Laboratory for Parallel Programming
Prof. Dr. Felix Wolf

TECHNISCHE UNIVERSITÄT DARMSTADT

Sergei Shudler, Sebastian Rinke

Systemnahe und Parallele Programmierung (WS 17/18)

Praktikum: MPI

Die Lösungen müssen bis zum 16. Januar 2018 in Moodle submittiert werden. Anschließend müssen Sie Ihre Lösungen einem Tutor vorführen. Alle Programmieraufgaben müssen in C und mit MPI auf dem Lichtenberg Cluster kompiliert und ausgeführt werden können. Alle Lösungen müssen zusammen in einer `tar` Datei eingereicht werden.

Machen Sie bitte in Ihrer Abgabe die Gruppennummer und die Namen der Mitglieder der Gruppe ersichtlich. Dies kann zum Beispiel direkt in der Lösung oder durch eine separate Textdatei als Teil Ihrer Abgabe erfolgen.

Um MPI auf dem Cluster zu nutzen laden Sie bitte vorher folgende Module mit:

- `module add gcc`
- `module add openmpi/gcc`

In diesem Praktikum implementieren Sie verschiedene Varianten der parallelen Multiplikation zweier Matrizen A und B. Der Einfachheit halber können Sie annehmen, dass A, B und die Ergebnismatrix C quadratisch sind, d.h., $A,B,C \in \mathbb{R}^{n\times n}$:

$$A=\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix} B=\begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{pmatrix} C=\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & \cdots & c_{n-1,n-1} \end{pmatrix}$$

Formal ist die Matrixmultiplikation definiert als ($0 \leq i, j \leq n-1$):

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} \cdot b_{k,j}$$

Aufgabe 1

(15 Punkte) Angenommen Sie haben p Prozesse und die Matrizen A,B,C, so dass nur zwei Matrizen zu einem Zeitpunkt gleichzeitig im Speicher eines Prozesses gehalten werden können. Dies bedeutet, dass wenn der Root-Prozess (Prozess mit Rang 0) zu Beginn A und B speichert, er nicht die Ergebnismatrix C speichern kann ohne A oder B zu löschen bzw. zu überschreiben. Nehmen Sie weiterhin an, dass alle Matrizen im Row-Major Format gespeichert sind (d.h., die Zeilen der gleichen Matrix sind konsekutiv im Speicher abgelegt) und, dass p ein Teiler von n ist.

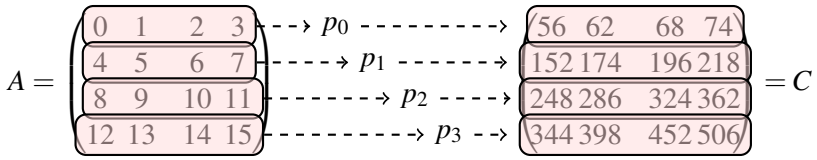
Wir können feststellen, dass wenn wir einen Block aus mehreren Zeilen von A mit der Matrix B multiplizieren, das Ergbnis mehrere Zeilen von C ist. Im folgenden Beispiel multiplizieren wir drei Zeilen von A mit B und erhalten die entsprechenden drei Zeilen in C:

$$\begin{pmatrix} a_{i,0} & a_{i,1} & \cdots & a_{i,n-1} \\ a_{i+1,0} & a_{i+1,1} & \cdots & a_{i+1,n-1} \\ a_{i+2,0} & a_{i+2,1} & \cdots & a_{i+2,n-1} \end{pmatrix} \cdot \begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1,0} & b_{n-1,1} & \cdots & b_{n-1,n-1} \end{pmatrix} = \begin{pmatrix} c_{i,0} & c_{i,1} & \cdots & c_{i,n-1} \\ c_{i+1,0} & c_{i+1,1} & \cdots & c_{i+1,n-1} \\ c_{i+2,0} & c_{i+2,1} & \cdots & c_{i+2,n-1} \end{pmatrix}$$

Die Lösung dieser Aufgabe basiert auf obigem Ansatz. Wir teilen die Matrix A in Blöcke auf und senden jeweils einen Block und die Matrix B an jeden Prozess. Jeder Prozess berechnet dann seinen

entsprechenden Block der Matrix C. Die Ergebnisblöcke werden am Ende auf dem Root-Prozess als Ergebnismatrix C gesammelt.

