Ατομική Διπλωματική Εργασία

ΑΝΑΛΥΣΗ ΚΑΙ ΓΡΑΦΙΚΗ ΑΠΕΙΚΟΝΙΣΗ ΠΑΡΑΛΛΗΛΩΝ (DDM) ΕΦΑΡΜΟΓΩΝ

Ζαχαρίας Ζαχαρίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αύγουστος 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση και Γραφική
Απεικόνιση Παράλληλων (DDM)
Εφαρμογών

Ζαχαρίας Ζαχαρίου

Επιβλέπων Καθηγητής
Παρασκευάς Ευριπίδου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Αύγουστος 2012

Περίληψη

Η ανάλυση της αποτελεσματικότητας και της συμπεριφοράς μιας παράλληλης εφαρμογής είναι εξίσου σημαντική με την ανάλυση της αποτελεσματικότητας και της συμπεριφοράς μιας σειριακής εφαρμογής. Ωστόσο, σήμερα με την ανάπτυξη των υπολογιστικών συστημάτων βλέπουμε ολοένα και περισσότερο να κυκλοφορούν συστήματα με πολλαπλούς επεξεργαστές και πυρήνες. Αυτό κάνει απαραίτητη την ανάγκη ανάπτυξης παράλληλων εφαρμογών που θα εκμεταλλεύονται πλήρως τις δυνατότητες των νέων υπολογιστικών συστημάτων.

Μπροστά στη μεγάλη ανάγκη για ανάπτυξη παράλληλων εφαρμογών, έπρεπε να δημιουργηθούν και τα κατάλληλα εργαλεία που θα μπορούσαν να μας προσφέρουν μια λεπτομερή και συγχρόνως αξιόπιστη ανάλυση αυτών των εφαρμογών. Το PARAVER είναι ένα από αυτά τα εργαλεία. Με χρήση του εργαλείου αυτού μπορούμε να έχουμε μια ποιοτική και γενική αντίληψη της συμπεριφοράς μιας εφαρμογής, καθώς και λεπτομερή και ποσοτική ανάλυση των προβλημάτων της. Επίσης, η δυνατότητα που προσφέρει για γραφική αναπαράσταση της εκτέλεσης μιας εφαρμογής, κάνει ακόμη πιο εύκολη την προσπάθεια του χρήστη για να κατανοήσει την συμπεριφορά της εφαρμογής του.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και κίνητρο της Διπλωματικής Εργασίας

Όπως όλοι γνωρίζουμε, το μέλλον των υπολογιστικών συστημάτων πλέον στηρίζεται σε συστήματα με πολλαπλούς επεξεργαστές και πυρήνες. Αποτέλεσμα του γεγονότος αυτού είναι η ανάγκη ανάπτυξης παράλληλων προγραμματιστικών μοντέλων, τα οποία θα μπορούν να εκμεταλλευτούν στο μέγιστο τις τεράστιες υπολογιστικές δυνατότητες που μας προσφέρουν τα συστήματα αυτά. MPI, OpenMP και Pthreads αποτελούν μερικά από αυτά τα προγραμματιστικά μοντέλα. Εκτός αυτών υπάρχει και το Data-Driven Multithreading (DDM) μοντέλο, το οποίο αναπτύσσεται από το DDM Group του Τμήματος Πληροφορικής, του Πανεπιστημίου Κύπρου. Έτσι, έχοντας πλέον αυτά τα προγραμματιστικά μοντέλα στη διάθεση μας, μπορούμε να αναπτύξουμε παράλληλες εφαρμογές τις οποίες στη συνέχεια θα εκτελούμε σε υπολογιστικά συστήματα με πολλαπλούς επεξεργαστές και πυρήνες.

Μέχρις εδώ, όλα καλά. Τι κάνουμε όμως όταν επιθυμούμε να ελέγξουμε την αποτελεσματικότητα και την συμπεριφορά μιας παράλληλης εφαρμογής που έχουμε αναπτύξει; Ακριβώς αυτός είναι ο σκοπός της Ατομικής Διπλωματικής Εργασίας μου, ο οποίος θα εκπληρωθεί μέσα από την μελέτη του εργαλείου PARAVER το οποίο αναπτύχθηκε από το Barcelona Supercomputing Center (BSC).

Πιο συγκεκριμένα, το PARAVER μας βοηθά να έχουμε μια ποιοτική και γενική αντίληψη της συμπεριφοράς μιας εφαρμογής, καθώς και λεπτομερή και ποσοτική ανάλυση των προβλημάτων της. Περισσότερες λεπτομέρειες για τις δυνατότητες του PARAVER μπορείτε να βρείτε στο **Κεφάλαιο 6 – PARAVER**.

1.2 Μεθοδολογία υλοποίησης μελέτης

Αρχικά, προσπάθησα να μάθω όσες περισσότερες λεπτομέρειες μπορούσα για το PARAVER, μέσα από την ιστοσελίδα του εργαλείου (https://www.bsc.es/computer-sciences/performance-tools/paraver) και αρκετά έγγραφα PDF που βρήκα σε αυτή. Μέσα από την μελέτη μου για το PARAVER, διαπίστωσα πως προτού ξεκινήσω να χρησιμοποιώ το συγκεκριμένο εργαλείο έπρεπε να μελετήσω ένα άλλο εργαλείο, το EXTRAE, που και αυτό αναπτύχθηκε από το Barcelona Supercomputing Center (BSC).

Το EXTRAE χρησιμοποιείται για την παραγωγή των trace αρχείων μιας εφαρμογής, τα οποία αρχεία στη συνέχεια τα ανοίγουμε με το PARAVER για να δούμε την συμπεριφορά της εφαρμογής. Ωστόσο, κατά την μελέτη του εργαλείου EXTRAE διαπίστωσα και πάλι πως για πλήρη εκμετάλλευση των δυνατοτήτων που προσφέρει, έπρεπε να μελετήσω το Performance Application Programming Interface (PAPI).

Το PAPI (http://icl.cs.utk.edu/papi/) αποτελεί ένα εύκολο τρόπο εξαγωγής χρήσιμων συμπερασμάτων για την εκτέλεση μιας εφαρμογής, μέσα από τους διάφορους μετρητές απόδοσης που μας προσφέρει.

Ακολούθως, λόγω κάποιων ιδιαίτερων απαιτήσεων του DDM μοντέλου τις οποίες τα εργαλεία EXTRAE και PARAVER δεν υποστήριζαν, χρειάστηκε να μελετήσω την δομή των τριών ειδών αρχείων (*.prv, *.pcf και *.row) που παράγει το EXTRAE για μια εφαρμογή, και τα οποία στη συνέχεια χρησιμοποιεί το PARAVER.

Αφού μελέτησα την δομή των τριών αυτών αρχείων, έγραψα έναν Parser σε γλώσσα C ο οποίος μετατρέπει κάποιες πληροφορίες των *.prv και *.pcf αρχείων, με τρόπο που να εξυπηρετεί τις επιπλέον απαιτήσεις του DDM μοντέλου.

Αφού έκανα όλα αυτά, συνέχισα με την μελέτη του εργαλείου PARAVER.

1.3 Δομή αυτής της διπλωματικής

Σε αυτή την διπλωματική περιέχονται 8 κεφάλαια. Περιληπτικά στο **Κεφάλαιο 2** θα γίνει μια περιγραφή του εργαλείου EXTRAE και των δυνατοτήτων που μας προσφέρει. Έπειτα, στο **Κεφάλαιο 3** θα γίνει μια περιγραφή του PAPI και των δυνατοτήτων που μας προσφέρει. Ακολούθως, στο **Κεφάλαιο 4** θα γίνει μια περιγραφή της δομής των τριών ειδών αρχείων (*.prv, *.pcf και *.row) που παράγει το EXTRAE και στο **Κεφάλαιο 5** θα επεξηγηθεί ο αλγόριθμος του Parser προγράμματος που έγραψα. Στο **Κεφάλαιο 6** θα αναλύσω το εργαλείο PARAVER και τις δυνατότητες που μας παρέχει. Επίσης, στο **Κεφάλαιο 7** θα περιγράψω τον τρόπο με τον οποίο κατάφερα να αναλύσω DDM εφαρμογές με τα εργαλεία EXTRAE και PARAVER. Στο **Κεφάλαιο 8** παραθέτω τα σημαντικότερα screenshots που πήρα για τις DDM εφαρμογές με το εργαλείο PARAVER καθώς και μια λεπτομερή ανάλυση αυτών. Τέλος, επισυνάπτω δύο παραρτήματα στα οποία μπορείτε να διαβάσετε τα e-mail επικοινωνίας που είχα ανταλλάξει με ερευνητές του Barcelona Supercomputing Center (BSC) για διάφορα προβλήματα που αντιμετώπισα με τα εργαλεία τους, EXTRAE και PARAVER.

Κεφάλαιο 2

Εργαλείο EXTRAE

2.1 Εισαγωγή στο εργαλείο EXTRAE.

2.2 Λήψη και εγκατάσταση του εργαλείου EXTRAE.

2.3 Περιγραφή XML αρχείου.

2.4 EXTRAE API.

2.5 Εκτέλεση εφαρμογής με το εργαλείο EXTRAE.

2.6 Διαδικασία συγχώνευσης *.mpit αρχείων (Merging Process).

2.7 Παραδείγματα.

2.1 Εισαγωγή στο εργαλείο EXTRAE

Το EXTRAE είναι ένα εργαλείο το οποίο ιχνογραφεί προγράμματα τα οποία έχουν γίνει compile και εκτελούνται με το shared memory μοντέλο (OpenMP και Pthreads), το message passing (MPI) προγραμματιστικό μοντέλο ή με συνδυασμό και των δύο μοντέλων.

Προς το παρόν είναι διαθέσιμο πάνω σε διάφορες πλατφόρμες και λειτουργικά συστήματα, όπως: IBM PowerPC (Linux/AIX), x86 και x86-64 Linux, OpenSolaris και FreeBSD.

Ο συνδυασμός του εργαλείου EXTRAE μαζί με το εργαλείο PARAVER προσφέρει τεράστιες δυνατότητες ποσοτικής και ποιοτικής ανάλυσης. Με χρήση των δύο αυτών εργαλείων μπορούμε να εντοπίσουμε τα ακριβή αίτια για τα οποία μια παράλληλη εφαρμογή υστερεί σε απόδοση (bottlenecks) και να βελτιώσουμε την απόδοσή της.

2.2 Λήψη και εγκατάσταση του εργαλείου EXTRAE

Οποιοσδήποτε μπορεί να κατεβάσει τον πηγαίο κώδικα του EXTRAE από την ιστοσελίδα του Barcelona Supercomputing Center (BSC) και πιο συγκεκριμένα από τον ακόλουθο σύνδεσμο:

https://www.bsc.es/computer-sciences/performance-tools/downloads

Αρκεί να εισάγει το e-mail του στο αντίστοιχο πεδίο, να επιλέξει το Extrae Sources όπως φαίνεται στην Εικόνα 2.2.1 και να πατήσει το κουμπί Download.

Module name	Version	Description
Extrae	2.2.1 (February 24th 2012)	Paraver/Dimemas instrumentation package (formerly named MPItrace). Intercepts MPI, OpenMP and Pthreads. Support to CUDA, Cell-SDK, StarSS and Nanos runtime. Open source under LGPL
Platform	**Requirements**	
Linux-x86 (32bits)	○ OpenMPI 1.4.4, PAPI 4.2.0 ○ Open MPI 1.4.4	
Linux-x86 (64bits)	○ OpenMPI 1.4.4, PAPI 4.1.4 ○ OpenMPI 1.4.4 ○ MPICH2 1.4, PAPI 4.1.4 ○ MPICH2 1.4	
Linux-PowerPC	○ 32 bits, MPICH 1.2.7, PAPI 3.6.2 ○ 64 bits, MPICH 1.2.7, PAPI 3.6.2	
Linux-Itanium	○ SGI MPI, PAPI 4.1.4 ○ SGI MPI	
BGP	○ PAPI 4.1.4	
Power-Cell	○ OpenMPI 1.2, SDK 3.0 (32 bits) ○ OpenMPI 1.2, SDK 3.0 (64 bits)	
SOURCES	◉ Dependencies: libxml2 2.5.0 (except Blue Gene); libunwind for linux ia64 and x86-64. Optional: PAPI; DynInst; liberty and libbfd; MPI; OpenMP	

Εικόνα 2.2.1

Έχοντας κατεβάσει και αποσυμπιέσει τον πηγαίο κώδικα του EXTRAE, μετακινούμαστε μέσα στο φάκελο και εκτελούμε την εντολή **./configure** ακολουθούμενη από κάποιες απαραίτητες επιλογές οι οποίες είναι:

- **--prefix=DIR**

  Το μονοπάτι στο οποίο θα εγκατασταθεί το EXTRAE. Μετά και την εκτέλεση της εντολής make install, στο μονοπάτι DIR θα βρίσκονται οι φάκελοι lib/, include/, share/ και bin/ οι οποίοι περιέχουν ότι χρειάζεται για να τρέξει το EXTRAE.

- **--with-mpi=DIR ή --without-mpi**

  Το μονοπάτι στο οποίο είναι εγκατεστημένο το MPI, έτσι ώστε να μπορεί το EXTRAE να το χρησιμοποιήσει. Σε περίπτωση που δεν θα χρησιμοποιηθεί MPI, δίνουμε την επιλογή --without-mpi.

- **--with-unwind=DIR ή --without-unwind**

  Το μονοπάτι στο οποίο μπορεί το EXTRAE να βρει τις βιβλιοθήκες και τα includes του Unwind. Το Unwind χρησιμοποιείται για λήψη callstack πληροφοριών σε διάφορες αρχιτεκτονικές. Σε περίπτωση που δεν θα χρησιμοποιηθεί η βιβλιοθήκη Unwind, δίνουμε την επιλογή --without-unwind.
- **--with-dyninst=DIR ή --without-dyninst**

  Το μονοπάτι στο οποίο είναι εγκατεστημένο το Dynamic Instrumentation, έτσι ώστε να μπορεί το EXTRAE να το χρησιμοποιήσει. Όταν χρησιμοποιείται το DynInst είναι απαραίτητο να δηλωθεί και το DWARF πακέτο, με χρήση της επιλογής --with-dwarf=DIR. Σε περίπτωση που δεν θα χρησιμοποιηθεί DynInst, δίνουμε την επιλογή --without-dyninst.
- **--with-papi=DIR ή --without-papi**

  Το μονοπάτι στο οποίο μπορεί το EXTRAE να βρει τις βιβλιοθήκες και τα includes του PAPI. Το PAPI χρησιμοποιείται για λήψη μετρητών απόδοσης. Σε περίπτωση που δεν θα χρησιμοποιηθεί η βιβλιοθήκη PAPI, δίνουμε την επιλογή --without-papi.

Εκτός από τις απαραίτητες επιλογές υπάρχουν και ορισμένες προαιρετικές. Οι σημαντικότερες και πιο χρήσιμες είναι:

- **--enable-merge-in-trace**

  Ενεργοποιώντας αυτή την επιλογή η διαδικασία συγχώνευσης των *.mpit αρχείων εκτελείται αυτόματα αμέσως μετά την εκτέλεση της εφαρμογής και χωρίς να χρειαστεί να δώσουμε εμείς την εντολή **./mpi2prv**. Απαραίτητη προϋπόθεση είναι το **merge enabled** πεδίο στο XML αρχείο να είναι **“yes”**.
- **--enable-posix-clock**

  Χρήση του POSIX clock (clock_gettime call) αντί των χαμηλού επιπέδου timing ρουτινών. Χρήσιμο εάν το σύστημα στο οποίο θα εγκαταστήσουμε το EXTRAE αλλάζει την συχνότητα των επεξεργαστών του κατά τη διάρκεια της εκτέλεσης μιας εφαρμογής.
- **--enable-sampling**

  Ενεργοποίηση υποστήριξης δειγματοληψίας με χρήση του PAPI.
- **--enable-pthread**

  Ενεργοποίηση υποστήριξης ιχνογράφησης κλήσεων της Pthread βιβλιοθήκης.

- **--enable-xml**

  Ενεργοποίηση υποστήριξης για XML configuration.

Για περισσότερες διαθέσιμες επιλογές κατά την εγκατάσταση του EXTRAE μπορείτε να συμβουλευτείτε το "Extrae User Guide" PDF το οποίο βρίσκεται στον σύνδεσμο: https://www.bsc.es/sites/default/files/public/computer_science/performance_tools/extrae-user-guide.pdf

Αφού εκτελέσουμε την εντολή **./configure**, ακολούθως εκτελούμε πρώτα την εντολή **make** και έπειτα την εντολή **make install**. Ολοκληρώνοντας τη διαδικασία αυτή, το EXTRAE εγκαθίσταται στο μονοπάτι που καθορίσαμε με την επιλογή **--prefix=DIR**.

Αν οποιαδήποτε στιγμή επιθυμούμε να δούμε με τι **./configure** εντολή εγκαταστάθηκε το EXTRAE, απλά εκτελούμε την εντολή:

${EXTRAE_HOME}/etc/configured.sh

Αφού πρώτα ορίσουμε την μεταβλητή EXTRAE_HOME να δείχνει στο μονοπάτι όπου εγκαταστάθηκε το EXTRAE.

export EXTRAE_HOME=DIR

Στην Εικόνα 2.2.2 φαίνεται ένα παράδειγμα εκτέλεσης του πιο πάνω στην μηχανή **teras**, την οποία χρησιμοποιούσα κατά την πρακτική εξάσκηση της Διπλωματικής Εργασίας μου.

```
EXTRAE_HOME points to /home/xeirwn/zacharias/EXTRAE_INSTALL and the directory exists .. OK
Extrae SVN revision: $ Rev : 711 $

Loaded specs for Extrae from /home/xeirwn/zacharias/EXTRAE_INSTALL/etc/extrae-vars.sh
Extrae was configured with:
$ ./configure --prefix=/home/xeirwn/zacharias/EXTRAE_INSTALL --enable-merge-in-trace --
enable-sampling --enable-pthread --enable-xml --without-mpi --without-unwind --without-
dyninst --with-papi=/home/xeirwn/zacharias/PAPI_INSTALL --no-create --no-recursion

CC was gcc
CFLAGS was -g -O2 -fno-optimize-sibling-calls -Wall -W
CXX was g++
CXXFLAGS was -g -O2 -Wall -W

MPI support seems to be disabled
PACX support seems to be disabled
LIBXML2_HOME points to /usr and the directory exists .. OK
PAPI_HOME points to /home/xeirwn/zacharias/PAPI_INSTALL and the directory exists .. OK
DYNINST support seems to be disabled
UNWINDing support seems to be disabled (or not needed)

Please, report bugs to tools@bsc.es
```

Εικόνα 2.2.2

2.3 Περιγραφή XML αρχείου

Το εργαλείο EXTRAE ρυθμίζεται μέσω ενός XML αρχείου, το οποίο καθορίζεται μέσω της μεταβλητής περιβάλλοντος EXTRAE_CONFIG_FILE. Κάθε τμήμα του XML αρχείου έχει ένα enabled πεδίο το οποίο επιτρέπει (“yes”) ή απαγορεύει (“no”) την επεξεργασία του τμήματος αυτού και όλων των υπό-τμημάτων που βρίσκονται μέσα σε αυτό. Μερικές φορές τα XML τμήματα χρησιμοποιούνται για καθορισμό χρόνου, όπως για παράδειγμα χρονική διάρκεια. Για αυτό τον σκοπό υπάρχει η ακόλουθη κωδικοποίηση:

- **n για nanoseconds**
- **u για microseconds**
- **m για milliseconds**
- **s για seconds**
- **M για minutes**
- **H για hours**
- **D για days**

Πιο κάτω θα αναφέρω και θα περιγράψω τα βασικότερα τμήματα ενός XML αρχείου. Για περισσότερα τμήματα και πληροφορίες για αυτά μπορείτε να συμβουλευτείτε το “Extrae User Guide” PDF το οποίο βρίσκεται στον σύνδεσμο: https://www.bsc.es/sites/default/files/public/computer_science/performance_tools/extrae-user-guide.pdf

- **Τμήμα “Trace Configuration”**

```
<?xml version='1.0'?>

<trace enabled="yes"
 home="/home/xeirwn/zacharias/EXTRAE_INSTALL"
 initial-mode="detail"
 type="paraver"
 xml-parser-id="Id: xml-parse.c 799 2011-10-20 16:02:03Z harald $"
>
```

Εικόνα 2.3.1

Στο τμήμα αυτό καθορίζεται η βασική συμπεριφορά της ιχνογράφησης. Το τμήμα αυτό, όπως φαίνεται και από την Εικόνα 2.3.1 μας παρέχει τις ακόλουθες επιλογές:

- ✓ **enabled** Γράφουμε “yes” αν επιθυμούμε την δημιουργία trace αρχείων.
- ✓ **home** Καθορίζουμε το μονοπάτι στο οποίο έχει εγκατασταθεί το εργαλείο EXTRAE.
- ✓ **initial-mode** Εδώ έχουμε τις δύο πιο κάτω επιλογές:
  - **detail** Παρέχει λεπτομερή ανάλυση του trace αρχείου.
  - **burst** Παρέχει συνοπτική ανάλυση του trace αρχείου (χωρίς πληροφορίες για OpenMP και MPI κλήσεις).
- ✓ **type** Εδώ, πάλι έχουμε δύο επιλογές:
  - **paraver** Τα ενδιάμεσα αρχεία που θα παράγει το EXTRAE προορίζονται για την δημιουργία trace αρχείων για το PARAVER.
  - **dimemas** Τα ενδιάμεσα αρχεία που θα παράγει το EXTRAE προορίζονται για την δημιουργία trace αρχείων για το DIMEMAS.
- ✓ **xml-parser-id** Χρησιμοποιείται για να ελέγξει εάν το σύστημα ανάλυσης (parsing scheme) ταιριάζει με το σύστημα αρχείων (file scheme) ή όχι.

