22 sind die Effizienzen aus Tabelle 11 grafisch dargestellt, wobei in jeder Abbildung ein anderer Maßstab gewählt wurde:

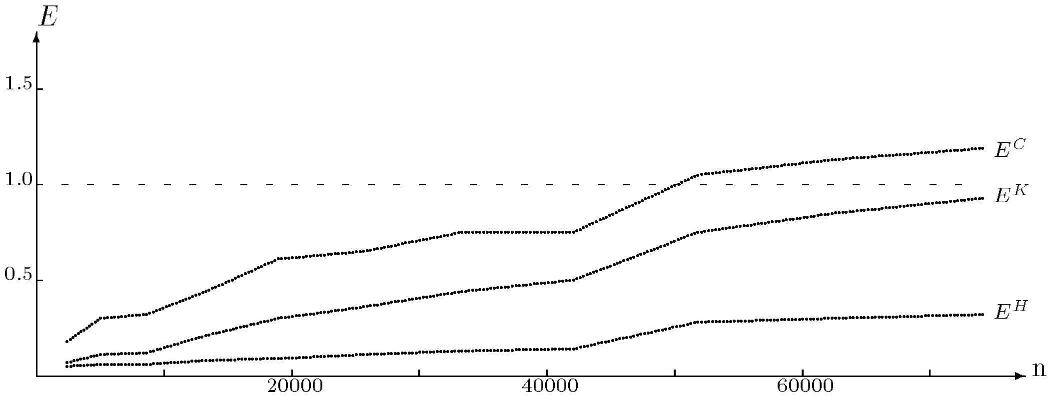

Abbildung 20: Effizienzen E^H, E^K und E^C (Keilgebiet, p=128)

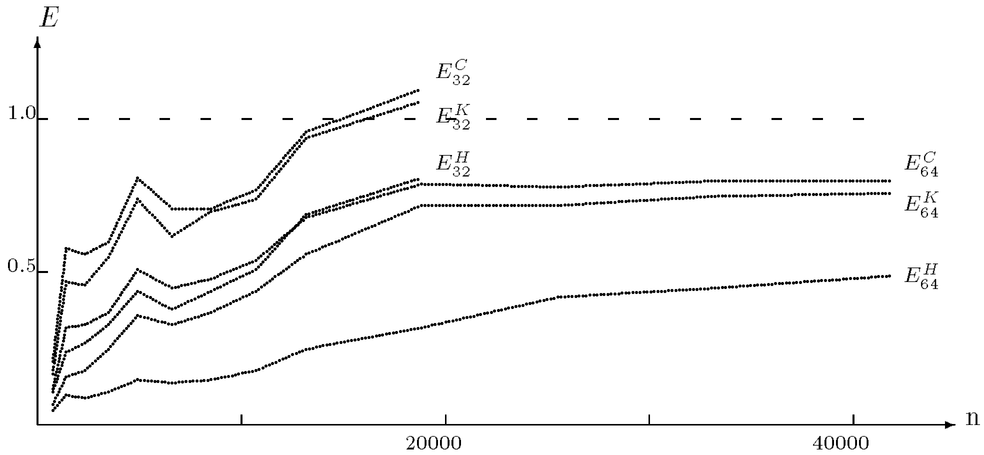

Abbildung 21: Effizienzen E^H, E^K und E^C (Keilgebiet, p=64 u. p=32)

Die ermittelten Daten zeigen, daß die Crosspointkommunikation in allen Fällen effizienter ist als die Koppelkantenkommunikationstechnologie und diese wiederum eine größere Effizienz besitzt, als die Hypercubekommunikation. Für die Effizienzunterschiede gilt $E^C - E^K < E^K - E^H$, d.h., der Effektivitätszuwachs beim Übergang von der Hypercube– zur Koppelkantenkommunikation ist größer als der beim Übergang von der Koppelkanten– zur Crosspointkommunikation. Letzterer wird erst bei Verwendung von wenigstens 64 Prozessoren wirklich bedeutsam.

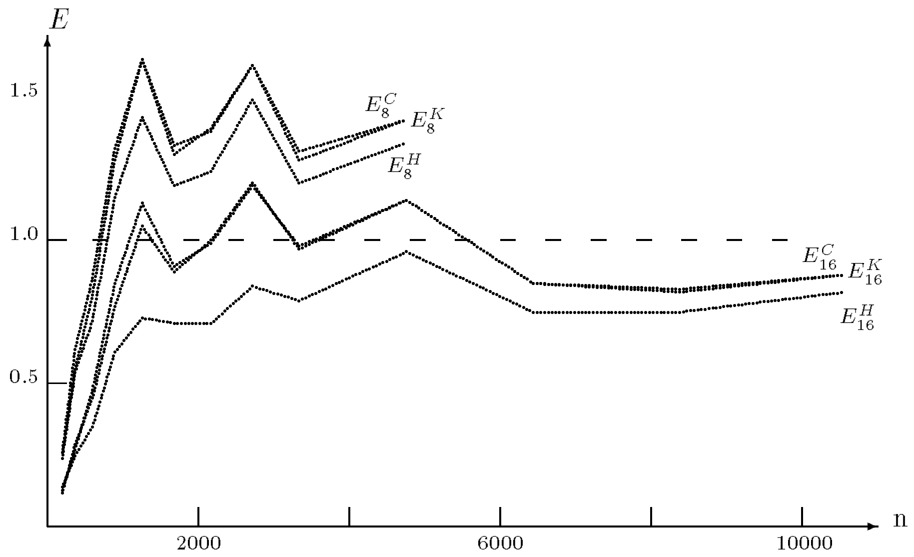

Abbildung 22: Effizienzen E^H, E^K und E^C (Keilgebiet, p=16 u. p=8)

Literatur

[1] G.Haase,T.Hommel,A.Meyer und M.Pester,
Bibliotheken zur Entwicklung paralleler Algorithmen,
Preprint Nr. SPC 94_4, Fakultät f. Mathematik, TU Chemnitz-Zwickau

[2] A.Meyer,
A Parallel Preconditioned Conjugate Gradient Method Using Domain Decomposition and Inexact Solvers on Each Subdomain,
Computing 45, 1990.

[3] R.L.Taylor,
FEAP - A finite element analysis program, Description and Users-Manual,
University of California, Berkeley,1990

[4] S.Meynen und P.Wriggers,
Tätigkeitsbericht Darmstadt zum Forschungsvorhaben Wr19/5-1
Parallele Iterationsverfahren,
Technische Hochschule Darmstadt, Institut für Mechanik, Juli 1994

[5] M.Meisel und A.Meyer,
Implementierung eines parallelen vorkonditionierten Schur–Komplement CG–Verfahrens in das Programmpaket FEAP,
Preprint Nr. SPC 95_2, Fakultät f. Math., TU Chemnitz-Zwickau, Januar 1995