Die folgende Abbildung zeigt diesen Ansatz. In dem Beispiel ist $n = 4$, $p = 4$ und die Werte der Koeffizienten in A und B sind $a_{i,j} = b_{i,j} = i \cdot n + j$ $(0 \leq i, j \leq n-1)$. Jeder Prozess empfängt einen Block von A, was in diesem Fall nur eine Zeile ist, und die Matrix B. Er multipliziert seinen Block von A mit B und erzeugt damit seinen Block von C. Alle Blöcke von C werden schließlich auf dem Root-Prozess gesammelt:



Die Implementierung besteht aus mehreren Teilaufgaben:

1. **(2 Punkte)** Initialisieren Sie die Matrizen A und B auf dem Root-Prozess mit Hilfe der Funktion `init_mat`. Diese Funktion initialisiert jedes Element entsprechend: $a_{i,j} = b_{i,j} = i \cdot n + j$ $(0 \leq i, j \leq n-1)$.
2. **(3 Punkte)** Nutzen Sie die Funktion `MPI_Scatter` um die Blöcke von A auf die Prozesse aufzuteilen.
3. **(2 Punkte)** Verwenden Sie eine Broadcast Funktion um B zu allen Prozessen zu senden.
4. **(2 Punkte)** Multiplizieren Sie den Block von A mit B auf jedem Prozess mit der Funktion `mult_mat_fast`.
5. **(3 Punkte)** Sammeln Sie alle Teilblöcke von C von allen Prozessen auf dem Root-Prozess. Aufgrund der oben genannten Speicherbeschränkung kann die Matrix C im Speicher einer der beiden nicht mehr benötigten Matrizen A oder B abgelegt werden.
6. **(3 Punkte)** Implementieren Sie eine Funktion um das Ergebnis der Multiplikation zu verifizieren. Ein einfacher Ansatz ist es zusätzlich die Multiplikation lokal auf dem Root-Prozess auszuführen und das Ergebnis mit der parallelen Variante zu vergleichen. In diesem Fall können Sie die Speicherbeschränkung außer Acht lassen. Vorsicht ist jedoch mit diesem Ansatz bei großen Matrizen geboten, da der Speicher nicht ausreichen könnte.

Nutzen Sie für Ihre Lösung die Vorlage `task1.c`. Die Funktion `mult_mat_fast` kann für die lokale Block-Matrix Multiplikation und die Ergebnisverifikation genutzt werden. Um die Matrizen zu initialisieren können Sie die Funktion `init_mat` nutzen. Als Batch Skript nutzen Sie bitte `batch_job.sh`.

Aufgabe 2

(30 Punkte) Aufgabe 1 nahm an, dass die Eingabematrizen A und B in den Speicher eines Prozesses passen. Häufig findet man aber im Hochleistungsrechnen sehr große Matrizen die nicht mehr im lokalen Speicher eines Prozesses Platz finden. In diesem Fall wird ein anderer Ansatz genutzt der das Thema dieser Aufgabe ist.

Sei die Anzahl Prozesse p eine Quadratzahl und $\sqrt{p}$ ein Teiler von n. Die Matrizen A, B, und C werden gleichmäßig auf ein zwei-dimensionales $\sqrt{p} \times \sqrt{p}$ Gitter von p Prozessen verteilt. Jeder Prozess erhält einen Teil von jeder Matrix. Jede dieser Teilmatrizen bezeichnen wir im Folgenden als *Block*. Jeder Block hat die Länge $q = n/\sqrt{p}$. Das heißt, jeder Prozess speichert jeweils einen $q \times q$ Block von jeder der Matrizen A, B und C. Der Parameter q sei auch der Befehlszeilen Parameter von Ihrem zu entwickelnden Programm.