ΠΡΟΣΟΧΗ! Το τμήμα <?xml version=’1.0’?> είναι υποχρεωτικό για όλα τα XML αρχεία και δεν πρέπει να αλλαχτεί.

- **Τμήμα “Pthread”**

```
<pthread enabled="yes">
  <locks enabled="no" />
  <counters enabled="yes" />
</pthread>
```

Εικόνα 2.3.2

Για να μπορούμε να εκμεταλλευτούμε το τμήμα αυτό θα πρέπει το EXTRAE να έχει εγκατασταθεί με την επιλογή --enable-pthread. Το τμήμα αυτό, όπως φαίνεται και από την Εικόνα 2.3.2 μας παρέχει τις ακόλουθες επιλογές:

- ✓ **pthread enabled** Γράφουμε “yes” αν επιθυμούμε την συλλογή πληροφοριών για τις pthread ρουτίνες.

- ✓ **locks enabled** Γράφουμε “yes” αν επιθυμούμε την συλλογή πληροφοριών για τα locks μέσα στις pthread ρουτίνες.
- ✓ **counters enabled** Γράφουμε “yes” αν επιθυμούμε την συλλογή μετρικών απόδοσης μέσα στις pthread ρουτίνες.

- **Τμήμα “Performance Counters”**

```
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-time="5s">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM
      <sampling enabled="yes" period="100000000">PAPI_TOT_CYC</sampling>
    </set>
    <set enabled="yes" domain="user" changeat-globalops="5">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_FP_INS
    </set>
  </cpu>
  <network enabled="yes" />
  <resource-usage enabled="yes" />
</counters>
```

Εικόνα 2.3.3

Το τμήμα αυτό, όπως φαίνεται και από την Εικόνα 2.3.3, μας παρέχει την δυνατότητα συλλογής μετρικών απόδοσης για διάφορα συστατικά του συστήματος, όπως:

- ✓ **Τον επεξεργαστή**. Αυτό επιτυγχάνεται με τη χρήση του PAPI (επιλογή --with-papi κατά την εγκατάσταση του EXTRAE) στο οποίο θα αναφερθώ στο **Κεφάλαιο 3**.

  Οι μετρικές απόδοσης του επεξεργαστή ρυθμίζονται μέσα στο υπό-τμήμα **<cpu>**. Μέσα στο τμήμα <cpu> ο χρήστης μπορεί να δηλώσει όσα sets επιθυμεί, τοποθετώντας τις ρυθμίσεις για κάθε set ανάμεσα στο **<set>** και **</set>**, όμως μόνο ένα set θα χρησιμοποιείται ανά πάσα στιγμή. Ο χρήστης μπορεί να καθορίσει στο τμήμα <cpu> από ποιο set θα ξεκινά, γράφοντας τον αριθμό του επιθυμητού set στο πεδίο starting-set-distribution.

  Σε κάθε set ο χρήστης μπορεί να καθορίσει ποιες μετρικές επιθυμεί να συλλέξει γράφοντας το κανονικό όνομα της μετρικής αυτής, σύμφωνα με το PAPI (**Κεφάλαιο 3**). Στο παράδειγμα της Εικόνας 2.3.3, το

πρώτο set συλλέγει τις μετρικές PAPI_TOT_INS (total instructions), PAPI_TOT_CYC (total cycles) και PAPI_L1_DCM (1st level cache misses). Το δεύτερο set συλλέγει τις μετρικές PAPI_TOT_INS (total instructions), PAPI_TOT_CYC (total cycles) και PAPI_FP_INS (floating point instructions).

ΠΡΟΣΟΧΗ! Το EXTRAE μπορεί να μαζέψει μέχρι οκτώ (8) μετρικές απόδοσης την ίδια χρονική στιγμή. Αυτό δεν σημαίνει απαραίτητα οκτώ (8) PAPI μετρικές απόδοσης, καθώς όπως θα διαβάσετε και στο **Κεφάλαιο 3** υπάρχουν κάποιες μετρικές απόδοσης που για να προκύψουν, χρησιμοποιούν και άλλες μετρικές. Άρα, στην ουσία κάποιες φορές δεν χρησιμοποιείται μια μετρική όπως φαίνεται, αλλά περισσότερες.

Επίσης, αν το EXTRAE έχει εγκατασταθεί με την επιλογή --enable-sampling, ο χρήστης μπορεί να καθορίσει κάθε set να συλλέγει πληροφορίες για τις μετρικές απόδοσης του κάθε φορά που μια μετρική απόδοσης φτάσει μια προκαθορισμένη τιμή. Η μετρική απόδοσης που θα χρησιμοποιηθεί για την δειγματοληψία πρέπει να περιέχεται μέσα στο set. Στο παράδειγμα της Εικόνας 2.3.3, το πρώτο set συλλέγει τις μετρικές απόδοσής του κάθε 100M κύκλους. Ακόμη, για κάθε set μπορούμε να καθορίσουμε σε ποια κατάσταση πρέπει να βρίσκεται η εφαρμογή έτσι ώστε το set να συλλέγει πληροφορίες για τις μετρικές απόδοσής του. Αυτό επιτυγχάνεται με την ανάθεση μιας από τις πιο κάτω τρεις επιλογές στο πεδίο domain.

- **kernel** Η εφαρμογή τρέχει σε λειτουργία πυρήνα.
- **user** Η εφαρμογή τρέχει σε λειτουργία χώρου χρήστη (user-space).
- **all** Ανεξαρτήτως του που τρέχει η εφαρμογή.

Τέλος, το set που χρησιμοποιείται μπορεί να αλλάξει είτε αυτόματα, είτε με εντολή του χρήστη. Οι δύο εντολές με τις οποίες μπορεί ο χρήστης να αλλάξει το set που χρησιμοποιείται είναι οι εξής:

- **Extrae_previous_hwc_set**
- **Extrae_next_hwc_set**

Οι εντολές αυτές αποτελούν μέρος του **EXTRAE API** και θα επεξηγηθούν στο επόμενο τμήμα (**2.4**) αυτού του κεφαλαίου. Για

αυτόματη αλλαγή του set που χρησιμοποιείται, πάλι υπάρχουν δύο επιλογές:

- **Με βάση το χρόνο.** Προσθέτοντας το πεδίο changeat-time και δίπλα τη διάρκεια χρήσης του set.
- **Με βάση τα MPI global operations.** Προσθέτοντας το πεδίο changeat-globalops και δίπλα τον αριθμό των global operations.

Αν η τιμή του πεδίου changeat-time ή changeat-globalops είναι μηδέν, τότε το set δεν αλλάζει αυτόματα.

- ✓ **Το δίκτυο**. Αυτό επιτρέπεται μόνο σε συστήματα με Myrinet GM/MX δίκτυα. Αν το <network> είναι ενεργοποιημένο, οι μετρικές απόδοσης του δικτύου εμφανίζονται στο τέλος της εκτέλεσης της εφαρμογής, δίνοντας μια περιγραφή της ολικής εκτέλεσης.
- ✓ **Το λειτουργικό σύστημα**. Αυτό επιτρέπεται με χρήση της κλήσης συστήματος getrusage(2), όταν το <resource-usage> είναι ενεργοποιημένο. Όπως και οι μετρικές απόδοσης του δικτύου, έτσι και οι μετρικές απόδοσης του Λ.Σ. εμφανίζονται στο τέλος της εκτέλεσης της εφαρμογής, δίνοντας μια περιγραφή της ολικής εκτέλεσης.

- **Τμήμα "Storage Management"**

Στο τμήμα αυτό καθορίζονται κάποια στοιχεία σχετικά με την αποθήκευση των αρχείων που παράγει το EXTRAE.

```
<storage enabled="yes">
  <trace-prefix enabled="yes">TRACE_0</trace-prefix>
  <size enabled="no">5</size>
  <temporal-directory enabled="yes">/home/xeirwn/multiTSU</temporal-directory>
  <final-directory enabled="yes">/home/xeirwn/multiTSU</final-directory>
  <gather-mpits enabled="no" />
</storage>
```

Εικόνα 2.3.4

Το τμήμα αυτό, όπως φαίνεται και από την Εικόνα 2.3.4 μας παρέχει τις ακόλουθες επιλογές:

- ✓ **trace prefix** Στο πεδίο αυτό μπορούμε να καθορίσουμε το πρόθεμα των ενδιάμεσων αρχείων (*.mpit και *.mpits) που παράγει το EXTRAE. Η προκαθορισμένη τιμή είναι "TRACE".

- ✓ **size** Επιτρέπει στον χρήστη να περιορίσει το μέγιστο μέγεθος (σε MB) του κάθε ενδιάμεσου αρχείου που παράγει το EXTRAE.
- ✓ **temporal-directory** Που θα αποθηκευτούν τα ενδιάμεσα αρχεία κατά την διάρκεια εκτέλεσης της εφαρμογής. Προκαθορισμένη τιμή είναι ο τρέχον φάκελος. Αν ο φάκελος δεν υπάρχει, τον δημιουργεί.
- ✓ **final-directory** Που θα αποθηκευτούν τα ενδιάμεσα αρχεία όταν τελειώσει η εκτέλεση της εφαρμογής. Προκαθορισμένη τιμή είναι ο τρέχον φάκελος. Αν ο φάκελος δεν υπάρχει, τον δημιουργεί.
- ✓ **gather-mpits** Αν το σύστημα δεν παρέχει ένα global filesystem, τα ενδιάμεσα αρχεία θα είναι σκορπισμένα στους διάφορους κόμβους. Με ενεργοποίηση αυτής της επιλογής θα χρησιμοποιηθεί το MPI (αν το EXTRAE έχει εγκατασταθεί με την επιλογή --with-mpi) για να μαζέψει όλα τα ενδιάμεσα αρχεία στον ριζικό κόμβο.

- **Τμήμα "Buffer Management"**

  Στο τμήμα αυτό καθορίζονται κάποια στοιχεία σχετικά με την συμπεριφορά του buffer ιχνογράφησης. Ακόμη και αν το enabled πεδίο είναι "no", το μέγεθος του buffer ορίζεται στα 500k events (προκαθορισμένη τιμή).

```
<buffer enabled="yes">
  <size enabled="yes">1000000</size>
  <circular enabled="no" />
</buffer>
```

Εικόνα 2.3.5

Το τμήμα αυτό, όπως φαίνεται και από την Εικόνα 2.3.5 μας παρέχει τις ακόλουθες επιλογές:

- ✓ **size** Καθορισμός του μεγέθους του buffer σε N events.
- ✓ **circular** Αν το πεδίο είναι ενεργοποιημένο, το buffer που θα χρησιμοποιηθεί θα είναι circular και θα αποδεσμευτεί μόνο όταν παραχθούν και τα τελευταία events της διαδικασίας ιχνογράφησης.

- **Τμήμα “Merge”**

  Αν το τμήμα αυτό είναι ενεργοποιημένο και το EXTRAE έχει εγκατασταθεί με την επιλογή --enable-merge-in-trace, η διαδικασία συγχώνευσης των ενδιάμεσων αρχείων θα ξεκινήσει αυτόματα μετά που θα τελειώσει η εκτέλεση της εφαρμογής. Το τελευταίο πεδίο αυτού του τμήματος θα χρησιμοποιηθεί ως το όνομα που θα πάρει το τελικό *.prv αρχείο (pthread_example.prv στο παράδειγμα της Εικόνας 2.3.6).

```
<merge enabled="no"
  synchronization="default"
  binary="pthread_example"
  tree-fan-out="16"
  max-memory="512"
  joint-states="yes"
  keep-mpits="yes"
  sort-addresses="yes"
>
  pthread_example.prv
</merge>
```

Εικόνα 2.3.6

Το τμήμα αυτό, όπως φαίνεται και από την Εικόνα 2.3.6 μας παρέχει τις ακόλουθες επιλογές:

- ✓ **synchronization** Μπορεί να πάρει τιμές default, node, task και no. Το πεδίο αυτό καθορίζει το πώς θα συγχρονιστούν τα task clocks (η τιμή default αντιστοιχεί στην τιμή node).
- ✓ **binary** Δείχνει στο binary το οποίο εκτελείται. Θα χρησιμοποιηθεί για μετάφραση των διευθύνσεων που έχουν συλλεχθεί σε αναφορές στον πηγαίο κώδικα.
- ✓ **tree-fan-out** Μόνο για MPI εκτελέσεις.
- ✓ **max-memory** Περιορίζει την ενδιάμεση διαδικασία συγχώνευσης έτσι ώστε να τρέχει μέχρι το καθορισμένο όριο σε MB.
- ✓ **joint-states** Μπορεί να πάρει τιμές yes και no. Καθορίζει αν το *.prv αρχείο που παράγεται θα διαχωρίζει ή ενώνει τις ίδιες και συνεχόμενες καταστάσεις (states). Η προκαθορισμένη τιμή είναι yes.
- ✓ **keep-mpits** Αν θα κρατήσει τα ενδιάμεσα (*.mpit) αρχεία αφού τελειώσει με την διαδικασία συγχώνευσης.

- ✓ **sort-addresses** Αν θα ταξινομήσει όλες τις διευθύνσεις που κάνουν αναφορά στον πηγαίο κώδικα. Η προκαθορισμένη τιμή είναι το yes.

Για περισσότερες πληροφορίες μπορείτε να διαβάσετε το τμήμα (**2.6**) αυτού του κεφαλαίου.

2.4 EXTRAE API

Το εργαλείο EXTRAE μας προσφέρει κάποιες συναρτήσεις οι οποίες μας βοηθούν στην καλύτερη ανάλυση μιας εφαρμογής. Οι συναρτήσεις αυτές αποτελούν το EXTRAE API, το οποίο χωρίζεται σε δύο επίπεδα:

- **Βασικό API**

  Οι συναρτήσεις του βασικού API είναι καθορισμένες στο:
  ${EXTRAE_HOME}/include/extrae_user_events.h
  και για να μπορούν να χρησιμοποιηθούν μέσα στον κώδικα μιας εφαρμογής θα πρέπει να κάνουμε **#include "extrae_user_events.h"** στην αρχή του κώδικα μας.

  Πιο κάτω θα αναφέρω και θα περιγράψω τις σημαντικότερες συναρτήσεις που προσφέρει το βασικό API. Για περισσότερες συναρτήσεις και πληροφορίες για αυτές μπορείτε να συμβουλευτείτε το "Extrae User Guide" PDF το οποίο βρίσκεται στον σύνδεσμο:
  https://www.bsc.es/sites/default/files/public/computer_science/performance_tools/extrae-user-guide.pdf

  - ✓ **void Extrae_init(void)** Η συνάρτηση αυτή αρχικοποιεί την βιβλιοθήκη ιχνογράφησης. Μπαίνει στην αρχή του κώδικα που επιθυμούμε να αναλύσουμε.
  - ✓ **void Extrae_fini(void)** Η συνάρτηση αυτή οριστικοποιεί την βιβλιοθήκη ιχνογράφησης και μεταφέρει τα ενδιάμεσα buffers ιχνογράφησης στον δίσκο. Μπαίνει στο τέλος του κώδικα που επιθυμούμε να αναλύσουμε.

- ✓ **void Extrae_event(unsigned type, unsigned value)** Η συνάρτηση αυτή προσθέτει ένα χρονολογημένο event στο trace αρχείο. Το event αυτό έχει δύο πεδία: type και value. Κάποιες περιπτώσεις χρήσης αυτής της συνάρτησης είναι:
  - **Ο προσδιορισμός της επανάληψης ενός βρόχου**. Ο χρήστης μπορεί να καθορίσει ένα συγκεκριμένο type για ένα βρόχο και το value να αλλάζει κάθε φορά ανάλογα με τον αριθμό επανάληψης, όπως φαίνεται στην Εικόνα 2.4.1.

```
for (i = 1; i <= MAX_ITERS; i++){

        Extrae_event (1000, i);
        [original loop code]

}

Extrae_event (1000, 0);
```

Εικόνα 2.4.1

Το τελευταίο Extrae_event δηλώνει το τέλος του βρόχου με το να αναθέτει στο πεδίο value την τιμή μηδέν.

  - **Ο προσδιορισμός συναρτήσεων του χρήστη**. Επιλέγοντας ένα σταθερό αριθμό type (6000019 στο παράδειγμα της Εικόνας 2.4.2) και διαφορετικά values για κάθε συνάρτηση. Value με τιμή ίση με μηδέν δηλώνει το τέλος της συνάρτησης.

```
void routine1 (void){

        Extrae_event (6000019, 1);
        [routine 1 code]
        Extrae_event (6000019, 0);

}
```

```
void routine2 (void){

        Extrae_event (6000019, 2);
        [routine 2 code]
        Extrae_event (6000019, 0);

}
```

Εικόνα 2.4.2

  - **Προσδιορισμός οποιουδήποτε σημείου της εφαρμογής** με χρήση ενός μοναδικού συνδυασμού τιμών type και value.

- ✓ **void Extrae_counters(void)** Η συνάρτηση αυτή προσθέτει στο trace αρχείο την τιμή των μετρικών απόδοσης του ενεργοποιημένου hardware set, όπως αυτό περιγράφηκε στο μέρος **2.3 – Τμήμα "Performance Counters"**.
- ✓ **void Extrae_eventandcounters(unsigned type, unsigned value)** Η συνάρτηση αυτή προσθέτει ένα χρονολογημένο event στο trace αρχείο και μαζί με αυτό και την τιμή των μετρικών απόδοσης του ενεργοποιημένου hardware set.
- ✓ **void Extrae_shutdown(void)** Η συνάρτηση αυτή απενεργοποιεί την ιχνογράφηση της εφαρμογής.
- ✓ **void Extrae_restart(void)** Η συνάρτηση αυτή ενεργοποιεί την ιχνογράφηση της εφαρμογής.

  Σκοπός των δύο αυτών συναρτήσεων (shutdown και restart) είναι:
  - o Η μείωση του μεγέθους του trace αρχείου.
  - o Η επικέντρωση στα σημαντικά σημεία της εφαρμογής.
- ✓ **void Extrae_previous_hwc_set(void)** Η συνάρτηση αυτή ενεργοποιεί το προηγούμενο set μετρικών απόδοσης του XML αρχείου.
- ✓ **void Extrae_next_hwc_set(void)** Η συνάρτηση αυτή ενεργοποιεί το επόμενο set μετρικών απόδοσης του XML αρχείου.
- ✓ **void Extrae_set_tracing_tasks(int from, int to)** Η συνάρτηση αυτή επιτρέπει στον χρήστη να επιλέξει από ποιες διεργασίες (tasks) επιθυμεί να αποθηκεύσει πληροφορίες μέσα στο trace αρχείο.

- **Extended API**

  Οι συναρτήσεις του Extended API είναι καθορισμένες στο:

  ${EXTRAE_HOME}/include/extrae_types.h

  και για να μπορούν να χρησιμοποιηθούν μέσα στον κώδικα μιας εφαρμογής θα πρέπει να κάνουμε **#include "extrae_types.h"** στην αρχή του κώδικα μας. Εκτός από τις συναρτήσεις, το Extended API ορίζει και τις ακόλουθες δύο εξειδικευμένες δομές:

  - ✓ **struct extrae_UserCommunication** Η οποία χρησιμοποιείται για κωδικοποίηση ενός event και μετατροπή του σε PARAVER επικοινωνία, όταν βρεθεί το ζευγάρι του αντίστοιχου event.

```
struct extrae_UserCommunication
{
  extrae_user_communication_types_t type;
  extrae_comm_tag_t tag;
  unsigned size; /* size_t? */
  extrae_comm_partner_t partner;
  extrae_comm_id_t id;
};
```

Εικόνα 2.4.3

Όπως φαίνεται και στην Εικόνα 2.4.3 η δομή αυτή περιέχει τα ακόλουθα πεδία:

- **type** Διαθέσιμες επιλογές είναι:
  - EXTRAE_USER_SEND, αν το event αφορά αποστολή.
  - EXTRAE_USER_RECV, αν το event αφορά παραλαβή.
- **tag** Η tag πληροφορία στην καταγραμμένη επικοινωνία.
- **size** Η πληροφορία μεγέθους στην καταγραμμένη επικοινωνία.
- **partner** Το ζευγάρι αυτής της επικοινωνίας. Ζεύγη από μηδέν μέχρι N-1 θεωρούνται μεταξύ διεργασιών, ενώ όλα τα threads μοιράζονται μία ουρά επικοινωνίας.
- **id** Αναγνωριστικό που χρησιμοποιείται για να ταιριάξει επικοινωνίες μεταξύ ζευγαριών.

