探索的財務ビッグデータ解析
―前処理の並列化―

地　道　正　行

要　旨

本稿では，データベース Orbis から抽出された世界の2,400万社を超える企業（一般事業会社）に関する財務データの前処理を行い，その工程を並列化することによって処理時間を短縮することを試みる．その際，前処理の全工程を自動実行することによって，再現可能性を確保する方策についても言及する．

キーワード：財務ビッグデータ（Financial Big Data），探索的データ解析（Exploratory Data Analysis），前処理（Preprocessing），データラングリング（Data Wrangling），並列化（Parallelization），再現可能性（Reproducibility）

I　はじめに

一般に，粗データ（raw data）は，さまざまな理由（フォーマットが不統一，欠測値，特殊記号の存在など）から，そのままではデータ解析環境に読み込む（import, load）ことができず，分析・解析することが難しい場合が多い．このことから，まずは「読み込めるファイル形式」に変換する必要がある．本研究では，この工程を「前処理」（preprocessing）と呼ぶことにする．次に，前処理を行ったデータセットのファイルを何らかのデータ解析環境に読み込み，分析・解析できるオブジェクトに変換する工程が必要となるが，本研究では，RStudio[1] 上でデータ解析環境 R[2] を使って行うため，Wickham and Grolemund（2016）に習って「データラングリング」（data

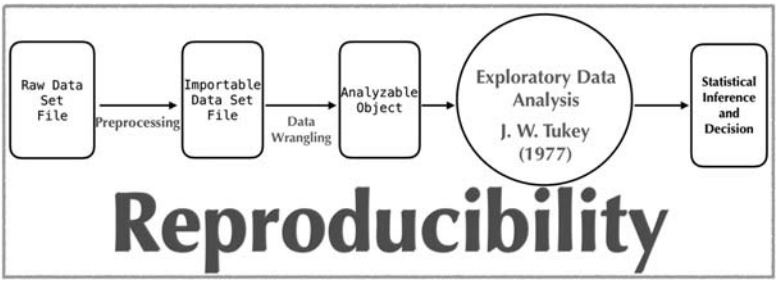

図１　探索的財務ビッグデータ解析の全工程

wrangling）と呼ぶことにする（図１も参照のこと[3]）．

また，近年ほぼ定着したと思われる「データサイエンス」や「ビッグデータ」などの用語が扱われる文献等における経験則として，前処理とデータラングリングの工程は，データを処理・分析・解析する全工程の50％から90％を占める[4] ともいわれ，この工程を如何に効率よく処理するかが問題となる．

本研究では，財務関連の情報を含むデータベースから抽出された規模の大きい粗データファイルを，多様なデータ解析環境で扱えるファイル形式（CSV ファイル）に変換する工程（前処理）について述べ，この工程を再現可能なものとするための試みについて述べる．さらに，前処理を並列化することによって，速度（verosity）を向上させることを試みる．

本稿の構成は次のようなものである．まず，II 節では，本稿で利用するデータベースとデータセットについて述べ，次に III 節で，データセットに関する（通常の）前処理の工程についての詳細を述べる．ここでは，ファイルに対する個々の処理工程を説明すると共に，それらを make コマンドを利用

1） https://rstudio.com/
2） https://www.r-project.org/
3） 本稿では，図１におけるデータラングリングを含むそれ以降の部分，すなわち「探索的データ解析」（Exploratory Data Analysis: EDA）と「統計的推測・決定」（statistical inference and decision）については言及しない．
4） 例えば，Patil（2012）（p. 18）では，
80% of the work in any data project is in cleaning the data.
と述べられている．

し，自動実行することによって，再現可能性を確保する方策についても言及する．さらに，IV 節では，GNU parallel を利用することによって前処理を並列化し，処理時間を短縮することを試みる．最後に，V 節では総括を行う．

II　データベースとデータセット

本研究では，Bureau van Dijk[5]（ビューロー・ヴァン・ダイク）社（以下 BvD と略）のデータベース Orbis（オービス）を利用する．このデータベースには，情報が入手可能な世界の全企業（データ抽出時点で約2.4億社以上を収録）のデータが国際比較可能な統一のフォームで収録されている．このデータベースから，データセットとして，主要財務情報（売上高，総資産など）を（1）連結（consolidated）財務諸表を優先的に抽出した24,014,352社と（2）非連結（un-consolidated）財務諸表を優先的に抽出した24,012,807社の最長10年分抽出したものを利用する．データセットのファイルは，1 個のサイズが 5 GB 程度の25個の TSV ファイル[6]（2 セット）からなり，合計のファイルサイズは，それぞれ約 127 GB，125 GB であり，行数は合計で各 2 億 6 千万行超である．なお，表 1，2 には，それぞれ，データセットの仕様とサイズを与えているので参照されたい．