Die Blöcke der Matrizen A, B und C bezeichnen wir basierend auf ihren Positionen in der Matrix. Zum Beispiel steht $A(0,0)$ für den obersten linken Block der Matrix A. Diese Bezeichnung ist unabhängig davon welcher Prozess den Block tatsächlich speichert. Sie dient nur dazu um die einzelnen

Blöcke einer Matrix zu benennen. In einem Beispiel mit 16 Prozessen hat die Matrix A die folgenden Blöcke:

A

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Nehmen wir nun an, dass jeder Prozess von allen drei Matrizen A, B und C jeweils die Blöcke mit der gleichen Position speichert. Zum Beispiel sind die Blöcke $A(0,0)$, $B(0,0)$ und $C(0,0)$ auf dem gleichen Prozess. Analog gilt dies für die anderen Blockkoordinaten. Die Berechnung des Blocks $C(0,0)$ der Matrix C kann nun wie folgt formuliert werden:

$$C(0,0) = A(0,0) \cdot B(0,0) + A(0,1) \cdot B(1,0) + A(0,2) \cdot B(2,0) + A(0,3) \cdot B(3,0)$$

Der Prozess der den Block $C(0,0)$ speichert hält auch die Blöcke $A(0,0)$ and $B(0,0)$. Er kann somit zunächst $A(0,0) \cdot B(0,0)$ berechnen und das Zwischenergebnis in $C(0,0)$ speichern. Um auch die anderen Blöcke von A und B für die Berechnung von $C(0,0)$ zu erhalten werden nun schrittweise alle Blöcke von A horizontal nach links und alle Blöcke von B vertikal nach oben im $\sqrt{p} \times \sqrt{p}$ Prozessgitter verschoben. Blöcke die dabei am linken bzw. oberen Rand des Gitters sind werden zyklisch wieder rechts bzw. unten eingefügt. Nach $\sqrt{p} - 1$ Verschiebungen, d.h., drei in unserem Beispiel, und den entsprechenden Berechnungen steht das Ergebnis in Block $C(0,0)$ fest. Dies beschreibt das grundsätzliche Vorgehen des Algorithmus. Jedoch sind die initialen Positionen von jedem Block, d.h. mit welchen drei Blöcken jeder Prozess startet, von großer Bedeutung. Betrachten wir die Multiplikation von Block $C(1,2)$:

$$C(1,2) = A(1,0) \cdot B(0,2) + A(1,1) \cdot B(1,2) + A(1,2) \cdot B(2,2) + A(1,3) \cdot B(3,2)$$

Der Prozess der $C(1,2)$ speichert besitzt auch ursprünglich $A(1,2)$ and $B(1,2)$, aber die Kombination $A(1,2) \cdot B(1,2)$ ist nicht in der Berechnung von $C(1,2)$ enthalten. Wenn der Prozess jedoch mit der Kombination $A(1,3) \cdot B(3,2)$ beginnen würde, dann funktioniert der beschriebene Algorithmus. Nach einmaligem Verschieben erhält der Prozess dann die Faktoren für $A(1,0) \cdot B(0,2)$ und nach der zweiten Runde erhält er die Faktoren für $A(1,1) \cdot B(1,2)$, und so weiter. Wir müssen also die Blöcke im Prozessgitter verschieben bevor die Berechnung nach dem beschriebenen Schema beginnen kann. Genauer gesagt muss für A die i-te Zeile i mal zyklisch nach links geschoben werden. Die i-te Spalte in B muss i mal zyklisch nach oben geschoben werden. Die folgende Abbildung zeigt die Positionen der Blöcke auf dem Prozessgitter nach der initialen Verschiebung. Für die Matrix C muss nichts verschoben werden. Mit dieser Anordnung der Blöcke auf dem Prozessgitter liefert nun der Algorithmus das korrekte Ergebnis.