✓ **struct extrae_CombinedEvents** Η οποία χρησιμοποιείται για να παράγει events που περιέχουν πολλαπλά είδη πληροφορίας, στην ίδια χρονική στιγμή.

```
struct extrae_CombinedEvents{
        int HardwareCounters;
        int Callers;
        int UserFunction;
        unsigned nEvents;
        extrae_type_t *Types;
        extrae_value_t *Values;
        unsigned nCommunications;
        extrae_user_communication_t *Communications;
};
```

Εικόνα 2.4.4

Όπως φαίνεται και στην Εικόνα 2.4.4 η δομή αυτή περιέχει τα ακόλουθα πεδία:

- **HardwareCounters** Παίρνει μη-μηδενική τιμή αν το event αυτό πρέπει να συλλέξει hardware μετρικές απόδοσης.
- **Callers** Παίρνει μη-μηδενική τιμή αν το event αυτό πρέπει να συλλέξει callstack πληροφορίες.
- **UserFunction** Διαθέσιμες επιλογές είναι:
  - EXTRAE_USER_FUNCTION_NONE, αν το event δεν πρέπει να παρέχει πληροφορίες για συναρτήσεις του χρήστη.
  - EXTRAE_USER_FUNCTION_ENTER, αν το event δείχνει την αρχή μιας συνάρτησης του χρήστη.
  - EXTRAE_USER_FUNCTION_LEAVE, αν το event δείχνει το τέλος μιας συνάρτησης του χρήστη.
- **nEvents** Ο αριθμός των events που δίνονται στα πεδία Types και Values.
- **Types** Δείκτης που περιέχει nEvents types τα οποία φυλάγονται στο trace αρχείο.
- **Values** Δείκτης που περιέχει nEvents values τα οποία φυλάγονται στο trace αρχείο.
- **nCommunications** Ο αριθμός των επικοινωνιών που δίνονται στο πεδίο Communications.
- **Communications** Δείκτης σε δομές **extrae_UserCommunication**. Περιέχει nCommunications στοιχεία που αντιπροσωπεύουν επικοινωνίες.

Πιο κάτω θα αναφέρω και θα περιγράψω τις σημαντικότερες συναρτήσεις που προσφέρει το Extended API. Για περισσότερες συναρτήσεις και πληροφορίες για αυτές μπορείτε να συμβουλευτείτε το "Extrae User Guide" PDF το οποίο βρίσκεται στον σύνδεσμο:
https://www.bsc.es/sites/default/files/public/computer_science/performance_tools/extrae-user-guide.pdf

- ✓ **void Extrae_init_UserCommunication(struct extrae_UserCommunication *)**
  Για αρχικοποίηση μιας δομής extrae_UserCommunication.
- ✓ **void Extrae_init_CombinedEvents(struct extrae_CombinedEvents *)**
  Για αρχικοποίηση μιας δομής extrae_CombinedEvents.

- ✓ **void Extrae_emit_CombinedEvents(struct extrae_CombinedEvents *)**
  Για μεταφορά των events μιας δομής extrae_CombinedEvents στο trace αρχείο.

Μέσα από την περιγραφή του Extended API κάποιος καταλαβαίνει πως με χρήση των δυνατοτήτων που μας προσφέρει μπορούμε να καταγράψουμε τις διάφορες επικοινωνίες που συμβαίνουν μεταξύ των νημάτων και διεργασιών μιας εφαρμογής. Οι επικοινωνίες αυτές εμφανίζονται αργότερα στο PARAVER μέσω μιας κίτρινης γραμμής η οποία ενώνει τα νήματα ή τις διεργασίες μεταξύ των οποίων διεξάγεται η επικοινωνία. Στην Εικόνα 2.4.5 μπορείτε να δείτε τον κώδικα που αναπαριστά μια επικοινωνία μεταξύ δύο νημάτων της ίδιας εφαρμογής.

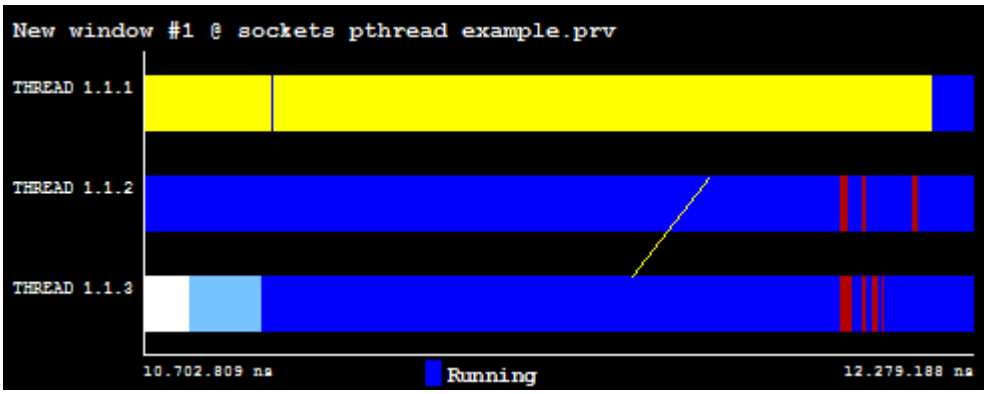
```
struct extrae_CombinedEvents events;
struct extrae_UserCommunication comm;
unsigned types[2] = { 123456, 123457 } ;
unsigned values[2] = { 1, 2 };
Extrae_init_UserCommunication (&comm);
comm.type = EXTRAE_USER_SEND;
comm.partner = 0;
comm.tag = 1234;
comm.size = 1024;
comm.id = 0xdeadbeef;
Extrae_init_CombinedEvents (&events);
events.nCommunications = 1;
events.Communications = &comm;
events.nEvents = 2;
events.Types = types;
events.Values = values;

Extrae_emit_CombinedEvents (&events);

//SENDING MESSAGE
if( write(sock,buffer,sizeof(buffer))<0 ){
  perror("write");
  exit(1);
}
```

```
struct extrae_CombinedEvents events;
struct extrae_UserCommunication comm;
unsigned types[2] = { 123456, 123457 } ;
unsigned values[2] = { 1, 2 };
Extrae_init_UserCommunication (&comm);
comm.type = EXTRAE_USER_RECV;
comm.partner = 0;
comm.tag = 1234;
comm.size = 1024;
comm.id = 0xdeadbeef;
Extrae_init_CombinedEvents (&events);
events.nCommunications = 1;
events.Communications = &comm;
events.nEvents = 2;
events.Types = types;
events.Values = values;

//RECEIVING MESSAGE
if( read(newsock,buffer,sizeof(buffer))<0 ){
  perror("read");
  exit(1);
}

Extrae_emit_CombinedEvents(&events);
```

Εικόνα 2.4.5

Στα αριστερά είναι ο κώδικας του νήματος που αποστέλλει το μήνυμα και στα δεξιά ο κώδικας του νήματος που παραλαμβάνει το μήνυμα. Παρατηρήστε πως στον κώδικα που γίνεται η αποστολή του μηνύματος η συνάρτηση **Extrae_emit_CombinedEvents** καλείται πριν να γίνει η αποστολή, ενώ στον κώδικα που γίνεται η παραλαβή του μηνύματος η συνάρτηση αυτή καλείται μετά που θα γίνει η παραλαβή. Αποτέλεσμα του κώδικα αυτού είναι η

δημιουργία ενός *.prv αρχείου το οποίο όταν ανοιχτεί με το PARAVER έχει την μορφή που φαίνεται στην Εικόνα 2.4.6.



Εικόνα 2.4.6

ΠΡΟΣΟΧΗ! Το εργαλείο EXTRAE δεν παρέχει την δυνατότητα για καταγραφή επικοινωνιών που συμβαίνουν μεταξύ διαφορετικών εφαρμογών. Αυτό μπορείτε να το διαπιστώσετε και από e-mail που έχω από το Barcelona Supercomputing Center (BSC) και το οποίο αποδεικνύει το πιο πάνω. Το e-mail αυτό μπορείτε να το διαβάσετε στο **ΠΑΡΑΡΤΗΜΑ Α** (σελ. Α-3). Εδώ ακριβώς, προκύπτει το πρόβλημα που ανέφερα και στο **Κεφάλαιο 1 – Εισαγωγή**, καθώς στο DDM μοντέλο έχουμε επικοινωνίες μεταξύ διαφορετικών εφαρμογών.

Ως μέτρο αντιμετώπισης του προβλήματος αυτού σκέφτηκα όπως χρησιμοποιώ την συνάρτηση **Extrae_event** όπου υπάρχει επικοινωνία. Πιο συγκεκριμένα, θα τοποθετώ μια συνάρτηση **Extrae_event** ακριβώς πριν το σημείο που γίνεται η αποστολή και μια ακριβώς μετά που γίνεται η παραλαβή. Οι δύο παράμετροι της συνάρτησης θα καθορίζουν τα ακόλουθα:

- ✓ **type** Το είδος της επικοινωνίας. Για το σκοπό αυτό αποφάσισα πως δέκα ακέραιες τιμές θα είναι κρατημένες για σκοπούς επικοινωνίας και δεν πρέπει να χρησιμοποιούνται στην παράμετρο type για οποιοδήποτε άλλο λόγο (Πίνακας 2.4.1).
- ✓ **value** Ποια εφαρμογή είναι ο παραλήπτης σε περίπτωση που το event είναι send, ή ποια εφαρμογή είναι ο αποστολέας σε περίπτωση που το event είναι receive.

TYPE NUM	TYPE STRING	MATCHES WITH	TYPE STRING	TYPE NUM
2	SEND MESSAGE	↔	RECEIVE MESSAGE	3
4	SEND COMMAND	↔	RECEIVE COMMAND	5
6	SEND TYPE_3	↔	RECEIVE TYPE_3	7
8	SEND TYPE_4	↔	RECEIVE TYPE_4	9
10	SEND TYPE_5	↔	RECEIVE TYPE_5	11

Πίνακας 2.4.1

Στην Εικόνα 2.4.7 μπορείτε να δείτε τον κώδικα που αναπαριστά μια επικοινωνία μεταξύ δύο διαφορετικών εφαρμογών.

```
//APPLICATION 2 CODE
//SENDING MESSAGE
Extrae_event(2,1);
if( write(sock,buffer,sizeof(buffer))<0 ){
  perror("write");
  exit(1);
}
```

```
//APPLICATION 1 CODE
//RECEIVING MESSAGE
if( read(newsock,buffer,sizeof(buffer))<0 ){
    perror("read");
    exit(1);
}
Extrae_event(3,2);
```

Εικόνα 2.4.7

Στα αριστερά είναι ο κώδικας της εφαρμογής 2 η οποία αποστέλλει ένα μήνυμα (type=2) στην εφαρμογή 1 (value=1), και στα δεξιά είναι ο κώδικας της εφαρμογής 1 η οποία παραλαμβάνει το μήνυμα (type=3) από την εφαρμογή 2 (value=2). Αποτέλεσμα του κώδικα αυτού είναι η δημιουργία ενός *.prv αρχείου το οποίο θα περιέχει αυτά τα δύο events. Όταν το αρχείο αυτό περάσει από το πρόγραμμα Parser που έγραψα, τα δύο αυτά events θα ζευγαρωθούν και θα μετατραπούν σε επικοινωνία. Μπορείτε να βρείτε περισσότερες λεπτομέρειες για το πρόγραμμα Parser που έγραψα στο **Κεφάλαιο 5 – Αλγόριθμος Parser**.

Με αυτό τον τρόπο κατάφερα να λύσω το πρόβλημα που παρουσιάστηκε, αφού με το πρόγραμμα Parser εξυπηρετείται η απαίτηση του DDM μοντέλου για αναπαράσταση επικοινωνιών μεταξύ διαφορετικών εφαρμογών.

2.5 Εκτέλεση εφαρμογής με το εργαλείο EXTRAE

Για να καταφέρουμε να εκτελέσουμε μια εφαρμογή με το εργαλείο EXTRAE πρέπει να ακολουθήσουμε τα τέσσερα παρακάτω βήματα:

- Ορισμός των δύο μεταβλητών περιβάλλοντος PAPI_HOME και EXTRAE_HOME, έτσι ώστε να δείχνουν στα μονοπάτια στα οποία έχουν εγκατασταθεί το PAPI και το EXTRAE αντίστοιχα. Για παράδειγμα:
  - ✓ export PAPI_HOME=/home/xeirwn/zacharias/PAPI_INSTALL
  - ✓ export EXTRAE_HOME=/home/xeirwn/zacharias/EXTRAE_INSTALL

- Ακολούθως, για εφαρμογές που χρησιμοποιούν pthreads πρέπει να μεταγλωττίσουμε το αρχείο κώδικά μας με την εντολή:
  - ✓ gcc **[C file]** -O -g -I$EXTRAE_HOME/include -L$EXTRAE_HOME/lib -lpttrace -L$PAPI_HOME/lib -lpapi -o **[executable]**

- Έπειτα, πρέπει να ορίσουμε τις δύο μεταβλητές περιβάλλοντος LD_LIBRARY_PATH και EXTRAE_CONFIG_FILE, έτσι ώστε η πρώτη να δείχνει στις βιβλιοθήκες του PAPI και του EXTRAE και η δεύτερη στο XML αρχείο που θα χρησιμοποιηθεί. Για παράδειγμα:
  - ✓ export LD_LIBRARY_PATH=$EXTRAE_HOME/lib:$PAPI_HOME/lib
  - ✓ export EXTRAE_CONFIG_FILE=extrae.xml

- Τέλος, εκτελούμε το executable αρχείο που προέκυψε από την μεταγλώττιση του πηγαίου κώδικά μας.

Αποτέλεσμα της εκτέλεσης μιας εφαρμογής με το εργαλείο EXTRAE είναι η δημιουργία δύο ειδών αρχείων, τα οποία θα βρίσκονται στο φάκελο final-directory που καθορίζεται στο τμήμα “Storage Management” του XML αρχείου. Τα αρχεία αυτά είναι:

- Τα ***.mpit** αρχεία. Ο αριθμός αυτών των αρχείων είναι ίσος με τον αριθμό των νημάτων και των διεργασιών που έτρεξαν την εφαρμογή. Κάθε ένα από αυτά περιέχει πληροφορίες σχετικά με το νήμα ή την διεργασία που έτρεξε.

- Το ***.mpits** αρχείο το οποίο περιέχει μια λίστα με τα *.mpit αρχεία που δημιουργήθηκαν (Εικόνα 2.5.1).

```
/home/xeirwn/set-0/TRACE.0000002369000000000000.mpit on teras named
/home/xeirwn/set-0/TRACE.0000002369000000000001.mpit on teras named
/home/xeirwn/set-0/TRACE.0000002369000000000002.mpit on teras named
```

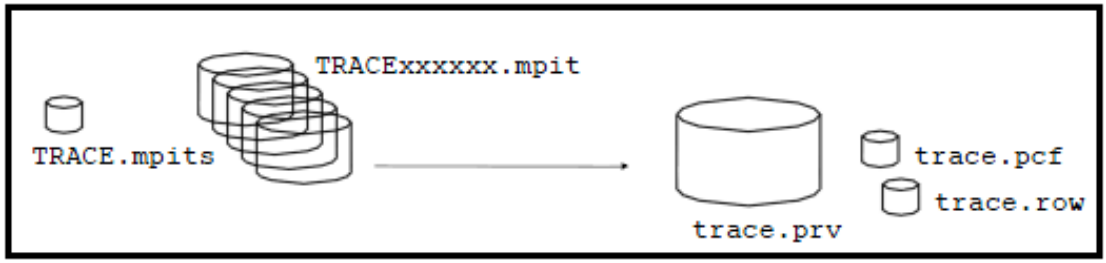
Εικόνα 2.5.1

Όπως παρατηρείτε από την Εικόνα 2.5.1 το *.mpits αρχείο έχει τα απόλυτα μονοπάτια στα οποία βρίσκονται τα *.mpit αρχεία. Μέσω της πληροφορίας που παίρνουμε από το συγκεκριμένο *.mpits αρχείο, μπορούμε να συμπεράνουμε πως η συγκεκριμένη εφαρμογή εκτελέστηκε από τρία νήματα/διεργασίες.

2.6 Διαδικασία συγχώνευσης *.mpit αρχείων (Merging Process)

Σε περίπτωση που το εργαλείο EXTRAE δεν έχει εγκατασταθεί με υποστήριξη της αυτόματης διαδικασίας συγχώνευσης (επιλογή --enable-merge-in-trace) και το πεδίο merge enabled του XML αρχείου δεν είναι ενεργοποιημένο, η διαδικασία αυτή πρέπει να εκτελεστεί ρητά από εμάς.

Κατά την διαδικασία συγχώνευσης ο Merger (mpi2prv) χρησιμοποιεί τα ενδιάμεσα trace αρχεία (*.mpit) για να δημιουργήσει τρία διαφορετικά αρχεία με το ίδιο όνομα αλλά διαφορετικό extension (Εικόνα 2.6.1).



Εικόνα 2.6.1

Όπως φαίνεται και από την Εικόνα 2.6.1 τα τρία αυτά αρχεία είναι τα ακόλουθα:

- ***.prv αρχείο** Το τελικό trace αρχείο που θα χρησιμοποιήσει το PARAVER.
- ***.pcf αρχείο** Το configuration αρχείο που θα χρησιμοποιήσει το PARAVER
- ***.row αρχείο** Καθορίζει τις ετικέτες των αντικειμένων που περιέχει το *.prv.

Περισσότερες λεπτομέρειες για τα τρία αυτά αρχεία μπορείτε να διαβάσετε στο **Κεφάλαιο 4 – Περιγραφή *.prv, *.pcf και *.row αρχείων**.

Για να μπορεί ωστόσο το PARAVER να χρησιμοποιήσει τα αρχεία αυτά θα πρέπει να γράψουμε "paraver" στο πεδίο type του τμήματος "Trace Configuration" του XML αρχείου. Έτσι δηλώνουμε πως τα αρχεία που θα προκύψουν από την διαδικασία συγχώνευσης προορίζονται για το εργαλείο PARAVER.

Ο PARAVER merger (mpi2prv) μας προφέρει κάποιες επιλογές κατά την διαδικασία συγχώνευσης. Πιο κάτω θα αναφέρω και θα περιγράψω τις σημαντικότερες επιλογές που προσφέρονται. Για περισσότερες επιλογές και πληροφορίες για αυτές μπορείτε να συμβουλευτείτε το "Extrae User Guide" PDF το οποίο βρίσκεται στον σύνδεσμο: https://www.bsc.es/sites/default/files/public/computer_science/performance_tools/extrae-user-guide.pdf

- **-o** Μας επιτρέπει να επιλέξουμε το όνομα των τριών αρχείων που θα δημιουργήσει ο PARAVER merger. Αν δεν δώσουμε την επιλογή αυτή, η προκαθορισμένη τιμή είναι "EXTRAE_Paraver_trace".
- **-e BINARY** Χρησιμοποιεί το BINARY για να μεταφράσει διευθύνσεις που είναι αποθηκευμένες μέσα στα ενδιάμεσα trace αρχεία, σε χρήσιμες πληροφορίες (για παράδειγμα ονόματα συναρτήσεων και γραμμή στον πηγαίο κώδικα). Απαραίτητη προϋπόθεση είναι η εφαρμογή να μεταγλωττιστεί με την επιλογή **–g**.
- **-f FILE.mpits** Όπου το FILE.mpits είναι το *.mpits αρχείο που προέκυψε από την εκτέλεση της εφαρμογής με το εργαλείο EXTRAE. Έτσι αντί να πρέπει να δώσουμε ένα σύνολο από *.mpit αρχεία, ο PARAVER merger χρησιμοποιεί αυτό το αρχείο για να βρει τα *.mpit αρχεία που θα συγχωνεύσει.
- -- Δεν αναφέρεται στο PDF, αλλά μπορείτε να το δείτε σε e-mail που έχω από το BSC. Το e-mail αυτό μπορείτε να το διαβάσετε στο **ΠΑΡΑΡΤΗΜΑ Α** (σελ. Α-1). Χρησιμοποιείται όταν θέλουμε να κάνουμε συγχώνευση των ενδιάμεσων trace αρχείων που προέκυψαν από εκτελέσεις διαφορετικών εφαρμογών (**./mpi2prv -f TRACE1.mpits -- -f TRACE2.mpits**).

Ένα παράδειγμα εκτέλεσης του PARAVER merger (mpi2prv) με τις πιο πάνω επιλογές που μας προσφέρει, είναι το ακόλουθο:

${EXTRAE_HOME}/bin/mpi2prv -f TRACE.mpits -e prog -o trace.prv

Σε περίπτωση που επιθυμούμε να προσθέσουμε στο τελικό trace αρχείο επιπλέον πληροφορίες σχετικά με τα events που τοποθετήσαμε μέσα στην εφαρμογή μας, μπορούμε να το πετύχουμε χρησιμοποιώντας την μεταβλητή περιβάλλοντος **EXTRAE_LABELS.** Εμείς απλά γράφουμε σε ένα αρχείο τις επιπλέον πληροφορίες σχετικά με τα δικά μας types και values που τοποθετήσαμε μέσα στον κώδικα της εφαρμογής μας (Εικόνα 2.6.2). Ακολούθως, ορίζουμε την μεταβλητή περιβάλλοντος **EXTRAE_LABELS** έτσι ώστε να δείχνει σε αυτό το αρχείο.

```
EVENT_TYPE
10 1 FUNCTION_1
VALUES
1 BEGIN
0 END



EVENT_TYPE
10 2 FUNCTION_2
VALUES
1 BEGIN
0 END
```

Εικόνα 2.6.2

Αφού το κάνουμε αυτό, εκτελούμε τον PARAVER merger (mpi2prv) και οι επιπλέον πληροφορίες του αρχείου που γράψαμε προσθέτονται μέσα στο *.pcf αρχείο.