表 1　データセット：仕様

データセット名	抽出年度	データベース	上場情報	抽出主体	抽出期間	抽出指標数
DS-Orbis-C-2018	2018	0rbis	上場・非上場	連結財務諸表優先	10年	82
DS-Orbis-U-2018	2018	0rbis	上場・非上場	非連結財務諸表優先	10年	82

本稿では，連結財務諸表優先で抽出したデータセット DS-Orbis-C-2018 の前処理について述べる．なお，非連結財務諸表優先で抽出したデータセット

5）BvD Web Page: https://www.bvdinfo.com/en-gb/
6）TSV（Tab Sepelated Values）ファイルとは，項目（カラム）間がタブ区切りのテキスト形式のファイルである．

表2　データセット：サイズ

データセット名	社数	総行数	ファイル	トータルサイズ
DS-Orbis-C-2018	24,014,352社	264,157,872行	01_orbis_2018.asc,...,025_orbis_2018.asc	約127 GB
DS-Orbis-U-2018	24,012,807社	264,140,888行	01_orbis_2018_LU.asc,...,025_orbis_2018_LU.asc	約125 GB

DS-Orbis-U-2018 に対する前処理も同様に行うことができるが，冗長であるので詳細は割愛する．

Ⅲ　前処理

1．オリジナルファイルの複製

データの前処理を行う前に，抽出されたデータセット DS-Orbis-C-2018 のファイル（以下，オリジナルファイルと呼ぶ）に対して直接処理を行わずに，まずはこれらのファイルの複製をとる[7]．このことを実行するために，以下のようなシェル・スクリプト・ファイルを作成した．

ソースコード1　オリジナルファイルの複製を行うスクリプトファイル：**renumbering.sh**

```
#!/bin/bash
for i in $(seq -w 25); do
cp "./rawdata/Orbis_Cons/"$i"_orbis_2018.asc" "./data"$i".txt";
done
```

このスクリプトを実行することによって，オリジナルファイルの複製 data01.txt,...,data25.txt が作成される．この工程の概略を図2に与える．

2．データセットのもつ問題点と解決法

複製されたデータセットファイル data01.txt,...,data25.txt は，データ間の分割符がタブ（\t）区切りのテキストファイル（TSV ファイル）

7）オリジナルのファイルを残しておく必要性は，前処理を含む全工程の再現可能性を確保するための一環でもある．このことは，一般に，前処理は試行錯誤を伴うものであり，誤った処理を行った際にオリジナルファイルが失われると，ベンダーから再納品が必要となるなど，時間と労力，場合によってはコストの問題などが発生する．

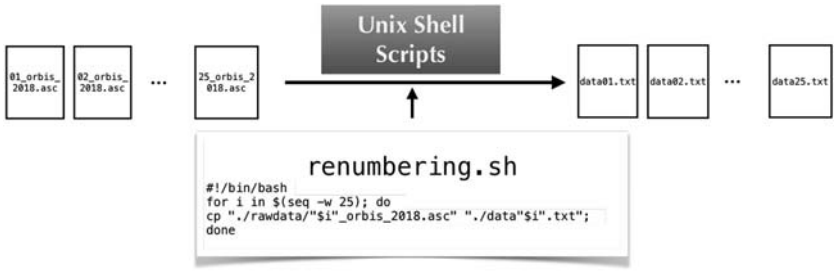

図2　オリジナルファイルの複製

であり，各サイズは約5GB（ギガバイト）・11,000,000行である[8)]．以下にファイル data01.txt の一部（先頭2行）を与える：

ソースコード2　データファイル **data01.txt** の一部

```
1 <U+FEFF>WALMART INC. Country City Postcode Telephone number BvD ID number National ID
    number National ID label National ID type IP identification number IP identification
    label ISIN number Ticker symbol Main exchange Listed/Delisted/Unlisted Consolidation
    code Closing date Number of months Audit status Accounting practice Source (for
    publicly quoted companies) Original units Original currency Exchange rate from
    original currency Fixed assets Intangible fixed assets Tangible fixed assets Other
    fixed assets Current assets Stock Debtors Other current assets Cash & cash equivalent
    Total assets Shareholders funds Capital Other shareholders funds Non-current
    liabilities Long term debt Other non-current liabilities Provisions Current
    liabilities Loans Creditors Other current liabilities Total shareh. funds & liab.
    Working capital Net current assets Enterprise value Number of employees Operating
    revenue (Turnover) Sales Costs of goods sold Gross profit Other operating expenses
    Operating P/L [=EBIT] Financial revenue Financial expenses Financial P/L P/L before
    tax Taxation P/L after tax Extr. and other revenue Extr. and other expenses Extr. and
    other P/L P/L for period [=Net income] Export revenue Material costs Costs of
    employees Depreciation & Amortization Other operating items Interest paid Research &
    Development expenses Cash flow Added value EBITDA US SIC, Primary code(s) Ibid, text
    description US SIC, Secondary code(s) Ibid, text description BvD major sector
    Information provider
2       2008 (th USD) United States of America BENTONVILLE 72716 +1 479 273 4000
          US710415188 71-0415188 EIN VAT/Tax number 9556N Reuters number US9311421039 WMT
           New York Stock Exchange (NYSE) Listed C1 31/01/2009 12 Unqualified Local GAAP
          10-K thousands USD 1 114,480,000 15,260,000 95,653,000 3,567,000 48,949,000
          34,511,000 3,905,000 10,533,000 7,275,000 163,429,000 65,285,000 393,000
          64,892,000 42,754,000 34,549,000 8,205,000 n.a. 55,390,000 6,163,000 28,849,000
           20,378,000 163,429,000 9,567,000 -6,441,000 219,773,662 2,100,000 404,254,000
          404,254,000 297,202,000 107,052,000 84,285,000 22,767,000 284,000 2,184,000
          -1,900,000 20,867,000 7,133,000 13,734,000 n.a. n.a. -353,000 13,381,000 n.a. n
          .a. n.a. 6,739,000 77,546,000 2,184,000 n.a. 20,120,000 n.a. 29,506,000 5331
          Variety stores 5411 Grocery stores Wholesale & retail trade Reuters
```