A

(0,0)	(0,1)	(0,2)	(0,3)
(1,1)	(1,2)	(1,3)	(1,0)
(2,2)	(2,3)	(2,0)	(2,1)
(3,3)	(3,0)	(3,1)	(3,2)

B

(0,0)	(1,1)	(2,2)	(3,3)
(1,0)	(2,1)	(3,2)	(0,3)
(2,0)	(3,1)	(0,2)	(1,3)
(3,0)	(0,1)	(1,2)	(2,3)

Um den Algorithmus zu implementieren werden noch folgende Funktionen benötigt:

- `MPI_Cart_create` – Erstellt einen neuen MPI Kommunikator der Informationen über eine kartesische Topologie enthält. Jeder Prozess in dem neuen Kommunikator hat kartesische Koordinaten (x,y). Damit kann er einfacher Nachrichten an seine Nachbarn im kartesischen Gitter senden. Zum Beispiel erleichert es eine Nachricht an den Prozess mit den Koordinaten $(x+1,y)$ zu senden.

  Weitere Informationen: `http://mpi.deino.net/mpi_functions/MPI_Cart_create.html`

- `MPI_Cart_coords` – Übersetzt den Rang des Prozesses in seine kartesischen Koordinaten (x,y). Üblicherweise wird diese Funktion nach `MPI_Cart_create` und mit dem Rang im neuen Kommunikator aus `MPI_Comm_rank` gerufen. Die Koordinaten (x,y) entsprechen in diesem Praktikum den Koordinaten der Blöcke $A(x,y)$, $B(x,y)$ und $C(x,y)$ die jeder Prozess zunächst am Anfang speichert.

  Weitere Informationen: `http://mpi.deino.net/mpi_functions/MPI_Cart_coords.html`

- `MPI_Cart_shift` – Nutzen wir mit einem kartesischen Kommunikator um die Ränge eines Quell- und Zielprozesses für das Empfangen und Senden von Nachrichten zu erhalten. Unter Angabe der Dimension in der kartesischen Topologie, der Richtung und der Entfernung zum Quell- und Zielprozess, erhalten wir einen Quell- und einen Zielrang. Um eine Nachricht vom Quellprozess zu empfangen und an den Zielprozess zu senden nutzen wir `MPI_Sendrecv`. Dieser Funktion übergeben wir u.a. die beiden Ränge.

  Weitere Informationen: `http://mpi.deino.net/mpi_functions/MPI_Cart_shift.html`

- `MPI_Sendrecv_replace` – Führt ein blockierendes Senden und Empfangen aus. Der gleiche Nachrichtenpuffer wird verwendet. Das heißt, der Pufferinhalt mit der gesendeten Nachricht wird mit der empfangenen Nachricht überschrieben.

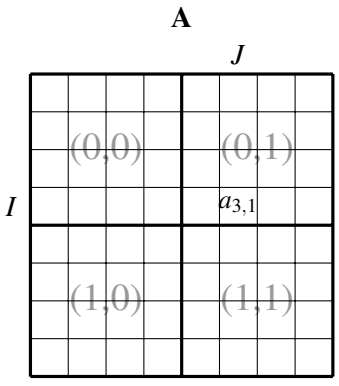
  Weitere Informationen: `http://mpi.deino.net/mpi_functions/MPI_Sendrecv_replace.html`

Der Algorithmus wird in mehreren Teilaufgaben implementiert:

1. **(2 Punkte)** Erstellen Sie einen zwei-dimensionalen $\sqrt{p} \times \sqrt{p}$ kartesischen Kommunikator mit periodischen Dimensionen.

2. **(3 Punkte)** Erstellen Sie die drei lokalen $q \times q$ Blöcke für die Matrizen A, B und C. Der Datentyp ist `double` und q ist der Eingabeparameter des Programms. Modifizieren Sie die Funktion `init_mat`, so dass die Elemente der Blöcke von A und B basierend auf ihren Positionen in der gesamten Matrix A bzw. B initialisiert werden. Das heißt, Sie müssen jedes Element $a_{i,j}$ und $b_{i,j}$ mit den lokalen Blockindizes $0 \leq i,j \leq q-1$ basierend auf deren globalen Indizes $0 \leq I,J \leq n-1$ mit $a_{i,j} = b_{i,j} = I \cdot n + J$ initialisieren. Nutzen Sie dazu die Größe und die Koordinaten des Blocks.