2.7 Παραδείγματα

Σε αυτό το μέρος θα δείξω κάποια παραδείγματα εκτέλεσης εφαρμογών με το εργαλείο EXTRAE, καθώς και τα παράγωγα της κάθε εκτέλεσης. Συγκεκριμένα, θα δώσω δύο παραδείγματα, το πρώτο για εκτέλεση μιας εφαρμογής και το δεύτερο για εκτέλεση πολλαπλών εφαρμογών.

- **Εκτέλεση μιας εφαρμογής**

  Αρχικά, βρισκόμαστε μέσα σε ένα φάκελο στον οποίο όπως φαίνεται και από την Εικόνα 2.7.1 έχει δύο αρχεία:

  - ✓ **extrae.xml** Το XML αρχείο που θα χρησιμοποιηθεί από το EXTRAE.
  - ✓ **user-comms.c** Ο πηγαίος κώδικας της εφαρμογής μας.

```
xeirwn@teras:~/zacharias/EXTRAE_INSTALL/share/example/PTHREAD$ ls -l
-rw-r--r-- 1 xeirwn xeirwn 1873 Aug 13 07:58 extrae.xml
-rw-r--r-- 1 xeirwn xeirwn 3519 Jul 25 12:37 user-comms.c
```

Εικόνα 2.7.1

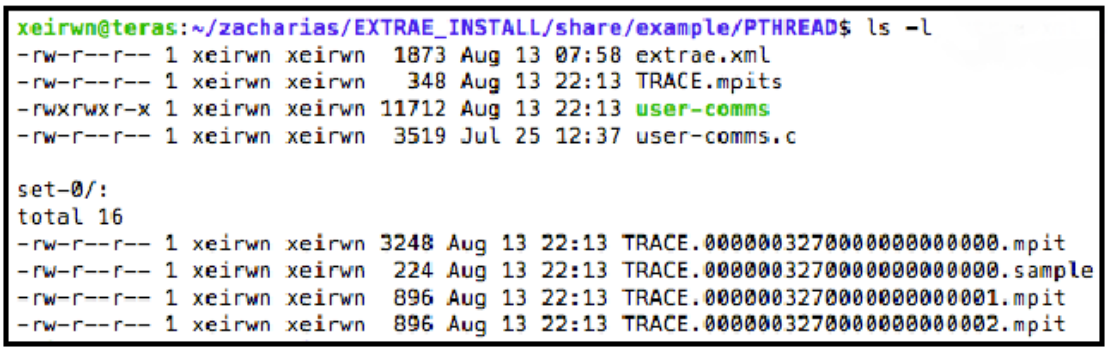
Στη συνέχεια, εκτελούμε τα πιο κάτω βήματα για δημιουργία του εκτελέσιμου αρχείου user-comms:

- ✓ export PAPI_HOME=/home/xeirwn/zacharias/PAPI_INSTALL
- ✓ export EXTRAE_HOME=/home/xeirwn/zacharias/EXTRAE_INSTALL
- ✓ gcc user-comms.c -O -g -I$EXTRAE_HOME/include -L$EXTRAE_HOME/lib -lpttrace -L$PAPI_HOME/lib -lpapi -o user-comms

Και έπειτα, το εκτελούμε με τα πιο κάτω βήματα:

- ✓ export LD_LIBRARY_PATH=$EXTRAE_HOME/lib:$PAPI_HOME/lib
- ✓ export EXTRAE_CONFIG_FILE=extrae.xml
- ✓ ./user-comms

Ως αποτέλεσμα της εκτέλεσης έχουμε την δημιουργία ενός TRACE.mpits αρχείου και ενός φακέλου set-0, ο οποίος περιέχει τα *.mpit αρχεία (Εικόνα 2.7.2).

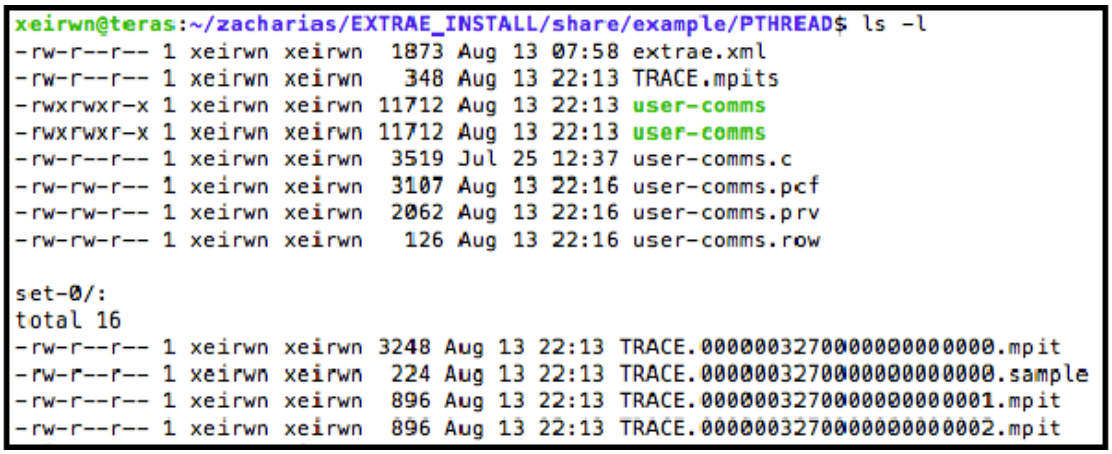


Εικόνα 2.7.2

Τέλος, εκτελούμε την διαδικασία συγχώνευσης των *.mpit αρχείων εκτελώντας την παρακάτω εντολή:

- ✓ $EXTRAE_HOME/bin/mpi2prv -f TRACE.mpits -e user-comms -o user-comms.prv

Αποτέλεσμα του πιο πάνω είναι η δημιουργία των τριών αρχείων (user-comms.prv, user-comms.pcf και user-comms.row) που χρησιμοποιεί το εργαλείο PARAVER (Εικόνα 2.7.3).



Εικόνα 2.7.3

- **Εκτέλεση πολλαπλών εφαρμογών**

  Όταν πρόκειται να εκτελέσουμε πολλαπλές εφαρμογές με το εργαλείο EXTRAE, ο καλύτερος για εμένα τρόπος είναι ο ακόλουθος:

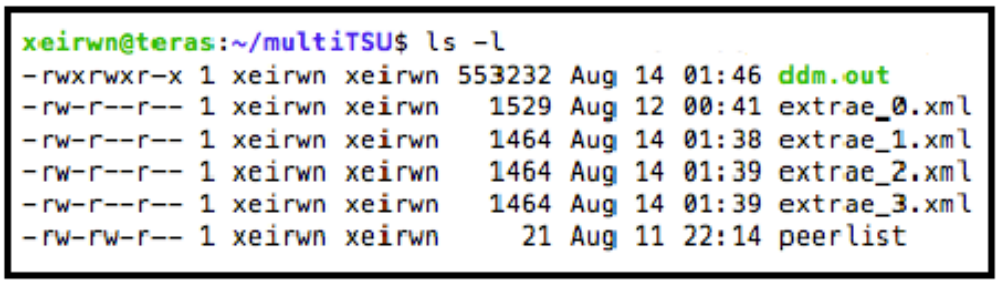
  Σε κάθε Makefile που έχει να κάνει με εφαρμογή που θα τρέξουμε με το εργαλείο EXTRAE, πάμε και ορίζουμε μέσα σε αυτό τις μεταβλητές PAPI_HOME και EXTRAE_HOME, έτσι ώστε να δείχνουν στα μονοπάτια στα οποία έχουν εγκατασταθεί το PAPI και το EXTRAE αντίστοιχα. Ακόμη, προσθέτουμε τις βιβλιοθήκες του PAPI και του EXTRAE στην εντολή που μεταγλωττίζει τον πηγαίο κώδικα της εφαρμογής μας. Ένα παράδειγμα τέτοιου Makefile μπορείτε να δείτε στην Εικόνα 2.7.4.

```
PAPI_HOME = /home/xeirwn/zacharias/PAPI_INSTALL
EXTRAE_HOME = /home/xeirwn/zacharias/EXTRAE_INSTALL
LIBS = /lib64/librt.so.1 -L$(EXTRAE_HOME)/lib -L$(PAPI_HOME)/lib -lpttrace -lpapi
ZFLAGS = -O -g -I$(EXTRAE_HOME)/include

OBJS = mmult.o

all: main.o
        @echo
        @echo g++ Project
        g++ $(CPPFLAGS) main.o $(ZFLAGS) $(LIBS) $(OBJS) $(MAINFLAGS) -o ../../ddm.out
        @echo
```

Εικόνα 2.7.4

Ακολούθως, δημιουργούμε τα εκτελέσιμα αρχεία των εφαρμογών που επιθυμούμε να αναλύσουμε. Εκτός αυτού, δημιουργούμε τόσα XML αρχεία όσα και οι εφαρμογές που επιθυμούμε να αναλύσουμε. Οι επιλογές μας σε κάθε XML αρχείο μπορούν να ρυθμιστούν ανάλογα με το τι θέλουμε να εξετάσουμε στην εφαρμογή για την οποία προορίζεται το αρχείο αυτό. Ωστόσο, τα μονοπάτια στα temporal και final πεδία του τμήματος “Storage Management” πρέπει να είναι τα ίδια σε όλα τα XML αρχεία. Αυτό που πρέπει οπωσδήποτε να διαφέρει σε κάθε XML αρχείο είναι το πεδίο trace-prefix του τμήματος “Storage Management”. Για παράδειγμα για το XML αρχείο της πρώτης εφαρμογής να είναι TRACE_0, της δεύτερης TRACE_1 και της N TRACE_N. Στην Εικόνα 2.7.5 μπορείτε να δείτε ένα τέτοιο παράδειγμα. Στο παράδειγμα αυτό θα τρέξω τέσσερις εφαρμογές, γι’ αυτό και δημιουργώ τέσσερα XML αρχεία. Ο λόγος που μόνο ένα εκτελέσιμο υπάρχει

(ddm.out), είναι γιατί και οι τέσσερις εκτελέσεις μου αφορούν εκτέλεση της ίδιας εφαρμογής. Ωστόσο, σε κάθε εκτέλεσή της θα χρησιμοποιεί διαφορετικό XML αρχείο.



Εικόνα 2.7.5

Έπειτα, εκτελούμε μέσα σε μια μεγάλη εντολή τα ακόλουθα:

- ✓ export LD_LIBRARY_PATH=$EXTRAE_HOME/lib:$PAPI_HOME/lib ;
- ✓ export EXTRAE_CONFIG_FILE=extrae_XXX.xml ;
- ✓ ./exec_XXX &

Η πρώτη εντολή γράφεται μόνο μια φορά στην αρχή και δεν επαναλαμβάνεται, ενώ η δεύτερη και η τρίτη επαναλαμβάνονται τόσες φορές όσες και οι εφαρμογές που θέλω να εκτελέσω.

Για παράδειγμα, αν θεωρήσουμε πως έχουμε τρία εκτελέσιμα διαφορετικών εφαρμογών (exec_1, exec_2 και exec_3) και τα αντίστοιχα XML αρχεία για κάθε ένα από αυτά (extrae_1.xml, extrae_2.xml και extrae_3.xml), η εντολή θα είναι:

export LD_LIBRARY_PATH=$EXTRAE_HOME/lib:$PAPI_HOME/lib; export EXTRAE_CONFIG_FILE=extrae_1.xml; ./exec_1 & export EXTRAE_CONFIG_FILE=extrae_2.xml; ./exec_2 & export EXTRAE_CONFIG_FILE=extrae_3.xml; ./exec_3 &

Στην Εικόνα 2.7.6 φαίνεται ένα τέτοιο παράδειγμα, που αποτελεί συνέχεια του παραδείγματος στην Εικόνα 2.7.5.

```
export LD_LIBRARY_PATH=/home/xeirwn/zacharias/EXTRAE_INSTALL/lib:/home/xeirwn/zacharias/PAPI_INSTALL/lib; exp
ort EXTRAE_CONFIG_FILE=extrae_0.xml; ./ddm.out --PEER_LIST_FILE=peerlist --NROW=256 --BSIZE=4 --SIM_ITER_NUM=
5 --KERNEL_NUM=1 & export EXTRAE_CONFIG_FILE=extrae_1.xml; ./ddm.out --PEER_LIST_FILE=peerlist --NROW=256 --B
SIZE=4 --SIM_ITER_NUM=5 --KERNEL_NUM=1 & export EXTRAE_CONFIG_FILE=extrae_2.xml; ./ddm.out --PEER_LIST_FILE=p
eerlist --NROW=256 --BSIZE=4 --SIM_ITER_NUM=5 --KERNEL_NUM=1 & export EXTRAE_CONFIG_FILE=extrae_3.xml; ./ddm.
out --PEER_LIST_FILE=peerlist --NROW=256 --BSIZE=4 --SIM_ITER_NUM=5 --KERNEL_NUM=1 &
```

Εικόνα 2.7.6

Αποτέλεσμα αυτής της εκτέλεσης είναι η δημιουργία τεσσάρων *.mpits αρχείων (όσα και οι εφαρμογές που εκτελέστηκαν) και ενός φακέλου set-0, ο οποίος περιέχει τα *.mpit αρχεία (Εικόνα 2.7.7).

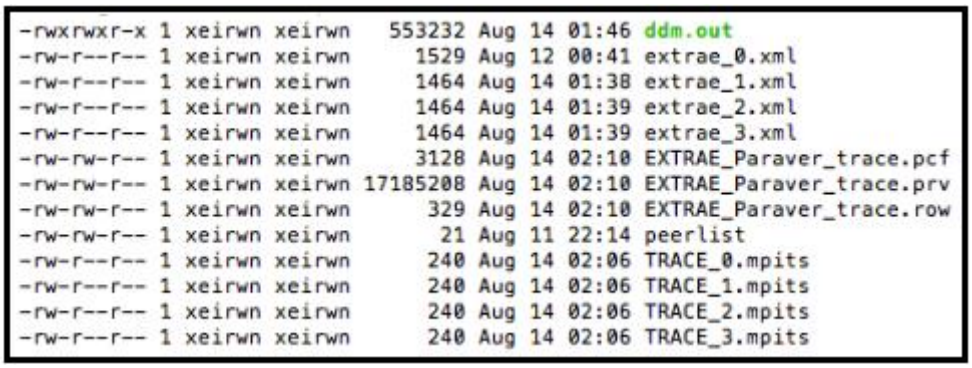
```
xeirwn@teras:~/multiTSU$ ls -l extrae_* peerlist ddm.out TRACE_* set-0/; echo
-rwxrwxr-x 1 xeirwn xeirwn 553232 Aug 14 01:46 ddm.out
-rw-r--r-- 1 xeirwn xeirwn   1529 Aug 12 00:41 extrae_0.xml
-rw-r--r-- 1 xeirwn xeirwn   1464 Aug 14 01:38 extrae_1.xml
-rw-r--r-- 1 xeirwn xeirwn   1464 Aug 14 01:39 extrae_2.xml
-rw-r--r-- 1 xeirwn xeirwn   1464 Aug 14 01:39 extrae_3.xml
-rw-rw-r-- 1 xeirwn xeirwn     21 Aug 11 22:14 peerlist
-rw-r--r-- 1 xeirwn xeirwn    240 Aug 14 02:06 TRACE_0.mpits
-rw-r--r-- 1 xeirwn xeirwn    240 Aug 14 02:06 TRACE_1.mpits
-rw-r--r-- 1 xeirwn xeirwn    240 Aug 14 02:06 TRACE_2.mpits
-rw-r--r-- 1 xeirwn xeirwn    240 Aug 14 02:06 TRACE_3.mpits

set-0/:
total 58304
-rw-r--r-- 1 xeirwn xeirwn     2800 Aug 14 02:06 TRACE_0.0000003995000000000000.mpit
-rw-r--r-- 1 xeirwn xeirwn      336 Aug 14 02:06 TRACE_0.0000003995000000000000.sample
-rw-r--r-- 1 xeirwn xeirwn 14910112 Aug 14 02:06 TRACE_0.0000003995000000000001.mpit
-rw-r--r-- 1 xeirwn xeirwn     2912 Aug 14 02:06 TRACE_0.0000003995000000000002.mpit
-rw-r--r-- 1 xeirwn xeirwn     2800 Aug 14 02:06 TRACE_1.0000003996000000000000.mpit
-rw-r--r-- 1 xeirwn xeirwn      224 Aug 14 02:06 TRACE_1.0000003996000000000000.sample
-rw-r--r-- 1 xeirwn xeirwn 14910112 Aug 14 02:06 TRACE_1.0000003996000000000001.mpit
-rw-r--r-- 1 xeirwn xeirwn     1904 Aug 14 02:06 TRACE_1.0000003996000000000002.mpit
-rw-r--r-- 1 xeirwn xeirwn     2800 Aug 14 02:06 TRACE_2.0000003997000000000000.mpit
-rw-r--r-- 1 xeirwn xeirwn      224 Aug 14 02:06 TRACE_2.0000003997000000000000.sample
-rw-r--r-- 1 xeirwn xeirwn 14910112 Aug 14 02:06 TRACE_2.0000003997000000000001.mpit
-rw-r--r-- 1 xeirwn xeirwn     1904 Aug 14 02:06 TRACE_2.0000003997000000000002.mpit
-rw-r--r-- 1 xeirwn xeirwn     2800 Aug 14 02:06 TRACE_3.0000003998000000000000.mpit
-rw-r--r-- 1 xeirwn xeirwn      224 Aug 14 02:06 TRACE_3.0000003998000000000000.sample
-rw-r--r-- 1 xeirwn xeirwn 14910112 Aug 14 02:06 TRACE_3.0000003998000000000001.mpit
-rw-r--r-- 1 xeirwn xeirwn     1904 Aug 14 02:06 TRACE_3.0000003998000000000002.mpit
```

Εικόνα 2.7.7

Τέλος, εκτελούμε την διαδικασία συγχώνευσης των *.mpit αρχείων εκτελώντας την παρακάτω εντολή:

- ✓ $EXTRAE_HOME/bin/mpi2prv -f TRACE_0.mpits -- -f TRACE_1.mpits -- -f TRACE_2.mpits -- -f TRACE_3.mpits

Αποτέλεσμα είναι η δημιουργία τριών αρχείων (EXTRAE_Paraver_trace.prv, EXTRAE_Paraver_trace.pcf και EXTRAE_Paraver_trace.row) τα οποία χρησιμοποιεί το εργαλείο PARAVER (Εικόνα 2.7.8).


-rwxrwxr-x 1 xeirwn xeirwn 553232 Aug 14 01:46 ddm.out
-rw-r--r-- 1 xeirwn xeirwn 1529 Aug 12 00:41 extrae_0.xml
-rw-r--r-- 1 xeirwn xeirwn 1464 Aug 14 01:38 extrae_1.xml
-rw-r--r-- 1 xeirwn xeirwn 1464 Aug 14 01:39 extrae_2.xml
-rw-r--r-- 1 xeirwn xeirwn 1464 Aug 14 01:39 extrae_3.xml
-rw-rw-r-- 1 xeirwn xeirwn 3128 Aug 14 02:10 EXTRAE_Paraver_trace.pcf
-rw-rw-r-- 1 xeirwn xeirwn 17185208 Aug 14 02:10 EXTRAE_Paraver_trace.prv
-rw-rw-r-- 1 xeirwn xeirwn 329 Aug 14 02:10 EXTRAE_Paraver_trace.row
-rw-rw-r-- 1 xeirwn xeirwn 21 Aug 11 22:14 peerlist
-rw-r--r-- 1 xeirwn xeirwn 240 Aug 14 02:06 TRACE_0.mpits
-rw-r--r-- 1 xeirwn xeirwn 240 Aug 14 02:06 TRACE_1.mpits
-rw-r--r-- 1 xeirwn xeirwn 240 Aug 14 02:06 TRACE_2.mpits
-rw-r--r-- 1 xeirwn xeirwn 240 Aug 14 02:06 TRACE_3.mpits

Εικόνα 2.7.8

Κεφάλαιο 3

Performance API

3.1 Εισαγωγή στο PAPI

Το PAPI είναι ακρώνυμο που αφορά το Performance Application Programming Interface. Το Project PAPI αναπτύσσεται στο Innovative Computing Laboratory του τμήματος Πληροφορικής, του Πανεπιστημίου του Tennessee. Σκοπός του project αυτού είναι η σχεδίαση, τυποποίηση και ανάπτυξη ενός φορητού και αποτελεσματικού API, που θα προσφέρει πρόσβαση στις hardware μετρικές απόδοσης των σύγχρονων μικροεπεξεργαστών.