8)　ファイル data25.txt のみ70MB・140888行である．

各データファイルの構造は，1行のヘッダー部分（ヘッダー行）と10行のデータ部分（データ行）を1ブロックとする1,000,000ブロックから構成される（図3参照）．

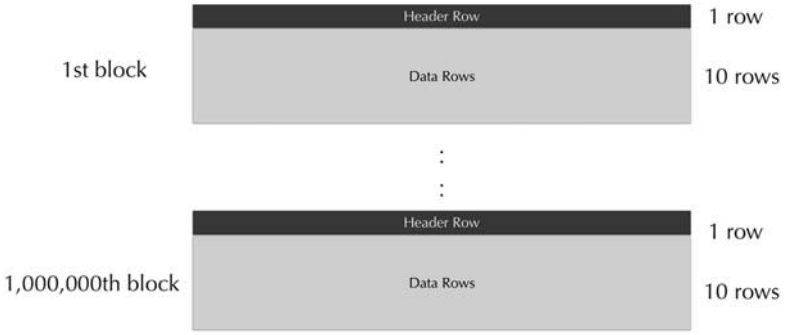

図3　データファイル **data01.txt,...,data24.txt** の構造

これらのデータセットファイルは，地道（2018-a）でも検討したように，ファイルに関する前処理が必要となる．今回のものは，以下のような問題をもつため，そのままではRに読み込むことが難しかった：

（PU1）BOM[9] の存在
（PU2）オペレーティングシステム（OS）間での行末コードの相違
（PU3）レイアウトの不統一（レイアウトが異なったヘッダー行とデータ行の混合）
（PU4）金額に関するフォーマット（カンマ区切り）
（PU5）欠損値コード（`n.a.`,`NA` など）の統一と欠損値コードが存在しない欠損値の存在
（PU6）データ行の先頭に余分なタブコード（`\t`）が存在

9）BOM とは Byte Order Mark の略称であり，Unicode の UTF-16 など16ビット幅のエンコーディング方式において，エンディアンを指定するためにファイルの先頭に記入される16ビットの値である（IT 用語辞典 e-Words `http://e-words.jp/` 参照）．

（PU7）特殊記号の存在（#[10] など）

以上の問題に対処することによって，R に読み込むことができるけれども，データ解析を実行するためには，さらに以下のような問題があった：

（PR1）列（変量）毎の型（数値，文字，年月日など）の指定（構造化）
（PR2）データ部分が収められたファイルには企業名が存在しない（ヘッダー部分に含まれる企業名との結合が必要）
（PR3）データセットの列名[11] の付与
（PR4）不完全なデータが存在（企業分類コード・分類名が各ブロックの先頭のみに収録されている）
（PR5）税金等に関連するデータの符号（マイナスからプラスへ）の変換
（PR6）企業名が一意ではない（同名の企業の存在）
（PR7）通貨換算レートの年のみの情報が存在しない

なお，これらの問題に加えて，本稿で扱っているファイルのサイズが「大きい」ことから，通常のエディタや表計算ソフトウェア[12] では整形が困難である．これらの問題に対して，以下のような手順で処理した：

（S1）UNIX コマンドやインタプリター（`grep`, `dos2unix`, `>`, (g)`sed` など）を利用して整形し，データファイル `data*.txt` ファイルを R に読み込める形式 `name*.rda`（企業名を含むヘッダー部分が収められたテキストファイル），`data*.rda`（データ部分が収められたテキストファイル）へ変換
（S2）データ解析環境 R を用いて処理後，CSV ファイルに変換し保存

10）実際に，“DATA#3 LIMITED” という社名をもつ企業が存在する．
11）ここでの列名は，データベースから抽出するデータセットの列の特性（企業名，決算年月日，財務関連の情報など）を表す．
12）Microsoft Excel 2016 では，2^{20}=1,048,576行を超えるファイルを扱うことはできない．