   Das folgende Beispiel zeigt den Unterschied zwischen lokalen und globalen Indizes. Wir haben vier Blöcke mit $q = 4$. Das Element $a_{3,1}$ in Block $A(0,1)$ hat die blocklokalen Indizes $i = 3$ und $j = 1$. Betrachten wir das Element als Teil der gesamten Matrix A, so sind seine globalen Indizes $I = 3$ und $J = 5$.



3. **(7 Punkte)** Implementieren Sie die initiale Verschiebung der Blöcke wie oben beschrieben. Zuerst sollte jeder Prozess `MPI_Cart_shift` verwenden um den Empfänger seines Blocks und den

Sender des für ihn bestimmten Blocks zu ermitteln. Nutzen Sie dann die erhaltenen Ränge mit `MPI_Sendrecv_replace` um jeweils einen Block zu senden und zu empfangen.

4. **(3 Punkte)** Damit die Blöcke in A nach links und die Blöcke in B nach oben schrittweise rotiert werden können nutzen Sie zunächst `MPI_Cart_shift` um die Nachbarn in den zwei Dimensionen im kartesischen Gitter zu bestimmen.

5. **(2 Punkte)** Nutzen Sie `mult_mat_fast` um die zwei Blöcke aus A und B die nach der initialen Verschiebung erhalten wurden lokal zu multiplizieren.

6. **(8 Punkte)** Implementieren Sie die Iterationen des Algorithmus. In jeder Iteration werden zunächst A und B um jeweils einen Block rotiert und anschließend die neuen Blöcke lokal mit `mult_mat_fast` multipliziert. Nutzen Sie `MPI_Sendrecv_replace` um A und B auf dem Prozessgitter zu rotieren. Die Ränge von Sender und Empfänger haben Sie bereits weiter oben vorher mit `MPI_Cart_shift` bestimmt.

7. **(5 Punkte)** Dadurch, dass Sie die Funktion `init_mat` in dieser Aufgabe weiter oben bereits modifiziert haben, werden die Elemente der Blöcke basierend auf ihren globalen Indizes initialisiert: $a_{I,J} = I \cdot n + J$ und $b_{I,J} = I \cdot n + J$. Das Ergebnis der Multiplikation ist demnach für jedes Element $c_{I,J}$ in einem Block von C: $c_{I,J} = \sum_{k=0}^{n-1} (I \cdot n + k) \cdot (k \cdot n + J)$. Dieser Term kann umgeformt werden zu

$$\begin{aligned} c_{I,J} &= \sum_{k=0}^{n-1} (In^2k + IJn + nk^2 + Jk) \\ &= In^2 \sum_{k=0}^{n-1} k + IJn^2 + n \sum_{k=0}^{n-1} k^2 + J \sum_{k=0}^{n-1} k. \end{aligned}$$

   Vereinfachen Sie diesen Term weiter indem Sie alle Summen auflösen. Nutzen Sie den erhaltenen Ausdruck um effizient Werte $c_{I,J}$ zu berechnen und die Korrekheit Ihres Algorithmus zu verifizieren. Die Verifikation soll keine Matrixelemente kommunizieren und auch keinen zusätzlichen Speicher allokieren. Nutzen Sie `MPI_Reduce` um die Resultate von allen Ergebnisvergleichen im Root-Prozess zu aggregieren.

   Beachten Sie, dass dieser Ansatz nur für kleine n, d.h., $n \leq 1000$ funktioniert. Für größere n führen Rundungsfehler zu signifikanten Unterschieden zwischen dem Ergebnis und dem Vergleichswert.

Aufgabe 3

(10 Punkte) In dieser Aufgabe werden Sie das Tool Extra-P verwenden um ein Leistungsmodell für die Laufzeit der Lösung von Aufgabe 1 zu bestimmen.