Οι hardware μετρικές απόδοσης υπάρχουν σε όλους τους σύγχρονους επεξεργαστές, όπως για παράδειγμα στους:

- Intel Pentium
- IA-64
- AMD Athlon
- IBM POWER series

Αυτές οι μετρικές εμφανίζονται ως μικρά σύνολα καταχωρητών, οι οποίοι καταμετρούν συμβάντα συγκεκριμένων σημάτων που αφορούν τη λειτουργία του επεξεργαστή. Η παρακολούθηση αυτών των συμβάντων βοηθά στη συσχέτιση της δομής του πηγαίου κώδικά μας με την αποτελεσματικότητα της αντιστοίχισης (mapping) αυτού του κώδικα

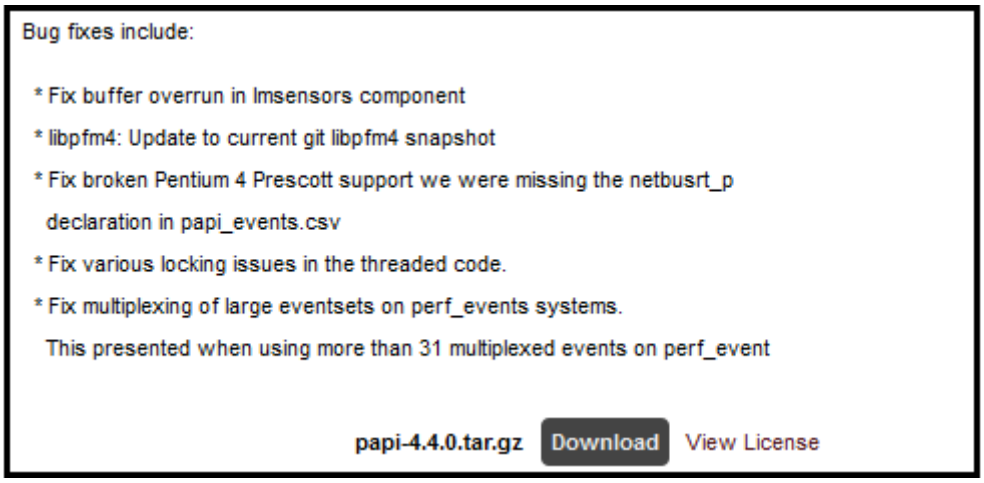
στην αρχιτεκτονική της μηχανής μας. Αυτή η συσχέτιση παρέχει στον προγραμματιστή χρήσιμες πληροφορίες για τμήματα του κώδικά του τα οποία μπορούν να βελτιωθούν.

3.2 Λήψη και εγκατάσταση του PAPI

Οποιοσδήποτε μπορεί να κατεβάσει τον πηγαίο κώδικα του PAPI από την ιστοσελίδα του Innovative Computing Laboratory (ICL) και πιο συγκεκριμένα από τον ακόλουθο σύνδεσμο:

http://icl.cs.utk.edu/papi/software/index.html

Αρκεί να μεταφερθεί στον πιο πάνω σύνδεσμο και να πατήσει το κουμπί Download που βρίσκεται δίπλα από την τελευταία έκδοση του PAPI (Εικόνα 3.2.1).



Εικόνα 3.2.1

Έχοντας κατεβάσει και αποσυμπιέσει τον πηγαίο κώδικα του PAPI, μετακινούμαστε μέσα στο φάκελο **src/** και εκτελούμε την εντολή:

./configure --prefix=DIR

Αυτή η εντολή δηλώνει πως θέλουμε να εγκαταστήσουμε το PAPI στο μονοπάτι DIR. Αφού εκτελέσουμε την εντολή **./configure**, ακολούθως εκτελούμε πρώτα την εντολή **make** και έπειτα την εντολή **make install**. Ολοκληρώνοντας τη διαδικασία αυτή, το PAPI εγκαθίσταται στο μονοπάτι που καθορίσαμε με την επιλογή **--prefix=DIR**.

3.3 Λίστα με τις διαθέσιμες μετρικές του PAPI

Στην Εικόνα 3.3.1 και στην Εικόνα 3.3.2, μπορείτε να δείτε τα προκαθορισμένα ονόματα για τις μετρικές απόδοσης που προσφέρει το PAPI, καθώς και μια σύντομη περιγραφή για κάθε μια από αυτές. Στο εργαλείο EXTRAE, ο χρήστης μπορεί να χρησιμοποιήσει όποιες μετρικές απόδοσης επιθυμεί, αρκεί να τις γράψει με το κανονικό όνομά τους (PRESET NAME) στο τμήμα “Performance Counters” του XML αρχείου.

PRESET NAME	DESCRIPTION	PRESET NAME	DESCRIPTION
PAPI_L1_DCM	Level 1 data cache misses	PAPI_FUL_ICY	Cycles with Maximum Instruction Issue
PAPI_L1_ICM	Level 1 instruction cache misses		
PAPI_L2_DCM	Level 2 data cache misses		
PAPI_L2_ICM	Level 2 instruction cache misses	PAPI_STL_CCY	Cycles with No Instruction Completion
PAPI_L3_DCM	Level 3 data cache misses	PAPI_FUL_CCY	Cycles with Maximum Instruction Completion
PAPI_L3_ICM	Level 3 instruction cache misses	PAPI_HW_INT	Hardware interrupts
PAPI_L1_TCM	Level 1 total cache misses	PAPI_BR_UCN	Unconditional branch instructions executed
PAPI_L2_TCM	Level 2 total cache misses	PAPI_BR_CN	Conditional branch instructions executed
PAPI_L3_TCM	Level 3 total cache misses	PAPI_BR_TKN	Conditional branch instructions taken
PAPI_CA_SNP	Requests for a Snoop	PAPI_BR_NTK	Conditional branch instructions not taken
PAPI_CA_SHR	Requests for access to shared cache line (SMP)	PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_CA_CLN	Requests for access to clean cache line (SMP)	PAPI_BR_PRC	Conditional branch instructions correctly predicted
PAPI_CA_INV	Cache Line Invalidation (SMP)	PAPI_FMA_INS	FMA instructions completed
PAPI_CA_ITV	Cache Line Intervention (SMP)	PAPI_TOT_IIS	Total instructions issued
PAPI_L3_LDM	Level 3 load misses	PAPI_TOT_INS	Total instructions executed
PAPI_L3_STM	Level 3 store misses	PAPI_INT_INS	Integer instructions executed
PAPI_BRU_IDL	Cycles branch units are idle	PAPI_FP_INS	Floating point instructions executed
PAPI_FXU_IDL	Cycles integer units are idle	PAPI_LD_INS	Load instructions executed
PAPI_FPU_IDL	Cycles floating point units are idle	PAPI_SR_INS	Store instructions executed
PAPI_LSU_IDL	Cycles load/store units are idle	PAPI_BR_INS	Total branch instructions executed
PAPI_TLB_DM	Data translation lookaside buffer misses	PAPI_VEC_INS	Vector/SIMD instructions executed
PAPI_TLB_IM	Instruction translation lookaside buffer misses	PAPI_FLOPS	Floating Point Instructions executed per second
PAPI_TLB_TL	Total translation lookaside buffer misses	PAPI_RES_STL	Cycles processor is stalled on resource
PAPI_L1_LDM	Level 1 load misses	PAPI_FP_STAL	Cycles any FP units are stalled
PAPI_L1_STM	Level 1 store misses	PAPI_TOT_CYC	Total cycles
PAPI_L2_LDM	Level 2 load misses	PAPI_IPS	Instructions executed per second
PAPI_L2_STM	Level 2 store misses	PAPI_LST_INS	Total load/store instructions executed
PAPI_BTAC_M	Branch target address cache (BTAC) misses	PAPI_SYC_INS	Synchronization instructions executed
PAPI_PRF_DM	Pre-fetch data instruction caused a miss	PAPI_L1_DCH	L1 data cache hits
PAPI_L3_DCH	Level 3 Data Cache Hit	PAPI_L2_DCH	L2 data cache hits
PAPI_TLB_SD	Translation lookaside buffer shootdowns (SMP)	PAPI_L1_DCA	L1 data cache accesses
PAPI_CSR_FAL	Failed store conditional instructions	PAPI_L2_DCA	L2 data cache accesses
PAPI_CSR_SUC	Successful store conditional instructions	PAPI_L3_DCA	L3 data cache accesses
PAPI_CSR_TOT	Total store conditional instructions	PAPI_L1_DCR	L1 data cache reads
PAPI_MEM_SCY	Cycles Stalled Waiting for Memory Access	PAPI_L2_DCR	L2 data cache reads
PAPI_MEM_RCY	Cycles Stalled Waiting for Memory Read	PAPI_L3_DCR	L3 data cache reads
PAPI_MEM_WCY	Cycles Stalled Waiting for Memory Write	PAPI_L1_DCW	L1 data cache writes
PAPI_STL_ICY	Cycles with No Instruction Issue		

Εικόνα 3.3.1

PRESET NAME	DESCRIPTION
PAPI_L2_DCW	L2 data cache writes
PAPI_L3_DCW	L3 data cache writes
PAPI_L1_ICH	L1 instruction cache hits
PAPI_L2_ICH	L2 instruction cache hits
PAPI_L3_ICH	L3 instruction cache hits
PAPI_L1_ICA	L1 instruction cache accesses
PAPI_L2_ICA	L2 instruction cache accesses
PAPI_L3_ICA	L3 instruction cache accesses
PAPI_L1_ICR	L1 instruction cache reads
PAPI_L2_ICR	L2 instruction cache reads
PAPI_L3_ICR	L3 instruction cache reads
PAPI_L1_ICW	L1 instruction cache writes
PAPI_L2_ICW	L2 instruction cache writes
PAPI_L3_ICW	L3 instruction cache writes
PAPI_L1_TCH	L1 total cache hits
PAPI_L2_TCH	L2 total cache hits
PAPI_L1_TCA	L1 total cache accesses
PAPI_L2_TCA	L2 total cache accesses
PAPI_L3_TCA	L3 total cache accesses
PAPI_L1_TCR	L1 total cache reads
PAPI_L2_TCR	L2 total cache reads
PAPI_L3_TCR	L3 total cache reads
PAPI_L1_TCW	L1 total cache writes
PAPI_L2_TCW	L2 total cache writes
PAPI_L3_TCW	L3 total cache writes
PAPI_FML_INS	Floating Multiply instructions
PAPI_FAD_INS	Floating Add instructions
PAPI_FDV_INS	Floating Divide instructions
PAPI_FSQ_INS	Floating Square Root instructions
PAPI_FNV_INS	Floating Inverse instructions

Εικόνα 3.3.2

3.4 Διαθέσιμες μετρικές PAPI στη μηχανή μας

Όπως διαβάσατε στο μέρος **3.3** αυτού του κεφαλαίου, το PAPI προσφέρει ένα μεγάλο φάσμα από μετρικές απόδοσης. Ωστόσο, πρέπει να γνωρίζετε πως συνήθως η μηχανή στην οποία εγκαθιστάτε το PAPI, περιορίζει την διαθεσιμότητα κάποιων μετρικών απόδοσης. Για να δείτε ανά πάσα στιγμή ποιες μετρικές απόδοσης είναι διαθέσιμες στην μηχανή σας, απλά μετακινηθείτε μέσα στο φάκελο **bin/**, ο οποίος βρίσκεται μέσα στο φάκελο που εγκαταστήσατε το PAPI. Ενώ βρισκόσαστε μέσα στον φάκελο **bin/**, εκτελείτε το binary αρχείο **papi_avail** το οποίο σας επιστρέφει στην οθόνη όλες τις μετρικές απόδοσης που προσφέρει το PAPI. Όσες από αυτές έχουν στην στήλη **Avail** την τιμή "Yes", σημαίνει πως είναι διαθέσιμες στην μηχανή σας (Εικόνα 3.4.1).

```
xeirwn@teras:~/PAPI_INSTALL/bin$ ./papi_avail
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version             : 4.4.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel(R) Xeon(R) CPU           E5320  @ 1.86GHz (15)
CPU Revision             : 11.000000
CPUID Info               : Family: 6  Model: 15  Stepping: 11
CPU Megahertz            : 1596.000000
CPU Clock Megahertz      : 1596
Hdw Threads per core     : 1
Cores per Socket         : 4
NUMA Nodes               : 1
CPUs per Node            : 8
Total CPUs               : 8
Number Hardware Counters : 5
Max Multiplex Counters   : 32
--------------------------------------------------------------------------------

    Name        Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  Yes   No   Level 2 instruction cache misses
PAPI_L3_DCM  0x80000004  No    No   Level 3 data cache misses
PAPI_L3_ICM  0x80000005  No    No   Level 3 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   No   Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes   No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  No    No   Level 3 cache misses
PAPI_CA_SNP  0x80000009  No    No   Requests for a snoop
PAPI_CA_SHR  0x8000000a  Yes   No   Requests for exclusive access to shared cache line
PAPI_CA_CLN  0x8000000b  Yes   No   Requests for exclusive access to clean cache line
PAPI_CA_INV  0x8000000c  No    No   Requests for cache line invalidation
PAPI_CA_ITV  0x8000000d  Yes   No   Requests for cache line intervention
PAPI_L3_LDM  0x8000000e  No    No   Level 3 load misses
PAPI_L3_STM  0x8000000f  No    No   Level 3 store misses
PAPI_BRU_IDL 0x80000010  No    No   Cycles branch units are idle
PAPI_FXU_IDL 0x80000011  No    No   Cycles integer units are idle
PAPI_FPU_IDL 0x80000012  No    No   Cycles floating point units are idle
PAPI_LSU_IDL 0x80000013  No    No   Cycles load/store units are idle
PAPI_TLB_DM  0x80000014  Yes   No   Data translation lookaside buffer misses
PAPI_TLB_IM  0x80000015  Yes   No   Instruction translation lookaside buffer misses
PAPI_TLB_TL  0x80000016  No    No   Total translation lookaside buffer misses
PAPI_L1_LDM  0x80000017  Yes   No   Level 1 load misses
PAPI_L1_STM  0x80000018  Yes   No   Level 1 store misses
PAPI_L2_LDM  0x80000019  Yes   Yes  Level 2 load misses
```

Εικόνα 3.4.1

3.5 Περιορισμοί PAPI στο εργαλείο EXTRAE

Όπως ανέφερα ήδη στο **Κεφάλαιο** 2, το εργαλείο EXTRAE μπορεί να μαζέψει μέχρι οκτώ (8) μετρικές απόδοσης την ίδια χρονική στιγμή. Αυτό δεν σημαίνει απαραίτητα οκτώ (8) PAPI μετρικές απόδοσης, καθώς κάποιες μετρικές απόδοσης του PAPI για να προκύψουν, χρησιμοποιούν και άλλες μετρικές. Άρα, στην ουσία κάποιες φορές δεν χρησιμοποιείται μια μετρική όπως φαίνεται, αλλά περισσότερες. Μπορείτε και πάλι να δείτε ποιες PAPI μετρικές εξαρτώνται από άλλες μετρικές, εκτελώντας πάλι το binary αρχείο **papi_avail**. Όσες από αυτές έχουν στην στήλη **Deriv** την τιμή “Yes”, σημαίνει πως για να προκύψουν χρησιμοποιούν και άλλες μετρικές (Εικόνα 3.4.1).

Κεφάλαιο 4

Περιγραφή *.prv, *.pcf και *.row αρχείων

4.1 Εισαγωγή

Όπως ανάφερα στο **Κεφάλαιο 2** μετά την εκτέλεση μιας εφαρμογής με το εργαλείο EXTRAE, ακολουθεί η διαδικασία συγχώνευσης κατά την οποία ο merger (mpi2prv) χρησιμοποιεί τα ενδιάμεσα trace αρχεία (*.mpit) για να δημιουργήσει τρία διαφορετικά αρχεία με το ίδιο όνομα αλλά διαφορετικό extension.

Τα τρία αυτά αρχεία είναι τα ακόλουθα:

- ***.prv αρχείο** Το τελικό trace αρχείο που θα χρησιμοποιήσει το PARAVER.
- ***.pcf αρχείο** Το configuration αρχείο που θα χρησιμοποιήσει το PARAVER
- ***.row αρχείο** Καθορίζει τις ετικέτες των αντικειμένων που περιέχει το *.prv.

Στο κεφάλαιο αυτό θα γίνει μια περιγραφή της δομής των τριών αυτών αρχείων.

4.2 Δομή *.prv αρχείου

Κύριος στόχος του *.prv αρχείου είναι να καθορίσει την δομή-εμφάνιση των αντικειμένων (νήματα, διεργασίες και εφαρμογές) που προκύπτουν από την ανάλυση με το εργαλείο EXTRAE.

Ένα αρχείο *.prv περιέχει τρία βασικά είδη εγγραφών (records) τα οποία είναι:

- **Καταστάσεις (states)**
  Η κατάσταση (running, idle κτλ) ενός αντικειμένου (νήματος, διεργασίας και εφαρμογής) για μια χρονική περίοδο.
- **Γεγονότα (events)**
  Τα διάφορα events που υπάρχουν στον κώδικά μας, είτε αυτά έχουν καθοριστεί από τον χρήστη με την συνάρτηση Extrae_event του EXTRAE API, είτε αυτά είναι προκαθορισμένα (π.χ. pthread_create, pthread_join). Ένα οποιοδήποτε event περιγράφεται από την χρονική στιγμή στην οποία συμβαίνει και δύο τιμές, type και value.
- **Επικοινωνίες (communications)**
  Οι φυσικές και λογικές επικοινωνίες που διεξάγονται μεταξύ διαφορετικών αντικειμένων (νημάτων, διεργασιών και εφαρμογών) και η χρονική διάρκεια αυτών.

Η δομή που ακολουθεί ένα αρχείο *.prv είναι η ακόλουθη:

- **Επικεφαλίδα (header)**
  Βρίσκεται στην πρώτη γραμμή κάθε *.prv αρχείου και έχει την ακόλουθη δομή:
  - ✓ **#Paraver (dd/mm/yyyy at hh:mm):** Καθορίζει την ημερομηνία και ώρα κατά την οποία δημιουργήθηκε το trace αρχείο.
  - ✓ **ftime:** Συνολική διάρκεια του trace αρχείου.
  - ✓ **nNodes(nCpus1[,nCpus2,…nCpusN]):** Καθορίζει τον αριθμό των κόμβων (nNodes) και τον αριθμό των επεξεργαστών (nCpus) σε κάθε κόμβο.
  - ✓ **nAppl:** Καθορίζει τον αριθμό των εφαρμογών μέσα στο trace αρχείο.

- ✓ **applicationList:** Καθορίζει την δομή κάθε μιας εφαρμογής από τις nAppl που υπάρχουν μέσα στο trace αρχείο. Κάθε εφαρμογή έχει το δικό της aplicationList χωρισμένο με τον χαρακτήρα «**:**». Η μορφή ενός aplicationList είναι η ακόλουθη:

  nTasks(nTreads1:node,…,nThreadsN:node)

  Όπου nTasks είναι ο αριθμός των διεργασιών μέσα στην εφαρμογή, nThreads ο αριθμός των νημάτων για κάθε μια διεργασία από τις nTasks και node ο κόμβος στον οποίο έτρεξε η κάθε μια διεργασία.

Στην Εικόνα 4.2.1 και στην Εικόνα 4.2.2 μπορείτε να δείτε δύο διαφορετικά παραδείγματα επικεφαλίδων σε *.prv αρχεία.

```
#Paraver (18/07/2012 at 04:03):17238_ns:1(2):2:1(1:1),01(1:1),0
1:1:1:1:1:0:172388584828291:2
1:2:2:1:1:0:172388777548291:2
2:1:1:1:1:172388584817291:40000001:1
```

Εικόνα 4.2.1

Από την Εικόνα 4.2.1 μπορούμε να συμπεράνουμε πως το *.prv αρχείο δημιουργήθηκε στις 18/7/2012, η ώρα 04:03. Έχει διάρκεια 17238 nanoseconds και εκτελέστηκε σε ένα (1) κόμβο, ο οποίος έχει δύο (2) επεξεργαστές. Επίσης, αποτελείται από δύο (2) εφαρμογές εκ των οποίων η πρώτη έχει μια (1) διεργασία η οποία τρέχει ένα (1) νήμα και εκτελείται στον κόμβο (1) και η δεύτερη έχει πάλι μια (1) διεργασία η οποία τρέχει ένα (1) νήμα και εκτελείται στον κόμβο (1).

```
#Paraver (06/08/2012 at 07:01):61972_ns:1(8):1:1(8:1),0
1:1:1:1:1:0:1129583:18
2:1:1:1:1:0:40000001:1
1:2:1:1:2:0:26413623:2
```

Εικόνα 4.2.2

Από την Εικόνα 4.2.2 μπορούμε να συμπεράνουμε πως το *.prv αρχείο δημιουργήθηκε στις 6/8/2012, η ώρα 07:01. Έχει διάρκεια 61972 nanoseconds και εκτελέστηκε σε ένα (1) κόμβο, ο οποίος έχει οκτώ (8) επεξεργαστές.

Επίσης, αποτελείται από μία (1) εφαρμογή, η οποία έχει μια (1) διεργασία η οποία τρέχει οκτώ (8) νήματα και εκτελείται στον κόμβο (1).

- **Σώμα (body)**

  Οτιδήποτε υπάρχει μετά την επικεφαλίδα. Στο σώμα του *.prv αρχείου περιέχονται οι διάφορες εγγραφές (records) που περιέγραψα προηγουμένως, ταξινομημένες σε χρονολογική σειρά (begin_time για states, time για events και lsend για communications). Σε περίπτωση που τρία διαφορετικά είδη εγγραφών συμβαίνουν στην ίδια χρονική στιγμή, τότε πρώτα μπαίνει η επικοινωνία (communication), έπειτα το γεγονός (event) και τέλος η κατάσταση (state). Σε κάθε εγγραφή υπάρχουν πληροφορίες για την πλειάδα εφαρμογή, διεργασία και νήμα, στην οποία έτρεξε η εγγραφή όπως επίσης και για τον αριθμό επεξεργαστή στον οποίο έτρεξε.