以下に，これらの工程における処理について述べる．

3．手順（S1）の処理の実行

手順（S1）における処理を，シェル・スクリプト・ファイル（seqscript.sh）に記述し，再現性を確保するためにシェル（/bin/bash）を起動することによって実行した（実行イメージについては図4も参照）：

シェルスクリプトによるデータファイル操作の自動実行

```
$ /bin/bash ./seqscript.sh
```

ソースコード3　シェル・スクリプト・ファイル：**seqscript.sh**

```
1 #!/bin/bash
2 for i in $(seq -w 25); do
3 echo $i
4 /bin/bash ./script.sh "data"$i".txt";
5 done
```

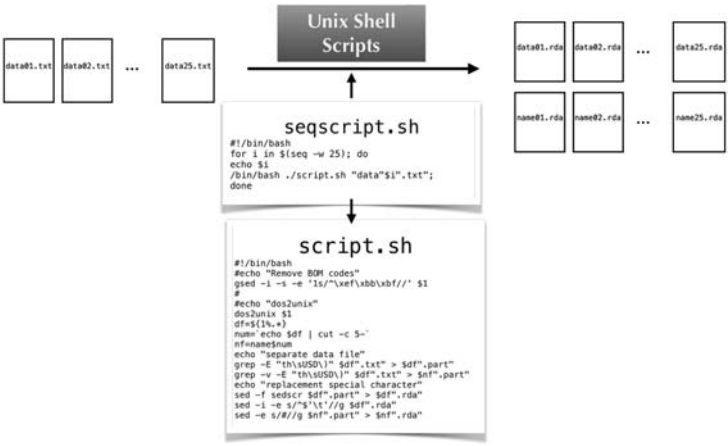

図4　データファイル処理のシェルスクリプトによる自動実行

なお，ソースコード3の4行目で利用されているシェル・スクリプト・ファ

イル script.sh は実際にファイルの処理を行うためのもので，ソースコード 4 で与えられる．

ソースコード 4　シェル・スクリプト・ファイル：**script.sh**

```
#!/bin/bash
#echo "Remove BOM codes"
gsed -i -s -e '1s/^\xef\xbb\xbf//' $1
#
#echo "dos2unix"
dos2unix -f $1
df=${1%.*}
num=`echo $df | cut -c 5-`
nf=name$num
echo "separate␣data␣file"
grep -E "th\sUSD\)" $df".txt" > $df".part"
grep -v -E "th\sUSD\)" $df".txt" > $nf".part"
echo "replacement␣special␣character"
sed -f sedscr $df".part" > $df".rda"
sed -i -e s/^$'\t'//g $df".rda"
sed -e s/#//g $nf".part" > $nf".rda"
```

なお，これらの処理は，地道（2018-a）で与えたものと同様であるため詳細は割愛する．

4．手順（S2）の処理の実行

手順 (S2) における処理を，シェル・スクリプト・ファイル（seqdatadump.sh）に記述し，再現性を確保するためにシェルを起動することによって実行した（実行イメージについては図 5 も参照）：

シェルスクリプトによる CSV ファイルの生成の自動実行

```
$ /bin/bash ./seqdatadump.sh
```

ソースコード 5　シェル・スクリプト・ファイル：**seqdatadump.sh**

```
#!/bin/bash
for i in $(seq -w 25) ; do
    echo $i
    Rscript datadump.R "data"$i".rda" "name"$i".rda" "firmfinBC"$i".csv";
done
```

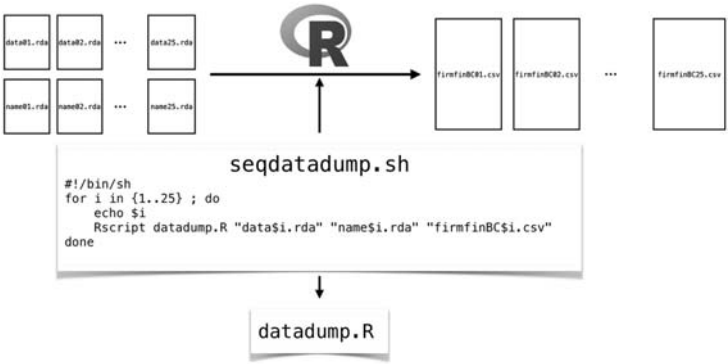

図5　シェルスクリプトによる CSV ファイルの生成の自動実行

なお，ソースコード5の4行目で利用されている R スクリプトファイル datadump.R は実際にファイルの処理を行うためのもので，ソースコード6で与えられる．