Um Extra-P zu nutzen müssen Sie zuerst mehrere Messungen für unterschiedliche Werte mit Ihrem Zielparameter durchführen. Da wir ein Modell für die Laufzeit in Abhängigkeit von der Eingabegröße berechnen möchten, ist unser Zielparameter die Matrixdimension n. Extra-P benötigt Laufzeitmessungen für mindestens fünf unterschiedliche Werte von n. Das Batch Skript `perf_analysis.sh` misst die Zeit von Ihrem Programm für die folgenden Werte von n: 256, 512, 768, 1024, 1280 und 1536. Für jede Eingabe führt das Skript zehn Iterationen aus und schreibt die Ergebnisse in die Datei `input.res`. Diese Datei ist eine Textdatei und die Eingabe für Extra-P. In jeder Iteration wird Ihr Programm mit 16 Prozessen auf einem Knoten ausgeführt. Ihr ausführbares Programm muss dafür `task1` benannt sein. Damit das Skript korrekt funktioniert entfernen Sie bitte nicht die Zeitfunktionen in Ihrem Code. Um nach den Laufzeitmessungen eine Modell mit Extra-P zu berechnen, führen Sie folgende Schritte auf einem Login Knoten aus:

1. `source /home/kurse/kurs00015/da_lpp/modules/loadModules.sh`
2. `module purge`
3. `module load gcc/4.9.4 intel/2016u4 papi/5.5.1 openmpi/intel`
4. `module load scorep/intel/ompi/1.4.2 python`

5. `module load extrap/2.0`

Wir nutzen hier Extra-P auf der Befehlszeile. Eine grafische Oberfläche existiert jedoch auch. Das Modell wird nun durch den Aufruf `extrap_cmd input.res` generiert. Die Ausgabe besteht aus sechs Feldern: <metric name>, <region name>, <model>, <no. terms>, <adj. R-squared>, <SMAPE>. Unser Modell ist im dritten Feld. Beachten Sie, dass der Modellparameter mit p benannt ist, obwohl er n in unserem Fall entspricht. Welches Modell erhalten Sie? Wie unterscheidet es sich von der $O(n^3)$ Komplexität einer einfachen Matrixmultiplikation?

Vim Cheat Sheet for Programmers

Esc Normal — Revision 2.0, Sept. 11, 2011 — Vim 7.3+ `:version`

http://michael.PeopleOfHonorOnly.com/vim/

HOW-TO make Vim not suck Out of the Box: ***:help statusline*** :set nocompatible ruler laststatus=2 showcmd showmode number — ***Search*** :set incsearch ignorecase smartcase hlsearch — ***Remove useless splash screen*** :set shortmess+=I

Best tips: http://vim.wikia.com/ — **Best scripts:** http://www.vim.org/scripts/index.php

:map <F9> :e $HOME/_vimrc<CR> :map <F6> :so $HOME/_vimrc<CR>

	Ctrl `	Ctrl 1	Ctrl @	Ctrl 3	Ctrl 4	Ctrl 5	Ctrl ^	Ctrl 7	Ctrl 8	Ctrl 9	Ctrl 0	Ctrl _	Ctrl =
	~ toggle case	! extern filter	@• play macro	# prev identifier	$ →|	% goto match	^ soft ←	& repeat :s	* next identifier	(begin sentence	) end sentence	_ cur line	+ ↓
	'• goto mark	1	2	3	4	5	6	7	8	9	0 hard ←	- ↑	= auto-format
Tab		14 block select Q ex mode / q• record macro	^w• window… W WORD ↘ / w word ↘	scroll line ↑ E end WORD ↘ / e end word ↘	12 :redo R Replace / r replace char	10 ctags return T• ← until char / t• until char →	scroll line ↓ Y copy line / y copy	half page ↑ U undo line / u undo	Ctrl I I insert ← / i ↹ insert	prev mark O open ⤒ / o open ⤓	9 ↑ P paste ⤒ / p paste ⤓	Normal { ↖ paragraph / [• misc.	ctags identifier } paragraph ↘ /]• misc.
Caps		7 incr. # A append → / a append ↴	S subst line / s subst char	10 half page ↓ D del → / d del	page ↓ F• ← find char / f• find char →	file/cursor info G goto eof / goto line# / g• extra	Ctrl H H Top screen / h ←	15 Ctrl J J Join lines / j ↓	K man page identifier / k ↑	redraw L Bottom screen / l →	Ctrl ; : cmd line / ; "next" f/F/t/T	Ctrl ' "• register / '• goto mark ←	Ctrl \ | goto col# / \•
Ctrl ^		:suspend Z• quit	7,11 decr. # X ← del char	Normal / Cancel C change →	block select V select lines	page ↑ B ↖ WORD	9,16 ↓ N "prev" find	15 Ctrl M M Middle screen	Ctrl , < undent	Ctrl . > indent	Ctrl / ?• find ↖		
Shift ⇧		z• extra	x del char →	c change	v select chars	b ↖ word	n find "next"	m• set mark	, "prev" f/F/t/T	. repeat cmd	/• find ↘		