  Η μορφή που έχει κάθε εγγραφή (record) είναι η ακόλουθη:

  - ✓ **Κατάσταση (state)**

    1:cpu_id:app_id:task_id:thread_id:begin_time:end_time:state

    Οι εγγραφές που αφορούν καταστάσεις (states) ξεκινούν πάντα με την τιμή 1. Ακολουθεί η πλειάδα επεξεργαστής, εφαρμογή, διεργασία και νήμα την οποία αφορά η κατάσταση αυτή, η διάρκεια της κατάστασης (χρόνος έναρξης – χρόνος λήξης) και η τιμή της κατάστασης. Κάθε τιμή κατάστασης αντιστοιχείται σε μια ετικέτα και ένα χρώμα, όπως θα δούμε αργότερα στην περιγραφή του *.pcf αρχείου. Για παράδειγμα στην Εικόνα 4.2.2 η δεύτερη γραμμή αφορά μια εγγραφή κατάστασης (18) η οποία συμβαίνει στον επεξεργαστή (1), εφαρμογή (1), διεργασία (1), νήμα (1) και έχει διάρκεια από την χρονική στιγμή μηδέν (0) μέχρι την χρονική στιγμή 1129583.

  - ✓ **Γεγονός (event)**

    2:cpu_id:app_id:task_id:thread_id:time:type1:value1:…:typeN:valueN

    Οι εγγραφές που αφορούν γεγονότα (events) ξεκινούν πάντα με την τιμή 2. Ακολουθεί η πλειάδα επεξεργαστής, εφαρμογή, διεργασία και νήμα την οποία αφορά το γεγονός αυτό, η χρονική στιγμή διεξαγωγής του γεγονότος και μια λίστα από πλειάδες type και value (ανάλογα με τον αριθμό των γεγονότων που συνέβησαν κατά τη χρονική στιγμή time). Κάθε πλειάδα type και value αντιστοιχείται σε μια ετικέτα και

ένα χρώμα, όπως θα δούμε αργότερα στην περιγραφή του *.pcf αρχείου. Για παράδειγμα στην Εικόνα 4.2.2 η τρίτη γραμμή αφορά μια εγγραφή γεγονότος με type 40000001 και value 1 η οποία συμβαίνει στον επεξεργαστή (1), εφαρμογή (1), διεργασία (1), νήμα (1) κατά την χρονική στιγμή μηδέν (0).

- ✓ **Επικοινωνία (communication)**

  3:object_send:lsend:psend:object_recv:lrecv:precv:size:tag

  Οι εγγραφές που αφορούν επικοινωνίες (communications) ξεκινούν πάντα με την τιμή 3. Ακολουθεί το πεδίο object_send το οποίο καθορίζει ποιο αντικείμενο είναι ο αποστολέας. Πιο συγκεκριμένα το object_send αποτελείται από την πλειάδα επεξεργαστής, εφαρμογή, διεργασία και νήμα. Ακολούθως έχουμε το lsend και psend, όπου το πρώτο καθορίζει την χρονική στιγμή που ο χρήστης επιθυμεί να στείλει ένα μήνυμα και το δεύτερο την χρονική στιγμή που όντως στέλλεται το μήνυμα. Ακολουθεί το πεδίο object_recv το οποίο καθορίζει ποιο αντικείμενο είναι ο παραλήπτης και το οποίο αποτελείται από την πλειάδα επεξεργαστής, εφαρμογή, διεργασία και νήμα. Ακολούθως έχουμε το lrecv και precv, όπου το πρώτο καθορίζει την χρονική στιγμή που ο χρήστης επιθυμεί να παραλάβει ένα μήνυμα και το δεύτερο την χρονική στιγμή που όντως παραλαμβάνεται το μήνυμα. Τέλος, έχουμε τα πεδία size και tag, όπου το πρώτο καθορίζει το μέγεθος του μηνύματος σε bytes και το δεύτερο, τον αναγνωριστικό τύπο του μηνύματος. Για παράδειγμα στην Εικόνα 4.2.3 η γραμμή αφορά μια εγγραφή επικοινωνίας με αποστολέα τον επεξεργαστή (2), εφαρμογή (2), διεργασία (1), νήμα (1) και παραλήπτη τον επεξεργαστή (1), εφαρμογή (1), διεργασία (1), νήμα (1). Η αποστολή (lsend και psend) γίνεται την χρονική στιγμή 172388782584291 και η παραλαβή (lrecv και precv) την χρονική στιγμή 172388782624291. Το μέγεθος του μηνύματος είναι 512 bytes και το tag μηδέν (0).

```
3:2:2:1:1:172388782584291:172388782584291:
1:1:1:1:172388782624291:172388782624291:512:0
```

Εικόνα 4.2.3

4.3 Δομή *.pcf αρχείου

Κύριος στόχος του *.pcf αρχείου είναι να καθορίσει τις ετικέτες (labels) και τα χρώματα για τις διάφορες καταστάσεις (states) και τα διάφορα γεγονότα (events) που περιέχει το *.prv αρχείο. Παράλληλα, μέσω του *.pcf αρχείου καθορίζονται και κάποιες επιλογές που αφορούν τον τρόπο εμφάνισης του *.prv αρχείου στο εργαλείο PARAVER.

Πιο κάτω θα αναφέρω και θα περιγράψω τα βασικότερα τμήματα ενός *.pcf αρχείου. Για περισσότερα τμήματα και πληροφορίες για αυτά μπορείτε να συμβουλευτείτε το "TRACEFILE DESCRIPTION" PDF το οποίο βρίσκεται στον σύνδεσμο:
http://www.bsc.es/media/1370.pdf

- **Τμήμα "DEFAULT_OPTIONS"**

```
DEFAULT_OPTIONS

LEVEL               THREAD
UNITS               NANOSEC
NUM_OF_STATE_COLORS 1000
```

Εικόνα 4.3.1

Στο τμήμα αυτό καθορίζονται κάποιες γενικές επιλογές τις οποίες θα εφαρμόσει το εργαλείο PARAVER όταν θα φορτώσει τα trace αρχεία. Το τμήμα αυτό, όπως φαίνεται και από την Εικόνα 4.3.1 μας παρέχει τις ακόλουθες επιλογές:

- ✓ **LEVEL** Καθορίζει σε τι επίπεδο (system, node, cpu, thread, task, appl, workload) ανάλυσης θα ανοιχτεί το trace αρχείο όταν φορτωθεί από το εργαλείο PARAVER.
- ✓ **UNITS** Καθορίζουμε τη μονάδα χρόνου (nanosec, microsec, milisec, sec, hour) που θα χρησιμοποιήσει το εργαλείο PARAVER.
- ✓ **NUM_OF_STATE_COLORS** Καθορίζουμε τον αριθμό των καταστάσεων (states) για τις οποίες ο χρήστης μπορεί να καθορίσει ετικέτα (label) στο τμήμα "STATES" και χρώμα στο τμήμα "STATES_COLOR".

- **Τμήμα “STATES”**

```
STATES
0    Idle
1    Running
2    Not created
3    Waiting a message
4    Blocking Send
5    Synchronization
6    Test/Probe
7    Scheduling and Fork/Join
8    Wait/WaitAll
9    Blocked
10    Immediate Send
11    Immediate Receive
12    I/O
13    Group Communication
14    Tracing Disabled
15    Others
16    Send Receive
17    Memory transfer
```

Εικόνα 4.3.2

Στο τμήμα αυτό, όπως φαίνεται και από την Εικόνα 4.3.2, καθορίζονται οι ετικέτες (labels) που θα χρησιμοποιηθούν για κάθε κατάσταση (state). Κάθε γραμμή έχει πρώτα την τιμή της κατάστασης (state) και δίπλα την ετικέτα (label) για αυτή την κατάσταση.

- **Τμήμα “STATES_COLOR”**

Στο τμήμα αυτό, όπως φαίνεται και από την Εικόνα 4.3.3, καθορίζονται τα χρώματα που θα χρησιμοποιηθούν για κάθε κατάσταση (state). Κάθε γραμμή έχει πρώτα την τιμή της κατάστασης (state) και δίπλα το χρώμα για αυτή την κατάσταση σε κλίμακα RGB.

```
STATES_COLOR
0    {117,195,255}
1    {0,0,255}
2    {255,255,255}
3    {255,0,0}
4    {255,0,174}
5    {179,0,0}
6    {0,255,0}
7    {255,255,0}
8    {235,0,0}
9    {0,162,0}
10    {255,0,255}
11    {100,100,177}
12    {172,174,41}
13    {255,144,26}
14    {2,255,177}
15    {192,224,0}
16    {66,66,66}
17    {255,0,96}
```

Εικόνα 4.3.3

- **Τμήμα "EVENTS"**

```
EVENT_TYPE
1    51100001    Send Size in PACX Global OP
1    51100002    Recv Size in PACX Global OP
1    51100003    Root in PACX Global OP
1    51100004    Communicator in PACX Global O

EVENT_TYPE
1000    1    COMMUNICATION_1
VALUES
1      SEND
0      RECEIVE
```

Εικόνα 4.3.4

Στο τμήμα αυτό, όπως φαίνεται και από την Εικόνα 4.3.4, καθορίζονται οι ετικέτες (labels) και τα χρώματα που θα χρησιμοποιηθούν για κάθε γεγονός (event). Ο χρήστης μπορεί να καθορίσει τα πιο πάνω, είτε ανοίγοντας το *.pcf αρχείο και γράφοντας μέσα σε αυτό τις ετικέτες (labels) και τα χρώματα που επιθυμεί για κάθε γεγονός (event), είτε χρησιμοποιώντας την μεταβλητή περιβάλλοντος **EXTRAE_LABELS** κατά την διαδικασία συγχώνευσης των *.mpit αρχείων (**Κεφάλαιο 2 – Μέρος 2.6**). Ο τρόπος σύνταξης είναι ο ακόλουθος:

EVENT_TYPE

gradient_color type label

gradient_color type label

……

[VALUES

value label

value label

……]

Όπου “gradient_color” είναι το χρώμα που θα χρησιμοποιηθεί για να ζωγραφίσει το εργαλείο PARAVER το event (καθορίζεται στο τμήμα GRADIENT_COLOR του *.pcf αρχείου, το οποίο έχει την ίδια μορφή με το τμήμα STATES_COLOR που περιγράφηκε προηγουμένως), “type” είναι ο τύπος του event και “label” η ετικέτα που θα εμφανίσει το εργαλείο PARAVER όταν ο χρήστης κάνει κλικ πάνω στο event. Προαιρετικά ο χρήστης μπορεί να δηλώσει κάτω από τον ορισμό ενός EVENT_TYPE, τις ετικέτες για τα διάφορα values που μπορεί να πάρουν τα types που δηλώθηκαν μέσα στο EVENT_TYPE. Αυτές τις ετικέτες θα εμφανίζει το εργαλείο PARAVER όταν ο χρήστης κάνει κλικ πάνω στο event.

4.4 Δομή *.row αρχείου

Κύριος στόχος του *.row αρχείου είναι να καθορίσει για κάθε επίπεδο (SYSTEM, NODE, CPU, WORKLOAD, APPL, TASK και THREAD)

- τον αριθμό των αντικειμένων στο επίπεδο αυτό και
- τις ετικέτες (labels) κάθε αντικειμένου στο επίπεδο αυτό

Η μορφή ενός *.row αρχείου είναι η ακόλουθη:

LEVEL "name of level" SIZE "number of objects in this level"
"object1_label"
"object2_label"
:
:
"objectN_label"

Ανάλογα με το επίπεδο ανάλυσης (SYSTEM, NODE, CPU, WORKLOAD, APPL, TASK και THREAD) που θα επιλεχθεί στο PARAVER όταν φορτωθούν τα trace αρχεία, το εργαλείο θα εμφανίζει τις ετικέτες που αντιστοιχούν στα αντικείμενα αυτού του επιπέδου.

```
LEVEL CPU SIZE 2
1.b103ws1
2.b103ws1

LEVEL NODE SIZE 1
b103ws1

LEVEL THREAD SIZE 2
THREAD 1.1.1
THREAD 2.1.1
```

Εικόνα 4.4.1

Στην Εικόνα 4.4.1 βλέπετε ένα παράδειγμα ενός *.row αρχείου στο οποίο σε επίπεδο CPU υπάρχουν δύο αντικείμενα με όνομα "b103ws1", σε επίπεδο NODE υπάρχει ένα αντικείμενο με όνομα "b103ws1" και σε επίπεδο THREAD υπάρχουν δύο αντικείμενα με ονόματα "THREAD 1.1.1" και "THREAD 2.1.1".

Κεφάλαιο 5

Αλγόριθμος Parser

5.1 Περιγραφή προβλήματος

Όπως ήδη θα έχετε διαβάσει στο **Κεφάλαιο 2**, το εργαλείο EXTRAE δεν παρέχει την δυνατότητα για καταγραφή επικοινωνιών που συμβαίνουν μεταξύ διαφορετικών εφαρμογών. Αυτό μπορείτε να το διαπιστώσετε και από e-mail που έχω από το Barcelona Supercomputing Center (BSC) και το οποίο αποδεικνύει το πιο πάνω. Το e-mail αυτό μπορείτε να το διαβάσετε στο **ΠΑΡΑΡΤΗΜΑ Α** (σελ. Α-3).

Η αδυναμία καταγραφής επικοινωνιών μεταξύ διαφορετικών εφαρμογών αποτελεί ένα μεγάλο πρόβλημα στο σκοπό της Διπλωματικής Εργασίας μου, καθώς στο DDM μοντέλο έχουμε επικοινωνίες μεταξύ διαφορετικών εφαρμογών τις οποίες είναι σημαντικό να καταγράψουμε, έτσι ώστε να μπορούμε αργότερα να τις αναλύσουμε με το εργαλείο PARAVER.

5.2 Λύση στο πρόβλημα

Ως μέτρο αντιμετώπισης του προβλήματος αυτού σκέφτηκα όπως χρησιμοποιώ την συνάρτηση **Extrae_event** όπου υπάρχει επικοινωνία μεταξύ διαφορετικών εφαρμογών. Πιο συγκεκριμένα, θα τοποθετώ μια συνάρτηση **Extrae_event** ακριβώς πριν το σημείο που γίνεται η αποστολή και μια ακριβώς μετά το σημείο που γίνεται η παραλαβή. Οι δύο παράμετροι της συνάρτησης θα καθορίζουν τα ακόλουθα:

- **type** Το είδος της επικοινωνίας. Για το σκοπό αυτό αποφάσισα πως δέκα ακέραιες τιμές θα είναι κρατημένες για σκοπούς επικοινωνίας και δεν πρέπει να χρησιμοποιούνται στην παράμετρο type για οποιοδήποτε άλλο λόγο (Πίνακας 5.2.1).
- **value** Μέσω του value θα παίρνω δύο πληροφορίες:
  - ✓ Το μέγεθος της επικοινωνίας σε bytes.
  - ✓ Ποιά εφαρμογή είναι ο παραλήπτης σε περίπτωση που το event είναι send, ή ποιά εφαρμογή είναι ο αποστολέας σε περίπτωση που το event είναι receive.

  Ο τρόπος με τον οποίο θα το πετύχω αυτό, βασίζεται στην διαίρεση της τιμής value με μια προκαθορισμένη τιμή η οποία ως ακέραιο πηλίκο θα δίνει το μέγεθος της επικοινωνίας και ως υπόλοιπο θα δίνει την εφαρμογή παραλήπτη/αποστολέα. Περισσότερες πληροφορίες για τον τρόπο αυτό μπορείτε να διαβάσετε στα **Μέρη 5.3** και **5.4** αυτού του Κεφαλαίου.

TYPE NUM	TYPE STRING	MATCHES WITH	TYPE STRING	TYPE NUM
2	SEND MESSAGE	↔	RECEIVE MESSAGE	3
4	SEND COMMAND	↔	RECEIVE COMMAND	5
6	SEND TYPE_3	↔	RECEIVE TYPE_3	7
8	SEND TYPE_4	↔	RECEIVE TYPE_4	9
10	SEND TYPE_5	↔	RECEIVE TYPE_5	11

Πίνακας 5.2.1

5.3 Ευθύνες προγραμματιστή

Για να πετύχει ο στόχος μας για αναπαράσταση των επικοινωνιών μεταξύ διαφορετικών εφαρμογών, πρέπει και ο προγραμματιστής που θα τρέξει τις εφαρμογές του με το εργαλείο EXTRAE να ακολουθήσει λίγα και απλά βήματα.

Πρώτο βήμα, είναι να τοποθετεί από μια συνάρτηση **Extrae_event** ακριβώς πριν από κάθε σημείο που γίνεται αποστολή (send). Οι δύο παράμετροι που θα περνά ο προγραμματιστής στην συνάρτηση **Extrae_event** θα προκύπτουν με την ακόλουθη λογική:

- **type** Μια τιμή από το σύνολο 2, 4, 6, 8, 10 οι οποίες όπως θα δείτε και στον Πίνακα 5.2.1, αντιστοιχούν σε τύπους αποστολής. Ο προγραμματιστής θα επιλέγει την τιμή που εκφράζει καλύτερα το τι στέλλει. Αν στέλλει μήνυμα, επιλέγει την τιμή 2 (SEND MESSAGE), ενώ αν στέλλει εντολή, επιλέγει την τιμή 4 (SEND COMMAND).
- **value** Το value θα προκύπτει με τον εξής τρόπο:

  value = recv_appl + (size_of_comm*mult) όπου:
  - ✓ **recv_appl** Ο αριθμός της εφαρμογής παραλήπτη.
  - ✓ **size_of_comm** Το μέγεθος της επικοινωνίας σε bytes.
  - ✓ **mult** Προκύπτει με την εξής λογική:
    - Αν θα τρέξω μονοψήφιο αριθμό εφαρμογών (2-9) τότε **mult=10**.
    - Αν θα τρέξω διψήφιο αριθμό εφαρμογών (10-99) τότε **mult=100**.
    - Αν θα τρέξω τριψήφιο αριθμό εφαρμογών (100-999) τότε **mult=1000**.

Δεύτερο και τελευταίο βήμα, είναι να τοποθετεί από μια συνάρτηση **Extrae_event** ακριβώς μετά από κάθε σημείο που γίνεται παραλαβή (receive). Οι δύο παράμετροι που θα περνά ο προγραμματιστής στην συνάρτηση **Extrae_event** θα προκύπτουν με την ακόλουθη λογική:

- **type** Μια τιμή από το σύνολο 3, 5, 7, 9, 11 οι οποίες όπως θα δείτε και στον Πίνακα 5.2.1, αντιστοιχούν σε τύπους παραλαβής. Ο προγραμματιστής θα επιλέγει την τιμή που εκφράζει καλύτερα το τι παραλαμβάνει. Αν παραλαμβάνει μήνυμα, επιλέγει την τιμή 3 (RECEIVE MESSAGE), ενώ αν παραλαμβάνει εντολή, επιλέγει την τιμή 5 (RECEIVE COMMAND).
- **value** Το value όταν πρόκειται για παραλαβή ισούται απλά με τον αριθμό της εφαρμογής αποστολέα.

Στην Εικόνα 5.3.1 μπορείτε να δείτε ένα παράδειγμα χρήσης της συνάρτησης **Extrae_event** ακριβώς πριν το σημείο που γίνεται η αποστολή.

```
Extrae_event( 2 , (node+1) + 1024*10 );
if (sendMessage(node,DDM_DATA,NULL,0,(void*)local_adc

        printf("sendMessage Error!\n");
        return 0;
}
```

Εικόνα 5.3.1

Στο παράδειγμα αυτό, η επικοινωνία έχει να κάνει με αποστολή ενός μηνύματος (type 2) μεγέθους 1024 bytes, στην εφαρμογή (node+1). Ο προγραμματιστής αυτής της εφαρμογής θα τρέξει μονοψήφιο (2-9) αριθμό εφαρμογών, καθώς το mult του ισούται με δέκα (10).

Στην Εικόνα 5.3.2 μπορείτε να δείτε ένα παράδειγμα χρήσης της συνάρτησης **Extrae_event** ακριβώς μετά το σημείο που γίνεται η παραλαβή.

```
node = receiveMessage(&Header,(void*)command_buffer,&data_v_addr);
if(node >= 0)
        Extrae_event(3,node+1);
```

Εικόνα 5.3.2

Στο παράδειγμα αυτό, η επικοινωνία έχει να κάνει με παραλαβή ενός μηνύματος (type 3) από την εφαρμογή (node+1).

5.4 Αλγόριθμος Paraver Parser

Στο πρόγραμμά μου κάνω χρήση δύο δομών δεδομένων. Η μια δομή αφορά μια λίστα από εγγραφές γεγονότων (events) και η άλλη, ένα δέντρο που περιέχει όλα τα είδη εγγραφών (events, states, communications). Πιο κάτω ακολουθεί μια περιγραφή των πεδίων κάθε δομής.