ソースコード6　R スクリプトファイル：**datadump.R**

```
require(dplyr)
require(readr)
args <- commandArgs(trailingOnly = T)
tmp1 <- read_tsv(args[1], na = "NA", quote = "", col_names = FALSE,
col_types = cols(
.default = col_double(),
X1 = col_character(),
X2 = col_character(),
X3 = col_character(),
X4 = col_character(),
X5 = col_character(),
X6 = col_character(),
X7 = col_character(),
X8 = col_character(),
X9 = col_character(),
X10 = col_character(),
X11 = col_character(),
X12 = col_character(),
X13 = col_character(),
X14 = col_character(),
X15 = col_character(),
X16 = col_character(),
X17 = col_date(format = "%d/%m/%Y"),
X18 = col_integer(),
X19 = col_character(),
X20 = col_character(),
X21 = col_character(),
X22 = col_character(),
```

```
X23 = col_character(),
X77 = col_integer(),
X78 = col_character(),
X79 = col_integer(),
X80 = col_character(),
X81 = col_character(),
X82 = col_character()
))
tmp2 <- read_tsv(args[2], na = "NA", quote = "", col_names = FALSE)
tmp2 <- data.frame(tmp2)
tmp3 <- tmp2[,1]
firmfin.raw.frame <- data.frame(rep(tmp3, each = 10), tmp1)
varnames <- scan("variablenameOrbis2018.txt", what = "")
colnames(firmfin.raw.frame) <- varnames
firmfin.frame <- mutate(firmfin.raw.frame,
            SIC_code1 = rep(SIC_code1[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            SIC_text1 = rep(SIC_text1[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            SIC_code2 = rep(SIC_code2[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            SIC_text2 = rep(SIC_text2[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            tel = rep(tel[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            NID_num = rep(NID_num[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            NID_lab = rep(NID_lab[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            NID_type = rep(NID_type[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            IP_num = rep(IP_num[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            IP_lab = rep(IP_lab[seq(1, dim(firmfin.raw.frame)[1], 10)], each = 10),
            firmID = paste(firm, ID),
            year = rep(seq(2008, 2017), length(tmp3)))
write_csv(firmfin.frame, args[3])
```

なお，これらの処理についても，地道（2018-a）で与えたものと同様であるため詳細は割愛する．

5．CSV ファイルの結合

これまでの処理によって，個々のデータセットファイルは R に読み込んで分析できるようになったが，一括してデータを解析するために，全てのファイルを結合したものを作成する．

このための処理を，シェル・スクリプト・ファイル（mergedata.sh）に記述し，再現性を確保するためにシェルを起動することによって実行した（実行イメージについては図 6 も参照）：

シェルスクリプトによる CSV ファイルの結合の自動実行

```
$ /bin/bash ./mergedata.sh
```

ソースコード7　シェル・スクリプト・ファイル：**mergedata.sh**

```
1 #!/bin/bash
2 ofn=firmfinBC
3 cat ${ofn}01.csv > $ofn.csv
4 for i in $(seq -w 02 25) ; do
5     echo $i
6     tail -n +2 $ofn$i.csv >> $ofn.csv
7 done
```

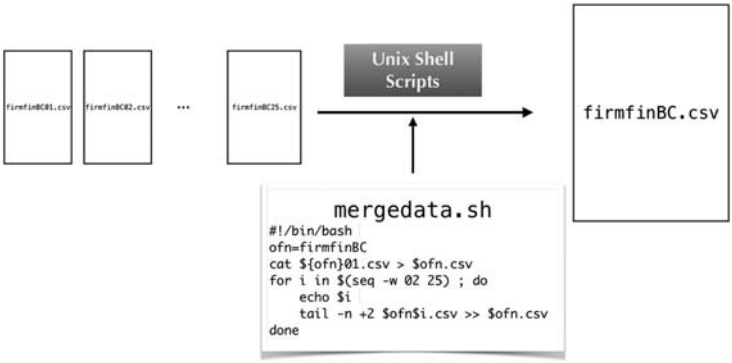

図6　シェルスクリプトによる CSV ファイルの結合の自動実行

なお，これらの処理は，CSV ファイル firmfinBC01.csv に，連続した番号をもつファイルの一行目を除いた（tail -n +2）ものをリダイレクション（>>）機能を使って順に結合することによって実行している．

6．make による自動実行

これまでの一連の前処理の全工程を Makefile にソースコード8のように記述し，UNIX の make コマンドを以下のように実行することによって自動的に処理を行い，再現性を確保した．

make による前処理の自動実行：ターゲット all

```
$ make all
```

ソースコード8：**Makefile** の **all** ターゲット

```
1 all:
2       /bin/bash renumbering.sh
3       /bin/bash seqscript.sh
4       /bin/bash seqdatadump.sh
5       /bin/bash mergedata.sh
```

