

Parallel Processing – HW 1

Embarrassingly Parallel Computations with MPI

Goal: Estimate π by Monte-Carlo computations

Dvir Zaguri 315602284
Yehonatan Arama 207938903

1. הקדמה

נדרשו לבצע חישוב של המספר π בשיטת 'Monte Carlo' על ידי הטלת נקודות על מישור X,Y. המטרה היא לבחון את יכולות החישוב המקבילי שמבצע מעבד בעל מספר ליבות עבור חישובים ארוכים עם המון איטרציות באמצעות MPI. בעבודה הני"ל השתמשנו בשני כלים profiling כנדרש בעבודה 'Scalasca' ו-'Jumpshot'. Scalasca היא כלי ראשי לניתוח ביצועים של אפליקציות מקביליות בעוד Jumpshot נותן ויזואליזציה של ביצוע המכונה ב-execution של תוכנית מסוימת ומאפשר למשתמש לנתח בעצמו.

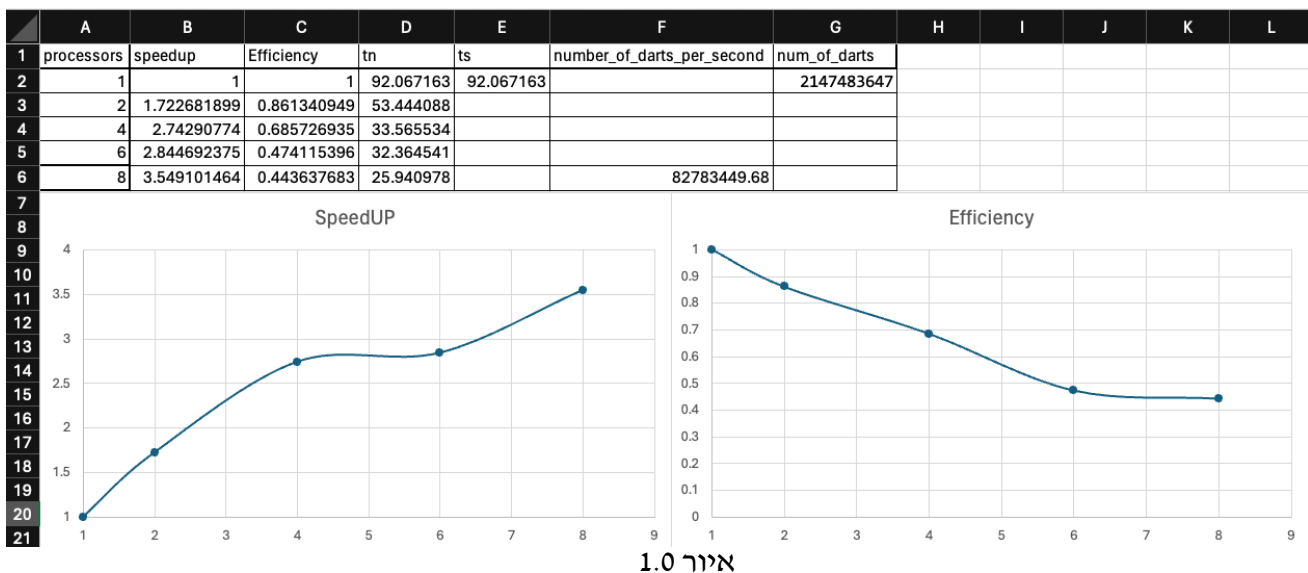
2. מהלך הניסוי

השיטה

בחירת שתי משתנים אקראיים המתפלגים אחיד בקטע $[-1, 1]$ כאשר כל אחד מהם מייצג את אחת הקואורדינטות במישור. היחס בין כמות הנקודות שמיקומם הוא בשטח המעגל שרדיוסו 1 החסום בריבוע $[-1, 1] \times [-1, 1]$ לבין כמות הנקודות שהוטלו הוא $\frac{\pi}{4}$, כיחס השטחים של המעגל והריבוע. לכן אחרי הטלת הנקודות כפלו את יחס הפגיעות/נסיונות ב-4.