Στην Εικόνα 5.4.1 μπορείτε να δείτε τη δομή λίστας που χρησιμοποιώ και τα πεδία των κόμβων που περιέχει. Στην λίστα αυτή μπαίνουν όσες εγγραφές γεγονότων (events) διαβάστηκαν από το *.prv αρχείο και **αφορούν επικοινωνίες**. Έχουν δηλαδή για **type τιμές από 2 ως 11** (Πίνακας 5.2.1).

```
typedef struct{
    unsigned long type, value;
}TV_COUPLES;

typedef struct e_v_t{
    TV_COUPLES type_value;
    struct e_v_t *next;
}EVENT_VALUE_TYPE;

typedef struct o_n_e{
    unsigned long cpu, time, events_num, comm_event_pos, app, task, thread;
    int has_comm_event;
    EVENT_VALUE_TYPE *head, *tail;
    struct o_n_e *next;
}EVENT_NODE;

typedef struct{
    unsigned long size;
    EVENT_NODE *head, *tail;
}EVENT_LIST;
```

Εικόνα 5.4.1

- **cpu** Επεξεργαστής στον οποίο συνέβηκε το γεγονός (event).
- **app** Εφαρμογή στην οποία συνέβηκε το γεγονός (event).
- **task** Διεργασία στην οποία συνέβηκε το γεγονός (event).
- **thread** Νήμα στο οποίο συνέβηκε το γεγονός (event).
- **time** Χρονική στιγμή στην οποία συνέβηκε το γεγονός (event).
- **events_num** Αριθμός ζευγαριών type και value που περιέχει, καθώς στην ίδια χρονική στιγμή μπορεί να συμβούν πολλά γεγονότα.

- **has_comm_event** Χρησιμοποιείται σαν boolean μεταβλητή και έχει πάντα τιμή TRUE (1).
- **head & tail** Λίστα στην οποία φυλάγονται όλα τα ζευγάρια type και value των "events_num" γεγονότων που συνέβηκαν κατά την χρονική στιγμή "time".
- **comm_event_pos** Ο κόμβος της λίστας με τα ζευγάρια type και value, ο οποίος περιέχει το γεγονός επικοινωνίας (comm_event), δηλαδή έχει type 2 ως 11.

Στην Εικόνα 5.4.2 μπορείτε να δείτε τη δομή δέντρου που χρησιμοποιώ και τα πεδία των κόμβων που περιέχει. Στο δέντρο αυτό μπαίνουν όλες οι εγγραφές (events, states, communications) που διαβάστηκαν από το *.prv αρχείο. Από την στιγμή που σε κάθε κόμβο μπορεί να μπουν πληροφορίες για διαφορετικά είδη εγγραφών (events, states, communications), σκέφτηκα όπως κάποια πεδία της δομής δέχονται διαφορετικά δεδομένα, ανάλογα και του είδους εγγραφής (events, states, communications) που εισάγεται. Έτσι δεν θα σπαταλούσα μνήμη για δημιουργία όλων των πιθανών πεδίων που περιέχουν τα τρία είδη εγγραφών (events, states, communications), όπως αυτά περιγράφηκαν στο **Κεφάλαιο 4 - Περιγραφή prv, pcf και row αρχείων**.

```
typedef struct t_n{
    int record_type;
    unsigned long cpu, app, task, thread, event_OR_start_OR_send_time;
    unsigned long finish_OR_receive_time, events_num_OR_state_OR_buffer_size;
    unsigned long cpu2, app2, task2, thread2, tag;
    EVENT_VALUE_TYPE *head, *tail;
    struct t_n *left, *right;
}TREE_NODE;
```

Εικόνα 5.4.2

Κάθε πεδίο της δεντρικής δομής περιέχει τα πιο κάτω στοιχεία:

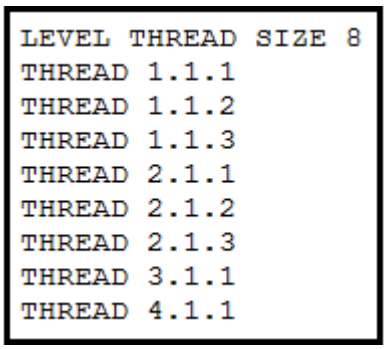
- **record_type** Το είδος της εγγραφής που διαβάστηκε.
  - ✓ **1** για κατάσταση (state).
  - ✓ **2** για γεγονός (event).
  - ✓ **3** για επικοινωνία (communication).
- **cpu** Επεξεργαστής στον οποίο συνέβηκε η εγγραφή (record).
- **app** Εφαρμογή στην οποία συνέβηκε η εγγραφή (record).

- **task** Διεργασία στην οποία συνέβηκε η εγγραφή (record).
- **thread** Νήμα στο οποίο συνέβηκε η εγγραφή (record).
- **event_OR_start_OR_send_time** Ανάλογα με το είδος της εγγραφής που διαβάστηκε:
    - ✓ **κατάσταση (state)** Χρονική στιγμή που ξεκινά η κατάσταση (state).
    - ✓ **γεγονός (event)** Χρονική στιγμή που συμβαίνει το γεγονός (event).
    - ✓ **επικοινωνία (communication)** Χρονική στιγμή αποστολής.
- **finish_OR_receive_time** Ανάλογα με το είδος της εγγραφής που διαβάστηκε:
    - ✓ **κατάσταση (state)** Χρονική στιγμή που σταματά η κατάσταση (state).
    - ✓ **γεγονός (event)** Το πεδίο αυτό είναι μηδέν (0).
    - ✓ **επικοινωνία (communication)** Χρονική στιγμή παραλαβής.
- **events_num_OR_state_OR_buffer_size** Ανάλογα με το είδος της εγγραφής που διαβάστηκε:
    - ✓ **κατάσταση (state)** Τιμή της κατάστασης (state).
    - ✓ **γεγονός (event)** Αριθμός γεγονότων (events) σε αυτή την εγγραφή.
    - ✓ **επικοινωνία (communication)** Μέγεθος επικοινωνίας σε bytes.
- **head & tail** Αφορά μόνο τις εγγραφές γεγονότων (events). Είναι μια λίστα η οποία περιέχει όλα τα ζευγάρια type και value των γεγονότων που περιέχονται στην εγγραφή.

Τα πέντε παρακάτω πεδία αφορούν μόνο τις εγγραφές επικοινωνίας (communications). Για τα άλλα δύο είδη εγγραφών (events, states) τα πεδία αυτά παίρνουν τιμή ίση με μηδέν (0).

- **cpu2** Επεξεργαστής στον οποίο γίνεται η παραλαβή.
- **app2** Εφαρμογή στην οποία γίνεται η παραλαβή.
- **task2** Διεργασία στην οποία γίνεται η παραλαβή.
- **thread2** Νήμα στο οποίο γίνεται η παραλαβή.
- **tag** Αναγνωριστικός τύπος μηνύματος.

Το πρόγραμμα ξεκινά την εκτέλεση του ανοίγοντας πρώτα το *.row αρχείο. Μέσα στο αρχείο αυτό, θα ψάξει να βρει το τμήμα με το χαμηλότερο επίπεδο, δηλαδή το επίπεδο THREAD. Στη συνέχεια διαβάζει ένα προς ένα τα αντικείμενα που περιέχονται κάτω από το επίπεδο αυτό και συγκρίνοντας τον αριθμό που υπάρχει πριν την πρώτη τελεία (αυτός ο αριθμός αφορά την εφαρμογή στην οποία τρέχει το νήμα), υπολογίζει τον αριθμό των διαφορετικών εφαρμογών.



Εικόνα 5.4.3

Για παράδειγμα στο *.row αρχείο της Εικόνας 5.4.3 ο αριθμός των διαφορετικών εφαρμογών που θα υπολογίσει το πρόγραμμα είναι τέσσερις (4). Και αυτό γιατί έχουμε τρία αντικείμενα με εφαρμογή (1), τρία αντικείμενα με εφαρμογή (2), ένα αντικείμενο με εφαρμογή (3) και ένα αντικείμενο με εφαρμογή (4).

Αφού υπολογίσει τον αριθμό των διαφορετικών εφαρμογών, στην συνέχεια το πρόγραμμα υπολογίζει την τιμή της μεταβλητής με την οποία θα διαιρεί τις τιμές value όσων event αφορούν επικοινωνίες (type 2 ως 11). Είναι από αυτή την διαίρεση που το ακέραιο πηλίκο θα δώσει το μέγεθος επικοινωνίας σε bytes και το υπόλοιπο, τον αριθμό της εφαρμογής αποστολέα/παραλήπτη. Η τιμή αυτής της μεταβλητής προκύπτει με την εξής λογική:

- Αν έχω μονοψήφιο αριθμό διαφορετικών εφαρμογών (2-9) τότε **μεταβλητή=10.**
- Αν έχω διψήφιο αριθμό διαφορετικών εφαρμογών (10-99) τότε **μεταβλητή=100.**
- Αν έχω τριψήφιο αριθμό διαφορετικών εφαρμογών (100-999) τότε **μεταβλητή=1000.**

Αφού υπολογίσει την τιμή αυτής της μεταβλητής, κλείνει το *.row αρχείο και ανοίγει το *.prv αρχείο. Από το αρχείο αυτό, αρχικά διαβάζει την πρώτη γραμμή η οποία δίνει χρήσιμες πληροφορίες για την εκτέλεση με το εργαλείο EXTRAE και την τοποθετεί σε μια συμβολοσειρά. Στην συνέχεια διαβάζει μία προς μία τις υπόλοιπες γραμμές, οι οποίες αφορούν εγγραφές state, event και communication, και τις τοποθετεί μέσα στην δεντρική δομή με χρονολογική σειρά.

- Στο αριστερό παιδί αν ο χρόνος είναι μικρότερος από τον κόμβο σύγκρισης.
- Στο δεξί παιδί αν ο χρόνος είναι μεγαλύτερος από τον κόμβο σύγκρισης.
- Αν ο χρόνος είναι ο ίδιος με τον κόμβο σύγκρισης:
    - ✓ Στο αριστερό παιδί αν ο τύπος εγγραφής (record type) είναι μεγαλύτερος ή ίσος του τύπου εγγραφής του κόμβου σύγκρισης.
    - ✓ Στο δεξί παιδί αν ο τύπος εγγραφής (record type) είναι μικρότερος του τύπου εγγραφής του κόμβου σύγκρισης.

Εδώ είναι σημαντικό να αναφέρω πως όσες εγγραφές γεγονότων (event records) αφορούν επικοινωνίες (δηλαδή έχουν type 2 ως 11), την στιγμή που μπαίνουν στη δεντρική δομή ως value τους δεν εισάγεται αυτούσιο το value τους, αλλά η τιμή που προκύπτει από το value%μεταβλητή και η οποία δίνει τον αριθμό εφαρμογής αποστολέα/παραλήπτη. Αυτό γίνεται για να μπορεί το PARAVER όταν θα ανοίξει τα αρχεία και κάνει η κλικ ο χρήστης στο event, να δείχνει ως label την εφαρμογή στην οποία στέλλεται ή από την οποία παραλαμβάνεται η επικοινωνία (Εικόνα 5.4.4). Εκτός από το να εισαχθούν στην δεντρική δομή, όσες εγγραφές γεγονότων (event records) αφορούν επικοινωνίες εισάγονται και στην δομή λίστας που ανέφερα πιο πάνω. Στην λίστα το value τους εισάγεται αυτούσιο.

Αφού τελειώσει με το διάβασμα των γραμμών του *.prv αρχείου, παίρνει την λίστα και ξεκινά από τον πρώτο κόμβο της. Για κάθε κόμβο της λίστας, προσπαθεί να βρει τον αμέσως επόμενο κόμβο ο οποίος αποτελεί το ζευγάρι του. Αυτό επιτυγχάνεται με σύγκριση των τιμών app, type και value των δύο κόμβων. Δύο κόμβοι θεωρούνται ζευγάρι αν ισχύουν και τα τρία πιο κάτω:

- Το app του πρώτου κόμβου ισούται με το value%μεταβλητή του δεύτερου κόμβου. Με απλά λόγια, αν η εφαρμογή που στέλλει/παραλαμβάνει (app) την επικοινωνία είναι η ίδια με την εφαρμογή αποστολέα/παραλήπτη (value%μεταβλητή).

- Το app του δεύτερου κόμβου ισούται με το value%μεταβλητή του πρώτου κόμβου. Η ίδια σκεπτική με το προηγούμενο βήμα.
- Αν το type του πρώτου κόμβου ταιριάζει με το type του δεύτερου (Πίνακας 5.2.1). Για παράδειγμα το type του πρώτου κόμβου να είναι 2 (SEND MESSAGE) και το type του δεύτερου κόμβου να είναι 3 (RECEIVE MESSAGE).

Όταν βρει ένα ζευγάρι ως πρώτο βήμα κάνει τα πεδία "has_comm_event" των κόμβων τους να ισούνται με FALSE, έτσι ώστε να μην επανελεγχθούν και ως δεύτερο βήμα εισάγει το ζευγάρι αυτό σαν εγγραφή επικοινωνίας (communication record) στην δεντρική δομή. Ως "events_num_OR_state_OR_buffer_size" πεδίο του δεντρικού κόμβου εισάγεται το μέγεθος της επικοινωνίας σε bytes, το οποίο προκύπτει από την πράξη value/μεταβλητή (όπου value, η τιμή value του κόμβου της λίστας που έχει type αποστολής). Η εισαγωγή στην δεντρική δομή γίνεται με βάση τη χρονική στιγμή "time" του κόμβου που έχει type αποστολής (δηλαδή 2, 4, 6, 8 ή 10).

Αφού εξετάσει όλους τους κόμβους της λίστας, το πρόγραμμα ανοίγει το *.pcf αρχείο και προσθέτει σε αυτό δέκα EVENT_TYPES τα οποία αφορούν τους τύπους επικοινωνίας που έχω προκαθορίσει (2-11). Κάτω από κάθε EVENT_TYPE έχει τόσα VALUES όσα και οι διαφορετικές εφαρμογές. Απλά στην ετικέτα των VALUES μπαίνει "to" αν το EVENT_TYPE έχει να κάνει με αποστολή, ή "from" αν το EVENT_TYPE έχει να κάνει με παραλαβή (Εικόνα 5.4.4).

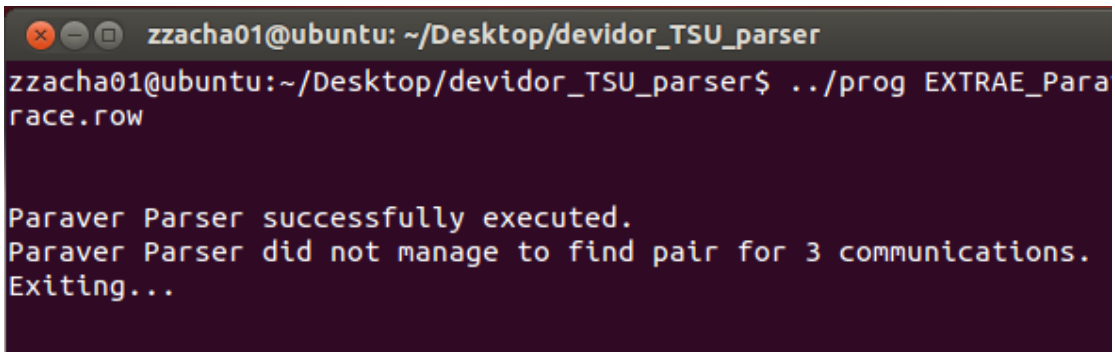
```
EVENT_TYPE
1000    2    SEND MESSAGE
VALUES
1       to APP_1
2       to APP_2


EVENT_TYPE
1000    3    RECEIVE MESSAGE
VALUES
1       from APP_1
2       from APP_2
```

Εικόνα 5.4.4

Η πιο πάνω διαδικασία γίνεται για να μπορεί ο χρήστης να δει τι είδος επικοινωνίας (SEND MESSAGE, REVEIVE MESSAGE, κτλ) συμβαίνει, όταν ανοίξει το PARAVER και κάνει κλικ πάνω στο event αυτό.

Ακολούθως, το πρόγραμμα ανοίγει και πάλι το *.prv αρχείο, αλλά αυτή την φορά για να το κάνει overwrite. Πρώτα τυπώνει μέσα σε αυτό την συμβολοσειρά η οποία περιείχε την πρώτη γραμμή του *.prv αρχείου και η οποία δίνει χρήσιμες πληροφορίες για την εκτέλεση με το εργαλείο EXTRAE. Έπειτα, τυπώνει μέσα σε αυτό την δεντρική δομή ξεκινώντας από το αριστερότερο φύλλο και καταλήγοντας προς το δεξιότερο, εξυπηρετώντας έτσι την απαίτηση του *.prv αρχείου για ταξινόμηση των εγγραφών του σε χρονολογική σειρά.

Τέλος, τυπώνει ένα μήνυμα στην οθόνη το οποίο αναφέρει στον χρήστη πως το πρόγραμμα εκτελέστηκε επιτυχώς. Παράλληλα, αναφέρει τον αριθμό των επικοινωνιών για τις οποίες δεν κατάφερε να βρει ζευγάρι (Εικόνα 5.4.5).



Εικόνα 5.4.5

Στην περίπτωση που υπάρχουν επικοινωνίες για τις οποίες δεν βρέθηκαν τα ζευγάρια τους, τότε αυτό σημαίνει πως ο προγραμματιστής ξέχασε να τοποθετήσει σε κάποια σημεία του κώδικά του τις συναρτήσεις **Extrae_event**. Στο παράδειγμα της Εικόνας 5.4.5 το πρόβλημα προκύπτει από το γεγονός ότι δεν τοποθετήθηκε η συνάρτηση **Extrae_event** στο σημείο που γινόταν broadcast. Αυτό είχε σαν αποτέλεσμα να καταγραφούν δέκα (10) λήψεις μηνυμάτων, ενώ είχαν καταγραφεί μόνο επτά (7) αποστολές μηνυμάτων. Η διαφορά (-3) μεταξύ των δύο τιμών οφείλεται στο γεγονός ότι έτρεχα τέσσερις (4) εφαρμογές, άρα είχα τρία (3) επιπλέον send στο σημείο του broadcast, τα οποία δεν κατέγραψα.
Αν στο μήνυμα που τυπώνει στην οθόνη το πρόγραμμα υπάρχει έστω και μια επικοινωνία για την οποία δεν βρέθηκε ζευγάρι, τότε πολύ πιθανόν το τελικό *.prv αρχείο που προκύπτει να περιέχει λάθος πληροφορία. Για αυτό, είναι καλύτερα ο προγραμματιστής να διορθώσει το λάθος του και ακολούθως να εκτελέσει ξανά το πρόγραμμα.

Στην Εικόνα 5.4.6 μπορείτε να δείτε ένα παράδειγμα μετατροπής των συναρτήσεων **Extrae_event** σε εγγραφή επικοινωνίας.

```
3:6:2:1:3:21284:21284:9:3:1:3:21285:21285:1024:0
2:6:2:1:3:21284:2:3
2:9:3:1:3:21285:3:2
```

Εικόνα 5.4.6

Στο παράδειγμα αυτό το event της δεύτερης γραμμής λέει πως η εφαρμογή (2) στέλλει μήνυμα (type 2) στην εφαρμογή (3) την χρονική στιγμή 21284. Το event της τρίτης γραμμής λέει πως η εφαρμογή (3) παραλαμβάνει μήνυμα (type 3) από την εφαρμογή (2) την χρονική στιγμή 21285. Αυτά τα δύο γεγονότα (events) μετατρέπονται στην εγγραφή επικοινωνίας της πρώτης γραμμής, η οποία περιέχει τις πιο πάνω πληροφορίες μαζί με την πληροφορία πως το μέγεθος της επικοινωνίας είναι 1024 bytes.

Η εντολή εκτέλεσης του προγράμματος paraver_parser είναι η ακόλουθη:

paraver_parser [*.prv file] [*.pcf file] [*.row file]

ΠΡΟΣΟΧΗ! Για μεγάλα *.prv αρχεία (π.χ. 500,000 γραμμές) προτού εκτελέσετε το πρόγραμμα, θα χρειαστεί να αλλάξετε το stack size, καθώς το πρόγραμμα χρησιμοποιεί αναδρομή η οποία φτάνει σε μεγάλο βάθος. Για να το κάνετε αυτό, πληκτρολογείτε στο terminal την εντολή:

ulimit -s [new size in KBytes]

Τέλος, να αναφέρω πως το πρόγραμμα Parser το χρησιμοποιείτε ΜΟΝΟ σε περίπτωση που γνωρίζετε πως ο κώδικας που εκτελείτε τρέχει διαφορετικές εφαρμογές, οι οποίες επικοινωνούν μεταξύ τους. Σε διαφορετική περίπτωση, ΔΕΝ χρειάζεται να χρησιμοποιήσετε τον Parser.

Κεφάλαιο 6

Εργαλείο PARAVER

6.1 Εισαγωγή στο εργαλείο PARAVER

Το PARAVER είναι ένα εργαλείο το οποίο προσφέρει ανάλυση και γραφική απεικόνιση παράλληλων εφαρμογών. Το PARAVER μας προσφέρει μια ποιοτική γενική αντίληψη της συμπεριφοράς μιας εφαρμογής, καθώς και λεπτομερή και ποσοτική ανάλυση των προβλημάτων της. Με αυτό τον τρόπο ο προγραμματιστής μπορεί να ανακαλύψει αν η εφαρμογή του υστερεί σε κάποια σημεία και αν ναι, τότε να επικεντρωθεί σε αυτά τα σημεία με σκοπό την περαιτέρω βελτίωση της απόδοσης της εφαρμογής του. Ως αποτέλεσμα της χρήσης του PARAVER, έχουμε μείωση του χρόνου ανάπτυξης μιας εφαρμογής και ελαχιστοποίηση των hardware πόρων που χρησιμοποιεί αυτή. Για παραγωγή trace αρχείων του PARAVER θα χρησιμοποιούμε το εργαλείο EXTRAE.