CSV ファイル[13] `firmfinBC.csv` の生成に関する全工程の処理の流れについては，図7を参照されたい．

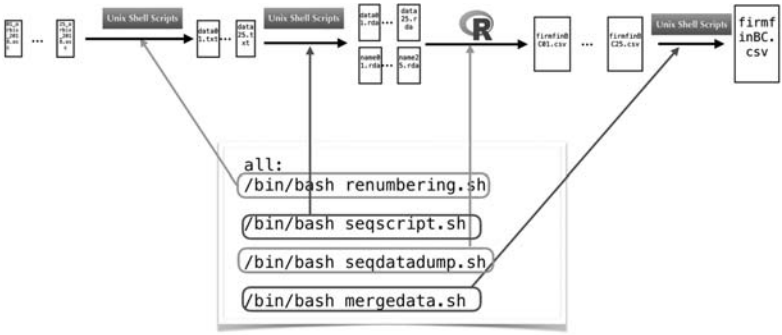

図7　**make** コマンドによる前処理の自動実行（連結決算企業の場合）

特に，ソースコード8における3行目の処理（`/bin/bash seqscript.sh`）が上記の手順（S1）に対応し，4行目の処理（`/bin/bash seqdatadump.sh`）が手順（S2）に対応する．

今回，筆者が利用できる前処理に適したと考えられる最も性能がよい環境[14] で，約6時間18分を要した．

Ⅳ　GNU parallel による前処理の並列化

近年のコンピュータは複数の CPU コアを搭載するものが通常になっており，これらのコアを同時に利用し，ジョブを並列化することによって処理時

13）連結財務データと非連結財務データの両方の場合で CSV ファイルを生成する前処理はほぼ同様の手順で行うことが可能である．

14）Dell Precision T7910（CPU：Intel®Xeon® プロセッサー E5-2687W v4（48コア），Memory：128GB, Storage：SSD 4TB ＋ HDD 4TBx2, OS：Ubuntu 18.04）

間を短縮することが試みられている．このことを実現するツールの一つが，GNU parallel[15] である（Tange（2018）参照）．GNU parallel はコンピュータ上でジョブを並列実行するためのシェルツールであり，macOS や Ubuntu（Linux）等の OS 上で利用可能である（GNU parallel の応用については，Janssens（2014）も参照のこと）．本研究では，GNU parallel を利用して前処理の工程を並列化し，処理速度の向上をはかることを試みる．

1．手順（S1）の並列処理の実行

III 節で与えた手順（S1）を実行するシェル・スクリプト・ファイル seqscript.sh（ソースコード 3）を GNU parallel で並列化する仕様（ソースコード 9 を参照）に変更する．

ソースコード 9　シェル・スクリプト・ファイル：**seqscript-p.sh**

```
#!/bin/bash
for i in $(seq -w 25); do
echo $i
/bin/bash ./script-p.sh "data"$i".txt";
done
```

なお，ソースコード 9 の 4 行目で利用されているシェル・スクリプト・ファイル script-p.sh は，実際にファイルの処理を並列化して行うためのもので，ソースコード10で与えられる．

ソースコード10　シェル・スクリプト・ファイル：**script-p.sh**

```
#!/bin/bash
echo "Remove␣BOM␣codes"
gsed -i -s -e '1s/^\xef\xbb\xbf//' $1
#
echo "dos2unix"
parallel --pipepart -k --block 100M -a $1 "dos2unix␣-f" > tmp
mv tmp $1
df=${1%.*}
num=`echo $df | cut -c 5-`
nf=name$num
echo "separate␣data␣file"
```

15）https://www.gnu.org/software/parallel/

```
12 parallel --pipepart -k --block 100M -a $df".txt" 'grep -E "th\sUSD\)"' > $df".part"
13 parallel --pipepart -k --block 100M -a $df".txt" 'grep -v -E "th\sUSD\)"' > $nf".part"
14 echo "replacement␣special␣character"
15 parallel --pipepart -k --block 100M -a $df".part" "sed␣-f␣sedscr" > tmp
16 parallel --pipepart -k --block 100M -a tmp "sed␣s/^$'\t'//g" > $df".rda"
17 parallel --pipepart -k --block 100M -a $nf".part" "sed␣-e␣s/#//g" > $nf".rda"
18 rm tmp
```

ソースコード10では，dos2nix コマンドによる行末コードの変換，grep コマンドによるファイルの分離，sed コマンドによる文字列の置換などの処理を GNU parallel を利用して並列化している．ここで，-k オプションは，処理の結果を入力順に出力するものであり，行に関する可換性がない処理の並列化における結果の整合性を保つために重要な指定である．なお，オプションなどの詳細については，Tange（2018）を参照されたい．