Unused & Duplicate keys

```
\ Ctrl-K Ctrl-S (free)
Ctrl-L (redraw)
13 ` near dup of '
14 Ctrl-Q = Ctrl-V
15 Ctrl-J = Ctrl-M = ^N
```

16 The search direction is relative; **next** is the initial direction, **previous** is the opposite direction. `n` `;` repeat same initial direction find. `N` `,` repeat opposite initial direction find. **Note**: `;` `,` only searches cursor line, `n` `N` searches buffer.

Legend:

Macro	Register name (0-9a-zA-Z) required
Op	Motion req.; act between cursor & dst
Cmd	Command
Ins	Command and enter insert mode
Move	Moves cursor or defines range for op
Find	Search (↖ = reverse, ↘ = forward)
tag	ctags / diffs / folding
Code	Code formatting, whitespace, etc.
Extra	Extended functionality; req. extra chars
•	Char arg req. `g z Z ^w ' " ` ...`

Modes — `:help modes`

n	Normal	`Esc ^[ ^c`
i	Insert	`a i r s`
v	Visual	`v V ^v ^q`
o	Op pending	`c d y < >`
c	Command Line	`: / ? !`

word `Foo ( src , ß dst , ß len );`

Note: There is no whitespace in-between 'Foo(src,' but before/after 'dst,'.

WORD `Foo ( src , ß dst , ß len );`

Startup

```
vim <filename> +123      goto line 123
vim <file> … -t Foo      edit at tag 'Foo'
vim <file> … -c "/Foo"   cmd: find 'Foo' & edit
vim -g    or    gvim     start GUI ver.
Linux  :set guifont=ProggyTinyTT\ 12
OSX    :set guifont=ProggyTiny\:h11
gvimdiff <file1> <file2> [<file3>]
```

Broken Keys Ctrl-I = Tab; Ctrl-[= ESC

Vim is *still* **unable** to map certain keys for your own use…

Caps, Ctrl-1, Ctrl-Shift-1, Ctrl-I, Ctrl-\, etc.

0 **See**: `src/ops.c -c "/valid_yank_reg"` *for "x reg. names*

6 **See**: `src/normal.c -c "/nv_cmds"` *for g• extra cmds*

11 **See**: `src/edit.c -c "/ctrl_x_msgs"` *for ^x• insert cmds*

:help cmdline

```
:r file  insert file
:w save      :gui  switch to GUI
:q quit      :q!   quit w/o save
:e <file>    edit file
:source %    exec cmds in cur file
:exec '...'  do cmd
```

:help movement

```
soft ^ I← Start of Line  1st non-whitespace
hard 0 I← Start of Line  column 0
     $ →I End of Line
     | move col 0        #| move col #
^b page ↑      ^f page ↓
^u ½ page ↑    ^d ½ page ↓
^e scroll line ↑   ^y scroll line ↓
1g start of file   0g end of file
#g goto line #     G end of file
[[ begin this func {
]] begin next func {
:set matchpairs=(:),{:},[:],<:>,?:\:
 %  goto matching { } < > [ ]
```