פירוט הפתרון

מימשנו את התוכנית כנדרש וביצענו 2,147,483,647 הטלות שונות על מנת לקבל את ערך המספר π בדיוק של 4 ספרות אחרי הנקודה העשרונית. יצאנו שני גרפים המייצגים את שיפור הביצועים של התוכנית, גם במהירות וגם ביעילות שלה ביחס למספר הליבות שהוקצו לתוכנית ה-MPI להשתמש לצורך ביצוע הקוד שכתבנו. את התוכנית קימפלנו והרצנו עם הרצת MPI עם 1,2,4,6 ו-8 ליבות, מדדנו זמנים ואת כלל הפרמטרים שנדרשו, להלן תוצאות ההרצות:



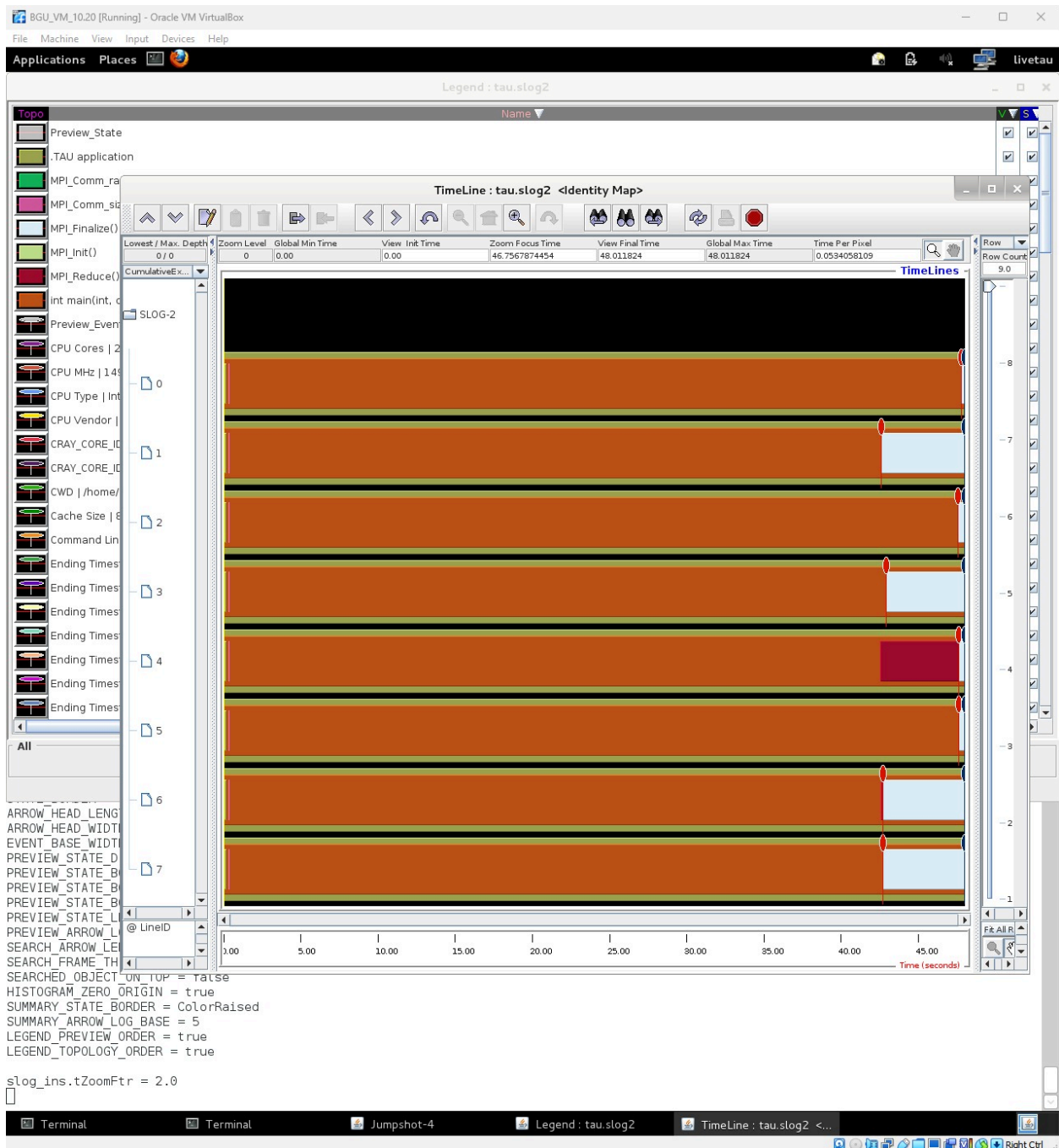