シェルスクリプトによるデータファイル操作の自動実行

```
$ /bin/bash ./seqscript-p.sh
```

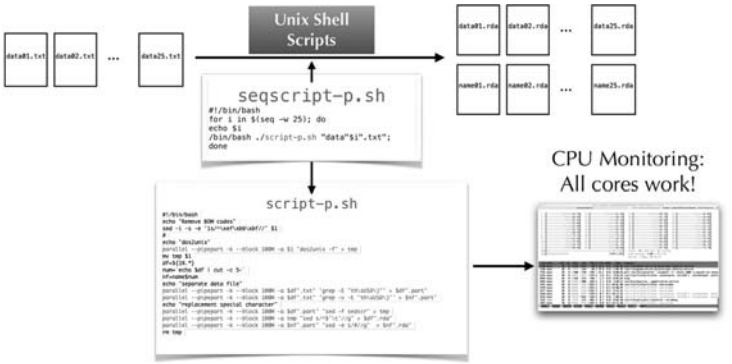

図 8　並列化されたデータファイル処理のシェルスクリプトによる自動実行

2．手順（S2）の並列処理の実行

III 節で与えた手順（S2）における処理を実行するシェル・スクリプト・ファイル seqdatadump.sh を GNU parallel で並列化する仕様（ソースコード11を参照）に変更する．

ソースコード11　シェル・スクリプト・ファイル：**seqdatadump-p.sh**

```
1 #!/bin/bash
2 echo "Start␣datadump␣parallel"
3 seq -w 10 | parallel --jobs 100% Rscript datadump.R "data"{}".rda" "name"{}".rda" "
      firmfinBC"{}".csv"
4 seq -w 11 20 | parallel --jobs 100% Rscript datadump.R "data"{}".rda" "name"{}".rda" "
      firmfinBC"{}".csv"
5 seq -w 21 25 | parallel --jobs 100% Rscript datadump.R "data"{}".rda" "name"{}".rda" "
      firmfinBC"{}".csv"
6 echo "End␣datadump␣parallel"
```

並列化処理を行うシェルスクリプトによる CSV ファイルの生成の自動実行

```
$ /bin/bash ./seqdatadump-p.sh
```

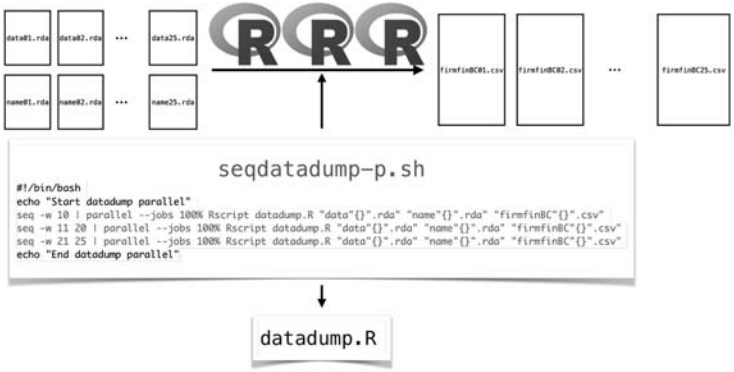

図 9　並列化処理を行うシェルスクリプトによる CSV ファイルの生成の自動実行

なお，ソースコード11の 3，4，5 行目で利用されている R スクリプトファイル datadump.R（ソースコード 6）は，シェル・スクリプト・ファイル seqdatadump.sh（ソースコード 5）で利用された実際にファイルの処理を行うためのものと同一である．

3．make による自動実行

seqscript-p.sh を利用するように Makefile おけるターゲット（all-p）を置き換えた（図10参照）．

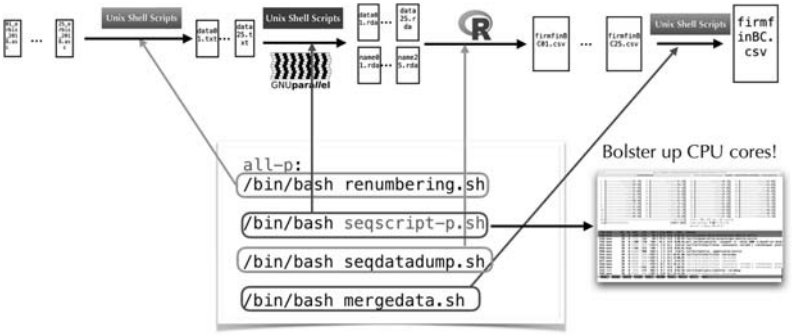

図10　GNU parallel による前処理並列化（1）

この改良によって，前節で前処理を行った同じ環境で，約１時間30分（６時間18分から４時間50分へ）短縮した．このことは，並列化を行わない場合に比べて77％に短縮されたことを表している．

次に，III 節で与えた処理（S2）を実行するシェル・スクリプト・ファイル `seqdatadump.sh` を GNU parallel で並列化する仕様に変更したもの `seqdatadump-p.sh` を利用するように `Makefile` おけるターゲット（`all-p2`）を置き換えた（図11参照）．