:help range

```
:s/Foo/Bar      find Foo replace w/ Bar
:s/Foo/Bar/g    ...all instances on line
:%s/Foo/Bar     apply to whole file
.,.+# cur line, cur line + # lines
  $ last line   '< start of select
                '> end of select
```

:help tags

```
:ts   list active tags
^]    jump to tag under cursor
^t    restore cursor before tag jump
^p    complete word
:ta Foo  manual jump to tag 'Foo'
```

:help diff

```
[c prev diff   :hi DiffAdd    guifg=#rrggbb
]c next diff   :hi DiffChange guibg=#rrggbb
:diffupdate    :hi DiffText   gui=none
  resync       :hi DiffDelete
```

:help folding

```
zR fold remove
zo fold open
zc fold close
zi invert all
zr fold reduce
zm fold more
```

:help changes

```
:changes
g; older change
g, newer change
```

:help syntax

```
:syntax enable
:set filetype=
c cpp sh make perl python
Note: chose only ONE type!
```

convert <eol>

```
:set fileformat=
  unix or dos or mac
  then :w to convert
```

:help recording

```
q* start recording
@* playback
 q  stop recording
@@  repeat
```

Code `= < > << >>`

```
:set backspace=indent,eol,start   allow backspace join lines
:set shiftwidth=#  indent width for ai
:set autoindent!   toggle auto-indent
:set lisp          lisp indent mode
:set tabstop=#     set tab stop every #th col
:set expandtab!    toggle hard/soft tabs
:set listchars=…   tab:>-,trail:-,nbsp:%,eol:$
:set list!         toggle whitespace
:set colorcolumn=80  visible right margin indicator
noremap + :s/^/\/\//<CR>   block comment
```

```
\ :map \ :Explore<CR>  manually type <,C,R,>
§0 "• before del/copy/paste to use register
  "+x  cut to system clipboard reg. '+'
  "+gP paste from system clipboard
 1 Number before any action repeats it
  2p   paste twice      3. repeat thrice
 2 Repeat op to act on current line
  yy   copy line        dd del line
  <<   undent line      >> indent line
 3 #   highlight words under cursor
 4 zz  save & quit      ZQ quit w/o save
 5 zz  center cursor line in window
  zh   scroll left      zl scroll right
  zt   scroll top       zb scroll bottom
§6 gg  top of file
  gf   open file under cursor
 7 ^a  incr # under cursor (Dec / Hex)
  ^x   decr # under cursor (Dec / Hex)
 8 *   start a "new" search
```

Insert mode

```
 9 ^p prev auto-complete  ^n next
10 ^d undent              ^t indent
§11 ^x* ^f filename completion
    ^s spelling           :set spell!
    ^k dictionary         ]s next bad
    ^t thesaurus          :help spell
12 ^r* paste register 0-9a-zA-Z or …
    +  clipboard (or '*')  :help c_CTRL-R
    "  last del/copy       % filename
```

```
:set numbers!   toggle line numbers
:set wrap!      toggle linewrap display
:set showmatch  highlite matching ()
noremap - :s/^\/\///<CR>  uncomment
```

:buffer #

```
:buffers  list
:new  blank file/buffer
:bn   next file
:bp   prev file
:bd   close file
:bd!  force close
:set lines=#
:set columns=#
:winpos # #
```

Windows

```
:help windows
^w* or :wincmd *
 w :switch to next
 c :close!
 n :new
 s :split  horz.
 v :vsplit vertical
 o :only   maximize
 = all same size
 h move to win ←
 j move to win ↓
 k move to win ↑
 l move to win →
:sp [<filename>]
 edit in split window
```

Cursor Bookmarks

```
:marks   ` ' ^ ^o
 ma  mark local 'a'
 'A  goto global 'A'
 ''  prev location
```

File / Directory

```
:Explore  or :e .
:set browsedir=…
one of buffer last
```