איור 1.0

באיור 1.0 ניתן לראות כי בהרצה המהירה ביותר של התוכנית (בהרצה באמצעות 8 ליבות) התבצעו במוצע 82,783,449.68 הטלות/שניה. בהסתכלות על הטבלה המצורפת במסמך המטלה ניתן לראות שבהרצה זו המחשב שלנו מתמקם בטבלת (2) Monte Carlo Darts Game בעמודה הימנית בין 'Matlab 2010 LANS 2.6 G i7 2 Core' להרצה של אותו ה-CPU ללא ה-MATLAB (בין שורה ראשונה לשנייה)

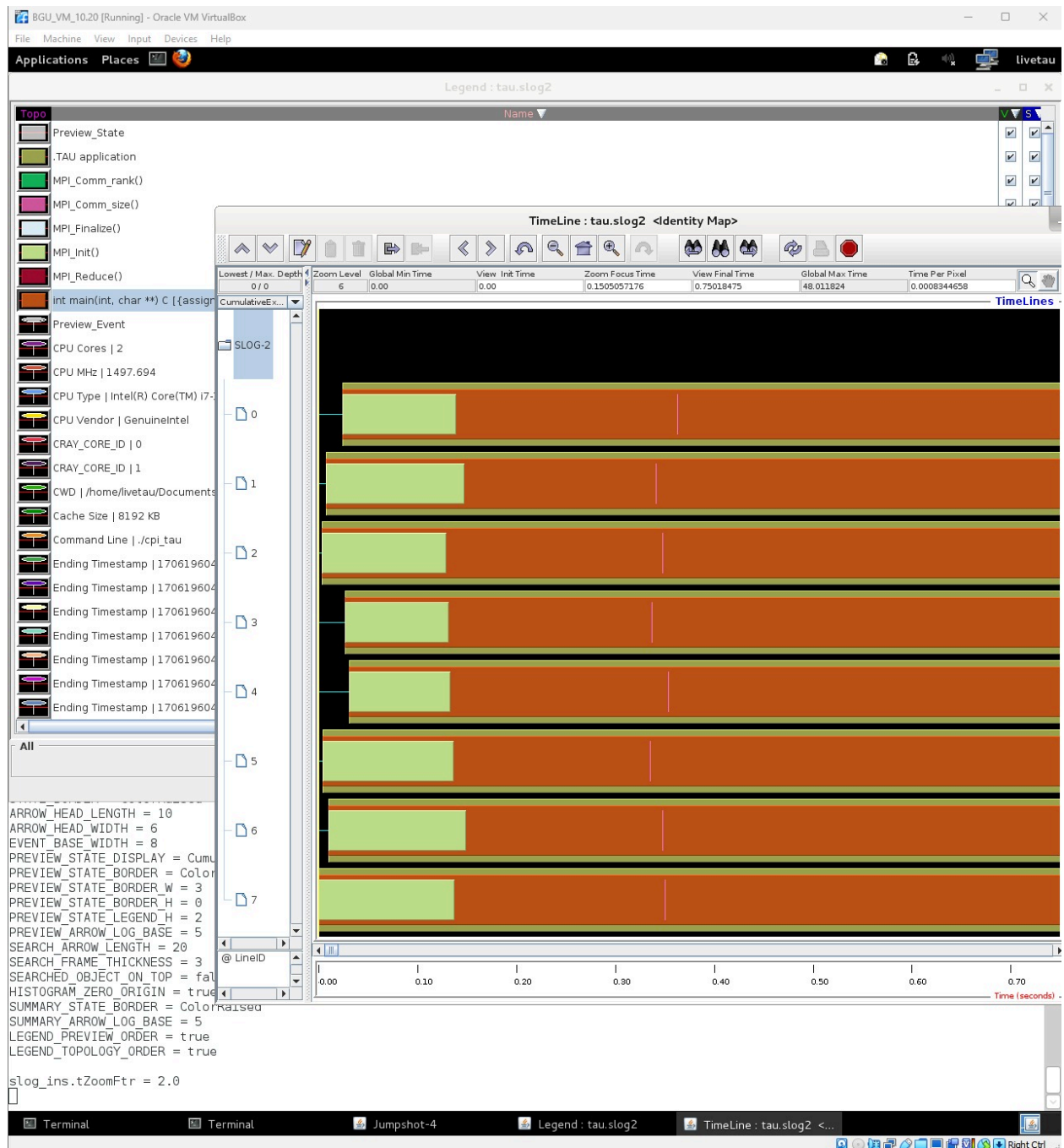
ויזואליזציה, trace ו-profiling באמצעות *JumpShot* :