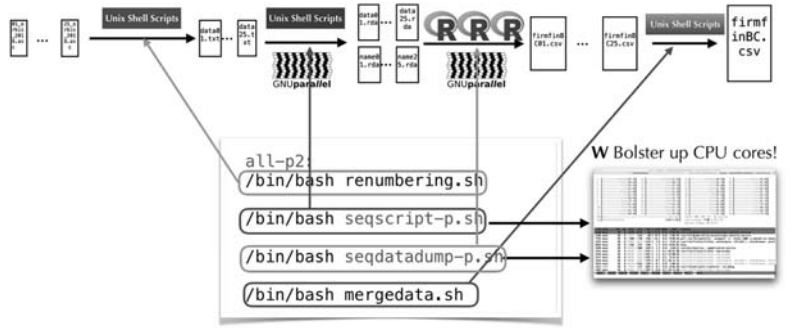

図11　GNU parallel による前処理並列化（2）

この改良によって，前節で前処理を行った同じ環境で，約５時間（６時間

18分から１時間20分へ）短縮した．このことは，並列化を行わない場合に比べて21％（約５分の１）に短縮されたことを表している．

V　おわりに

本稿では，BvD のデータベース Orbis から連結財務諸表優先で抽出したデータセット DS-Orbis-C-2018 のファイルを前処理する工程を GNU parallel を利用することによって並列化し，処理時間を大幅に短縮できることを報告した．この結果は，データを処理・分析・解析する全工程にかかる時間（労力）の50％から90％を占めるといわれる部分を大幅（約５分の１）に短縮できたことを意味する．なお，非連結財務諸表優先で抽出したデータセット DS-Orbis-C-2018 のファイルに関する処理も同様に行うことができ，処理時間もほぼ同じであることを申し添えておく．

ただし，現段階では，データラングリングとして，前処理によって生成された CSV ファイル（`firmfinBC.csv`，`firmfinBU.csv`）を RStudio 上で Apache Spark[16] と SparkR パッケージを利用することによって，データ分析・解析できるオブジェクトへ変換しており，この工程にかかる時間（と負荷）は無視できないものといえる．

この問題に対して，東京大学の専有利用型リアルタイムデータ解析ノード（FENNEL）と GPGPU[17] 環境でデータベース管理システム PostgreSQL[18] と PG-Strom[19] を利用することによって，ラングリングを速度の面から改善することを現在検討している．

（筆者は関西学院大学商学部教授）

16）`https://spark.apache.org/`
17）GPGPU とは，General-Purpose computing on Graphics Processing Units の略語であり，画像処理を高速に実行する GPU（Graphics Processing Unit）の機能を，画像処理以外の用途に転用することである（IT 用語辞典 `http://e-words.jp/w/GPGPU.html` 参照）．
18）`https://www.postgresql.org/`
19）`https://heterodb.github.io/pg-strom/ja/`

参考文献

［1］ Janssens, J.（2014）*Data Science at the Command Line,* O'Reilly Media.（太田満久，下田倫大，増田泰彦監訳，長尾高弘訳（2015）『コマンドラインではじめるデータサイエンス―分析プロセスを自在に進めるテクニック―』，オライリー・ジャパン.）

［2］ 地道正行（2018-a）『探索的財務ビッグデータ解析―前処理，データラングリング，再現可能性―』，商学論究，第66巻，第1号，pp. 1-31，関西学院大学商学研究会.

［3］ 地道正行（2018-b）『探索的財務ビッグデータ解析―データ可視化，統計モデリング，モデル選択，モデル評価，動的文書生成，再現可能研究―』，商学論究，第66巻，第2号，pp. 1-41，関西学院大学商学研究会.

［4］ 本橋智光（2018）『前処理大全―データ分析のための SQL/R/Python 実践テクニック―』，技術評論社.

［5］ 西田圭介（2017）『ビッグデータを支える技術―刻々とデータが脈打つ自動化の世界―』，技術評論社.

［6］ Patil, DJ（2012）*Data Jujitsu: The Art of Turning Data into Product,* An O'Reilly Radar Report, O'Reilly.

［7］ Tange, Ole,（2018）*GNU Parallel 2018,* ISBN: 9781387509881, DOI: 10.5281/zenodo.1146014, URL: https://doi.org/10.5281/zenodo.1146014, Mar, 2018.

［8］ Wickham, H. and G. Grolemund（2016）*R for Data Science,* O'Reilly.

謝辞

本研究の一部は以下の研究費より助成を得ている．ここに感謝の意を表する．

科研費 科学研究費基盤研究 C：「グラフィカル・データ・アナリシスによる格差研究と社会環境会計による解決方法の提案」（2016年～2018年），課題番号：16K04022，研究代表者：阪智香

科研費 科学研究費基盤研究 C：「共有価値創造（CSV）のための社会環境会計の構築」（2019年～2021年），課題番号：19K02006，研究代表者：阪智香

平成30年度学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題：「財務ビッグデータの可視化と統計モデリング」，課題番号：jh181001-NWJ，研究代表者：地道正行

平成31年度学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題：「財務ビッグデータの可視化と統計モデリング」，課題番号：jh191002-NWJ，研究代表者：地道正行

関西学院大学図書館図書費 B，研究設備費（III），個人研究費

また，BvD の増田歩氏にはデータの抽出に関して多大なるご協力いただいた．ここに感謝の意を表する．