באיורים המצורפים ניתן לראות את הניתוח של חלק מריצת התוכנית כאשר מצורף בחלון האחורי יותר מקרא המתאר איזה מהחלקים בריצה צבוע באיזה צבע, וכמה זמן לקח באופן יחסי כל חלק מהקוד בכל ליבה.

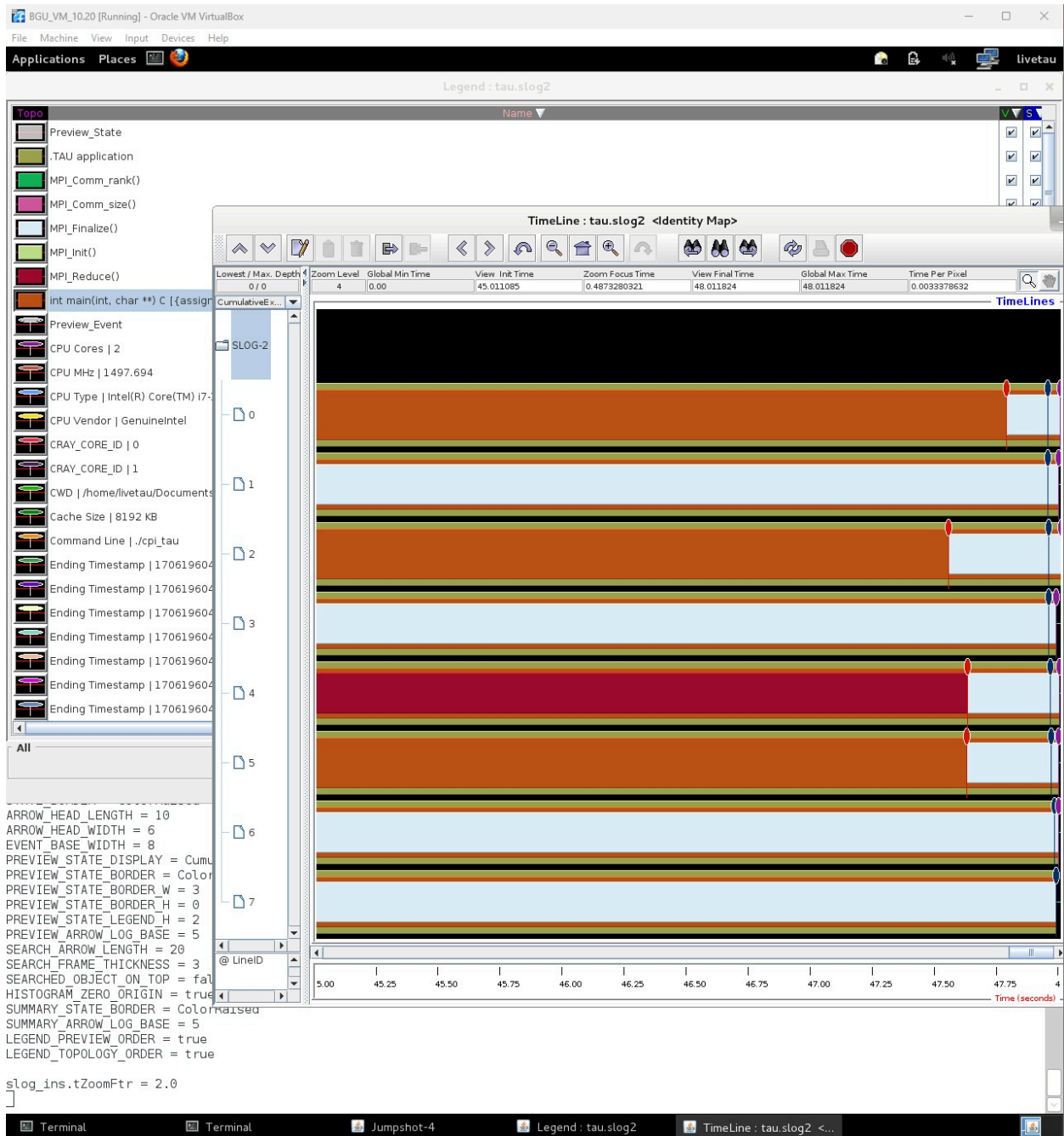
ניתן לשים לב ששלב ה-*Initialize* וה-*Finalize* בתוכנית לקחו יותר זמן משאר הפקודות בעוד שהשאר כולן לקחו בערך זמן שווה.



איור 1.1



איור 1.2



איור 1.3

Profiling:

BGU_VM_10.20 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Applications Places

Terminal

File Edit View Search Terminal Help

```

TreeDir is FBInfo(38 @ 28153)
Annotations is FBInfo(0 @ 0)
Postamble is FBInfo(0 @ 0)

1521 enters: 0 exits: 0

Number of Drawables = 648
timeElapsed between 1 & 2 = 96 msec
timeElapsed between 2 & 3 = 339 msec
bash-4.0$ jumpshot ./
assignment1.c cpi_tau
tau.edf tau.trace.1.0.0.trc
.assignment1.c.swp events.0.edf
tau.slog2 tau.trace.2.0.0.trc
assignment1.o events.1.edf
tau.trace.0.0.0.trc tau.trace.3.0.0.trc
bash-4.0$ jumpshot ./tau.slog2
GUI_LIBDIR is set. GUI_LIBDIR = /usr/local
Java is version 1.7.0_01.
Starting the SLOG-2 Display Program ....
Jumpshot-4 setup file : /home/livetau/.ju
Initialize Parameters:
Y_AXIS_ROOT_LABEL = SLOG-2
INIT_SLOG2_LEVEL_READ = 4
AUTO_WINDOWS_LOCATION = true
SCREEN_HEIGHT_RATIO = 0.5
TIME_SCROLL_UNIT_RATIO = 0.01
Y_AXIS_ROOT_VISIBLE = true
ACTIVE_REFRESH = false
ROW_RESIZE_MODE = Row
BACKGROUND_COLOR = black
STATE_HEIGHT_FACTOR = 0.9
NESTING_HEIGHT_FACTOR = 0.8
ARROW_ANTIALIASING = default
MIN_WIDTH_TO_DRAG = 4
CLICK_RADIUS_TO_LINE = 3
LEFTCLICK_INSTANT_ZOOM = true
STATE_BORDER = ColorRaised
ARROW_HEAD_LENGTH = 10
ARROW_HEAD_WIDTH = 6
EVENT_BASE_WIDTH = 8
PREVIEW_STATE_DISPLAY = CumulativeExclusionRe
PREVIEW_STATE_BORDER = ColorXOR
PREVIEW_STATE_BORDER_W = 3
PREVIEW_STATE_BORDER_H = 0
PREVIEW_STATE_LEGEND_H = 2
PREVIEW_ARROW_LOG_BASE = 5
SEARCH_ARROW_LENGTH = 20
SEARCH_FRAME_THICKNESS = 3
SEARCHED_OBJECT_ON_TOP = false
HISTOGRAM_ZERO_ORIGIN = true
SUMMARY_STATE_BORDER = ColorRaised
SUMMARY_ARROW_LOG_BASE = 5
LEGEND_PREVIEW_ORDER = true
LEGEND_TOPOLOGY_ORDER = true

slog_ins.tZoomFtr = 2.0

^Cbash-4.0$ ^C
bash-4.0$ ^C
bash-4.0$ paraprof

```

TAU: ParaProf Manager

File Options Help

Applications

Standard Applications

Default App

Default Exp

assignment1/Documents/livetau/

TIME

Default (jdbc:h2:/home/livetau/.ParaPro

perfexplorer_working (jdbc:h2:/home/li

TrialField	Value
Name	assignment1/Docu...
Application ID	0
Experiment ID	0
Trial ID	0
CPU Cores	2
CPU MHz	1497.694
CPU Type	Intel(R) Core(TM) ...
CPU Vendor	GenuineIntel
CRAY_CORE_ID	0
CWD	/home/livetau/Doc...
Cache Size	8192 KB
Command Line	./cpi_tau
Ending Timestamp	1706196040034251
Executable	/home/livetau/Doc...
File Type Index	1
File Type Name	TAU profiles
Hostname	localhost.localdom...
Local Time	2024-01-25T07:19...
MPI Processor Name	localhost.localdom...

TAU: ParaProf: /home/livetau/Documents/assignment1

File Options Windows Help

Metric: TIME

Value: Exclusive

Std. Dev.

Mean

Max

Min

node 0

node 1

node 2

node 3

node 4

node 5

node 6

node 7

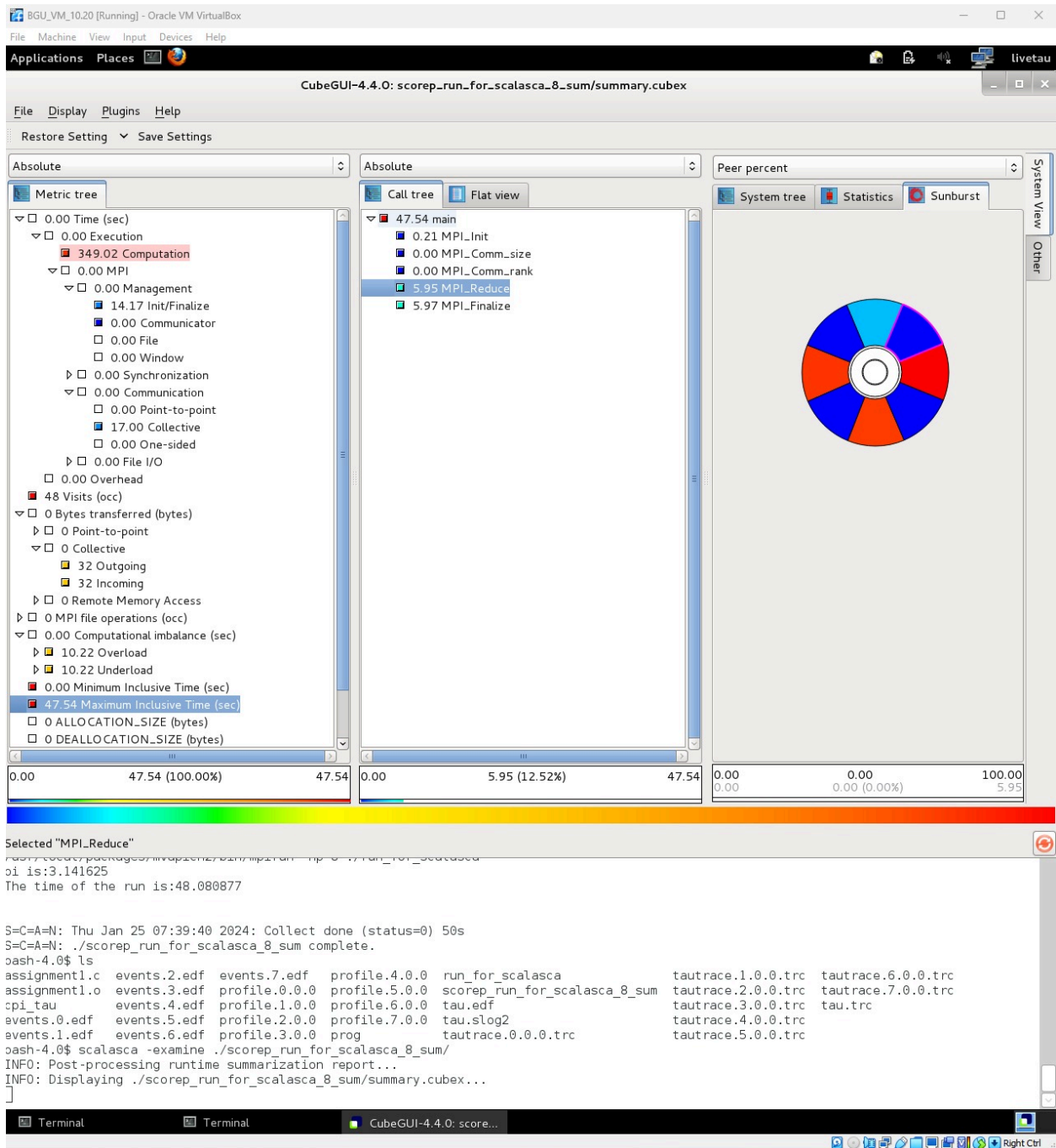
Terminal

Terminal

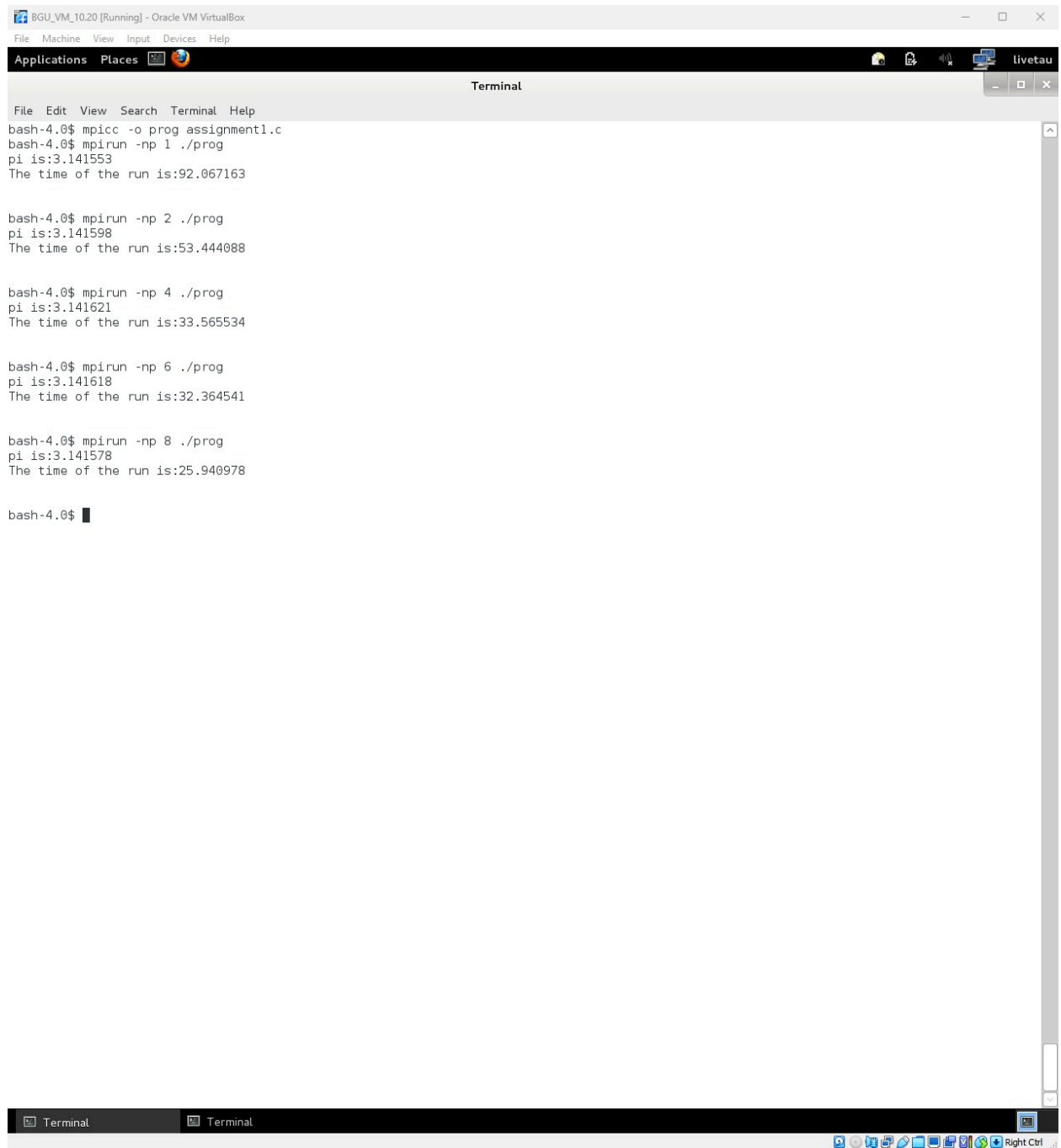
TAU: ParaProf Manager

TAU: ParaProf: /home/...

Right Ctrl



באיורים הבאים מצורף תיעוד של ריצת הקוד ותוצאות הזמנים וחשוב הערך של π מהטרמינל:



```
File Edit View Search Terminal Help
bash-4.0$ mpicc -o prog assignment1.c
bash-4.0$ mpirun -np 1 ./prog
pi is:3.141553
The time of the run is:92.067163

bash-4.0$ mpirun -np 2 ./prog
pi is:3.141598
The time of the run is:53.444088

bash-4.0$ mpirun -np 4 ./prog
pi is:3.141621
The time of the run is:33.565534

bash-4.0$ mpirun -np 6 ./prog
pi is:3.141618
The time of the run is:32.364541

bash-4.0$ mpirun -np 8 ./prog
pi is:3.141578
The time of the run is:25.940978

bash-4.0$
```

איור 1.6

3. סיכום ומסקנות

ניתן להבחין בתוצאות הניסוי הן בכלי ה-profilng והן בטבלאת ה-excel ולהבחין שאכן מקבילות התוכנית אינה אפשרית בכל חלקי התוכנית ולכן אורך הביצוע של התוכנית אינו מתנהג באופן מדויק מספיק בהתאמה למספר הליבות המוקצות בכל הרצה וכך גם זמני הרצת התוכנית. ניתן להבחין שבהרצת תוכנית mpi צריך להתחשב ב-tradeoff בזמנים וכוח העיבוד הנדרש עבור איתחול (*Initialize*) וסיום תוכנית mpi (*Finalize*) כאשר מנסים לייעל זמני ריצה של תוכנית. בתוצאות כלי המדידה ראינו שפקודות אלו לקחו די הרבה זמן ביחס לזמן ריצת התוכנית.