Το PARAVER υποστηρίζει:

- Λεπτομερή ποσοτική ανάλυση της απόδοσης προγραμμάτων.
- Ταυτόχρονη συγκριτική ανάλυση πολλαπλών traces.
- Γρήγορη ανάλυση μεγάλων traces.

Τα προγραμματιστικά μοντέλα που υποστηρίζει το PARAVER είναι τα ακόλουθα:

- MPI
- OpenMP
- Pthreads
- CUDA
- OmpSs

Το PARAVER υποστηρίζει δύο είδη όψεων (views) για παρουσίαση των πληροφοριών σχετικά με την απόδοση μιας εφαρμογής:

- **Γραφική Όψη** Αναπαράσταση της συμπεριφοράς της εφαρμογής στο χρόνο, με τρόπο που να μπορεί εύκολα ο χρήστης να την κατανοήσει.
- **Στατιστική Όψη** Αριθμητική ανάλυση των δεδομένων για στοιχεία που θέλει να επιλέξει-εξετάσει ο χρήστης.

Με χρήση του PARAVER μπορούμε να δούμε:

- Τον βαθμό παραλληλισμού μιας εφαρμογής.
- Το χρονοδιάγραμμα του επεξεργαστή.
- Την εξέλιξη της τιμής μιας συγκεκριμένης-επιλεγμένης μεταβλητής.
- Το load balance των διάφορων parallel loops.
- Τις εντολές ανά κύκλο (Instructions/Cycle) για κάθε νήμα.

Μια άλλη δυνατότητα που μας προσφέρει το PARAVER, είναι η ταυτόχρονη προβολή διαφορετικών trace αρχείων. Με αυτό τον τρόπο μπορούμε να συγκρίνουμε:

- Δύο διαφορετικά versions ενός κώδικα.
- Την συμπεριφορά δύο διαφορετικών μηχανών.
- Τις διαφορές μεταξύ δύο εκτελέσεων.
- Τον αντίκτυπο που έχει το μέγεθος προβλήματος (problem size).

6.2 Λήψη, εγκατάσταση και εκτέλεση του εργαλείου PARAVER

Στο σημείο αυτό πρέπει να αναφέρω πως στην προσπάθειά μου για να εγκαταστήσω το PARAVER με χρήση του πηγαίου κώδικά του, αντιμετώπισα πολλά προβλήματα και έχασα αρκετό χρόνο. Μπορείτε να δείτε τα σχετικά προβλήματα και τις λύσεις που μου πρότειναν μέσω emails οι ερευνητές του Barcelona Supercomputing Center (BSC) στο **ΠΑΡΑΡΤΗΜΑ Β**.

Για αποφυγή περαιτέρω καθυστέρησης αποφάσισα όπως αντί του πηγαίου κώδικα, να κατεβάσω την εγκατάσταση για την πλατφόρμα Linux-x86 64 bits. Για να το κάνω αυτό άνοιξα τον ακόλουθο σύνδεσμο:

https://www.bsc.es/computer-sciences/performance-tools/downloads

Εισήγαγα το e-mail μου στο αντίστοιχο πεδίο, επέλεξα το Linux-x86 64 bits όπως φαίνεται στην Εικόνα 6.2.1 και πάτησα το κουμπί Download.

Module name	Version	Description
NEW Paraver	4.3.4 (July 13th 2012)	New Paraver (OPEN SOURCE). Runs under Linux, Windows and Mac OS X (wxWidgets). LGPL license
Platform	**Requirements**	
SOURCES	○ Dependencies: Boost >= 1.36; Zlib; wxWidgets >= 2.8.0; wxPropertyGrid >= 1.4.0	
Linux-x86	○ 32 bits ◉ 64 bits	
Windows	○	
Linux-PowerPC	○	
Linux-Itanium	○	
Mac OS X	○ Mac OS X 10.6	

Εικόνα 6.2.1

Έχοντας κατεβάσει και αποσυμπιέσει τον φάκελο, το μόνο πράγμα που απομένει για να ανοίξουμε ένα trace αρχείο με το εργαλείο PARAVER είναι:

- Να καθορίσουμε το μονοπάτι στο οποίο βρίσκεται ο φάκελος PARAVER, εκτελώντας την εντολή:
  export PARAVER_HOME=DIR
- Να ανοίξουμε ένα trace αρχείο εκτελώντας την εντολή:
  $PARAVER_HOME/bin/wxparaver [trace_file_name.prv**]**

6.3 PARAVER Object Model

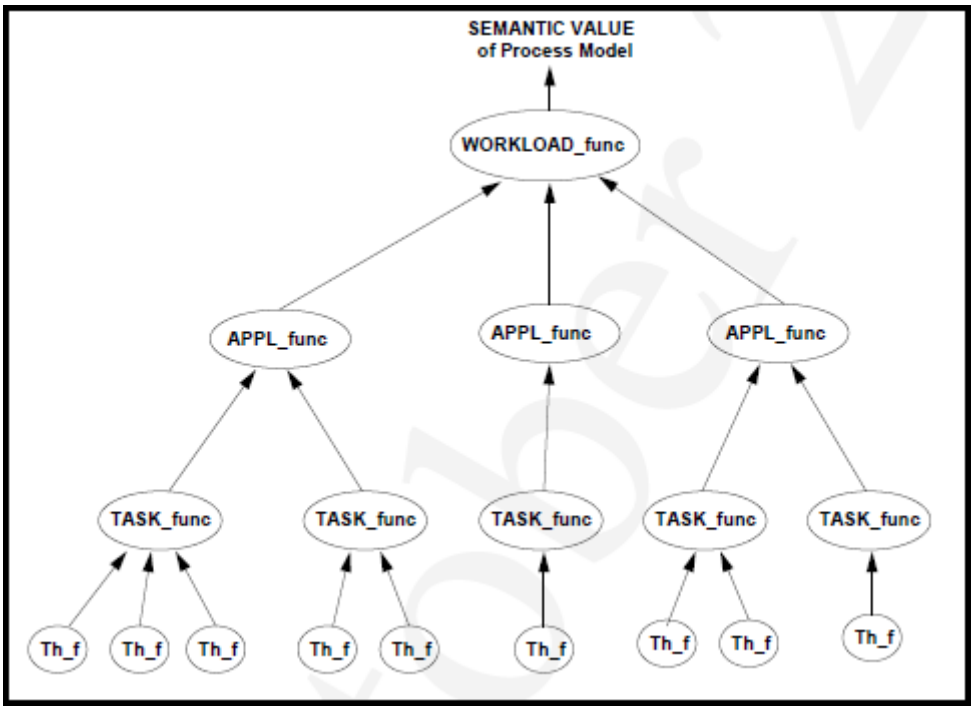
Η λειτουργία του PARAVER βασίζεται σε δύο μοντέλα. Το **process model** και το **resource model**. Ανάλογα με το μοντέλο και το επίπεδο αυτού του μοντέλου που θα επιλέξει ο χρήστης, το PARAVER θα δώσει την ανάλογη αναπαράσταση.

- **Process Model**

  Τα επίπεδα που αποτελούν αυτό το μοντέλο είναι τα ακόλουθα:

  - ✓ Σύνολο εφαρμογών (WORKLOAD)
  - ✓ Εφαρμογή (APPL)
  - ✓ Διαδικασία (TASK)
  - ✓ Νήμα (THREAD)

  Για να υπολογίσει το PARAVER μια τιμή που αναπαρίσταται σε ένα αντικείμενο που χρησιμοποιεί ένα από τα πιο πάνω επίπεδα, χρησιμοποιεί συναρτήσεις οι οποίες υπολογίζουν ιεραρχικά (από το χαμηλότερο στο ψηλότερο επίπεδο) τις τιμές των χαμηλότερων επιπέδων (Εικόνα 6.3.1). Αν για παράδειγμα θέλω το PARAVER να προβάλει πληροφορίες σε επίπεδο TASK, οι τιμές θα υπολογίζονται με βάση τις τιμές των νημάτων (THREAD) που αποτελούν το TASK.



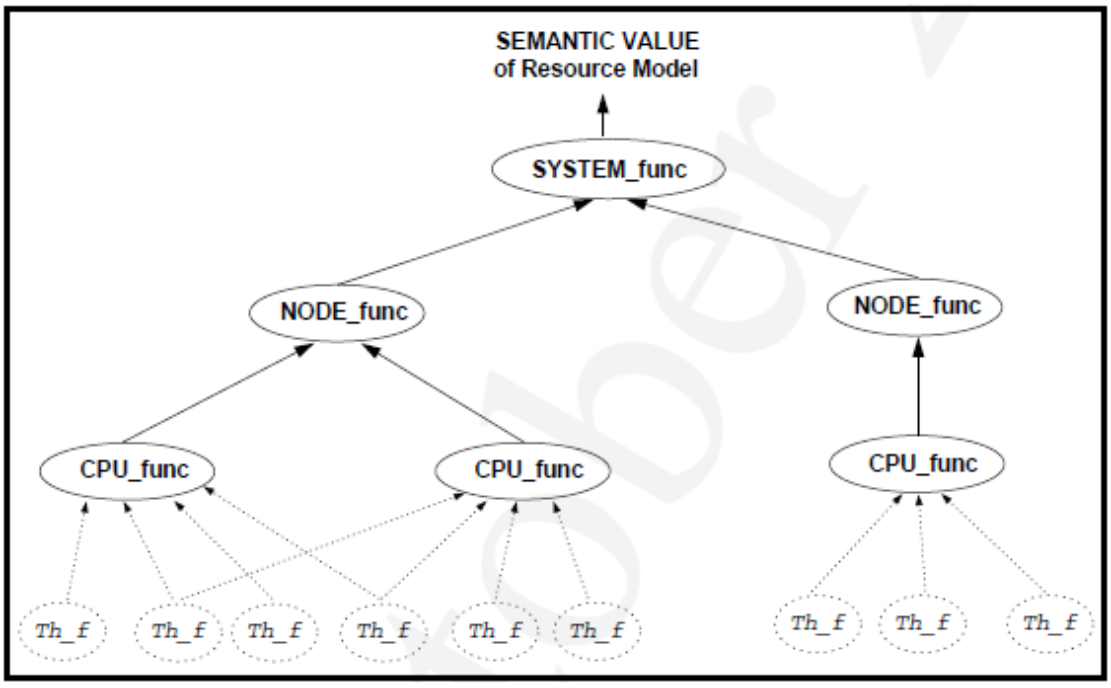
Εικόνα 6.3.1

- **Resource Model**

  Τα επίπεδα που αποτελούν αυτό το μοντέλο είναι τα ακόλουθα:

  - ✓ Σύμπλεγμα από κόμβους (SYSTEM)
  - ✓ Κόμβος – Σύνολο από επεξεργαστές (NODE)
  - ✓ Επεξεργαστής (CPU)

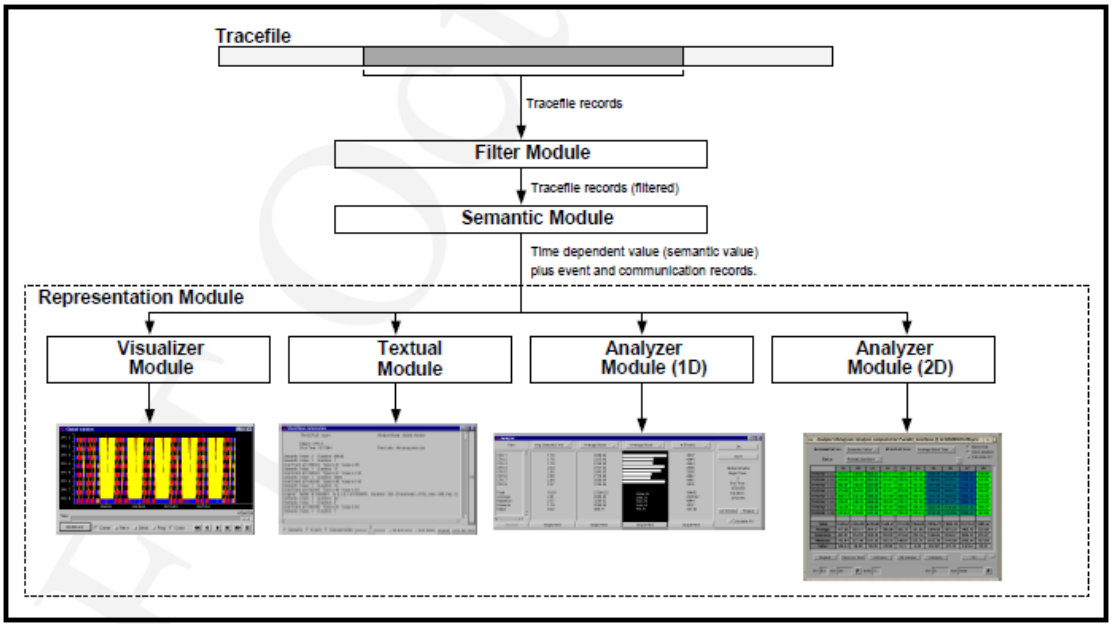
  Η τιμή που αναπαρίσταται σε ένα αντικείμενο που χρησιμοποιεί το CPU επίπεδο, είναι η τιμή του νήματος που εκτελείται την συγκεκριμένη χρονική στιγμή στον επεξεργαστή. Για τα άλλα δύο επίπεδα (SYSTEM και NODE) το PARAVER χρησιμοποιεί συναρτήσεις οι οποίες υπολογίζουν ιεραρχικά (από το χαμηλότερο στο ψηλότερο επίπεδο) τις τιμές των χαμηλότερων επιπέδων (Εικόνα 6.3.2).



Εικόνα 6.3.2

6.4 Εσωτερική δομή PARAVER

Το PARAVER εξάγει πληροφορίες από το trace αρχείο κάνοντας χρήση τριών μονάδων (modules), οι οποίες είναι το **Filter Module**, το **Semantic Module** και το **Representation Module** (Εικόνα 6.4.1).



Εικόνα 6.4.1

- **Filter Module**

  Με βάση τις επιλογές που θα κάνει ο χρήστης του εργαλείου PARAVER, η μονάδα αυτή θα παρέχει στις άλλες δύο μονάδες μερική όψη του trace αρχείου. Σκοπός της μονάδας αυτής είναι η επιλογή-μείωση της πληροφορίας που αντλείται από το trace αρχείο και την οποία παρουσιάζει το PARAVER. Για παράδειγμα ο χρήστης μπορεί να επιλέξει να αντλούνται πληροφορίες για events με συγκεκριμένο type από το trace αρχείο. Όσα events έχουν διαφορετικό type από αυτό που επέλεξε ο χρήστης δεν θα λαμβάνονται υπόψη.

- **Semantic Module**

  Αποτελεί το κλειδί του PARAVER. Αφού λάβει την φιλτραρισμένη πληροφορία από την προηγούμενη μονάδα (Filter), είναι υπεύθυνο για να υπολογίσει την τιμή που θα παρουσιάσει το PARAVER για κάθε αντικείμενο, ανάλογα με το μοντέλο (process ή resource model) και το επίπεδο που έχει επιλεχτεί. Για να υπολογίσει

αυτή την τιμή, κάνει χρήση των συναρτήσεων του process ή resource model, όπως φαίνεται στην Εικόνα 6.3.1 και στην Εικόνα 6.3.2.

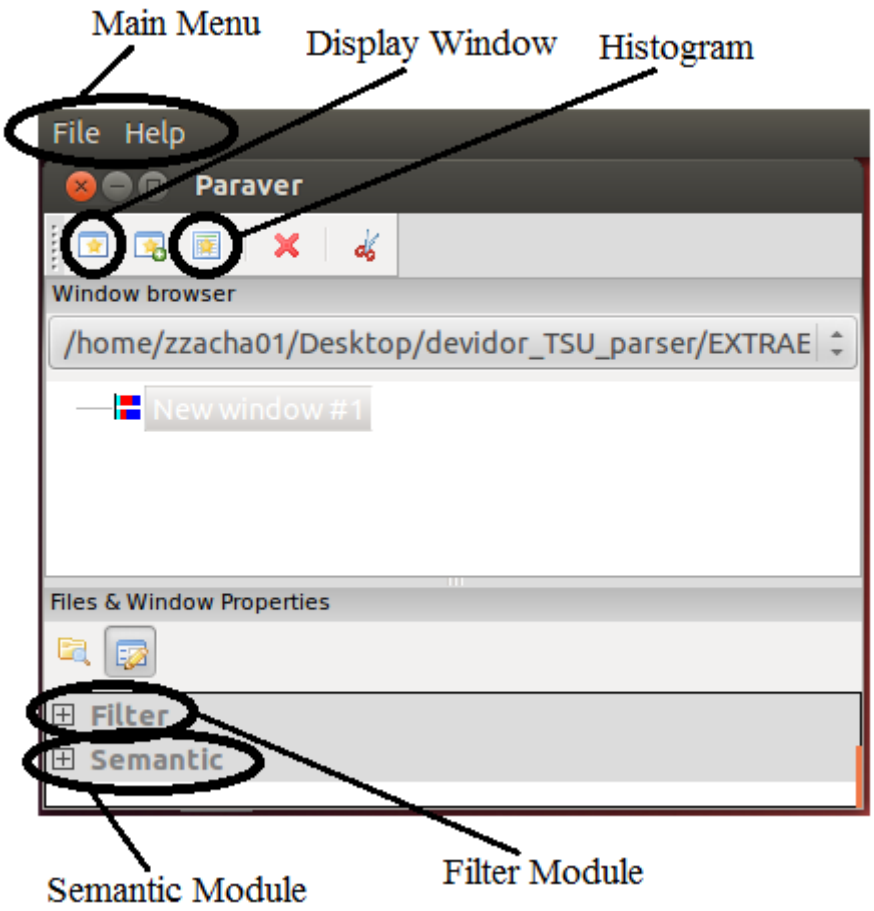
- **Representation Module**

  Αποτελεί την τελευταία μονάδα του PARAVER και χωρίζεται σε δύο μονάδες:

  - ✓ **Visualization Module** Αυτή η μονάδα είναι υπεύθυνη για την γραφική αναπαράσταση του trace αρχείου. Στην μονάδα αυτή καθορίζεται το πώς θα εμφανιστούν τα events, values και γενικά όλα τα δεδομένα που παραλήφθηκαν από την προηγούμενη μονάδα (Semantic Module). Επίσης, η μονάδα αυτή είναι υπεύθυνη για θέματα όπως το window size, display scale κτλ.
  - ✓ **Analyzer Module** Αυτή η μονάδα είναι υπεύθυνη για την παρουσίαση διάφορων στατιστικών μετρήσεων που αφορούν όλα τα δεδομένα που παραλήφθηκαν από την προηγούμενη μονάδα (Semantic Module). Για παράδειγμα παρουσιάζει τον αριθμό των events, των επικοινωνιών κτλ.

6.5 Κύρια όψη PARAVER

Όταν εκκινήσουμε το PARAVER, έχουμε την εμφάνιση που βλέπουμε στην Εικόνα 6.5.1.

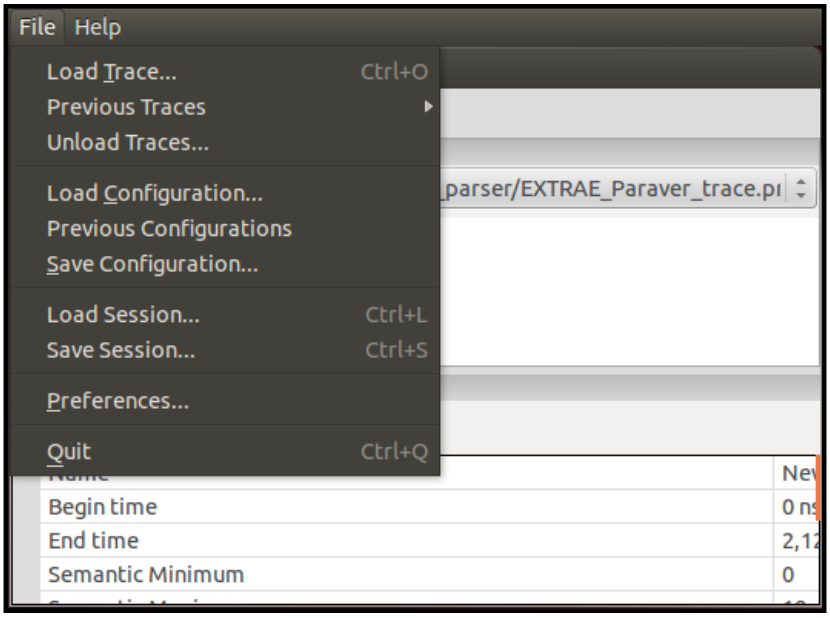


Εικόνα 6.5.1

Όπως παρατηρείτε από την Εικόνα 6.5.1, μπορούμε να χωρίσουμε τις επιλογές που μας προσφέρει το PARAVER σε πέντε κατηγορίες. Το **Main Menu**, το **Display Window**, το **Histogram**, το **Filter Module** και το **Semantic Module**.

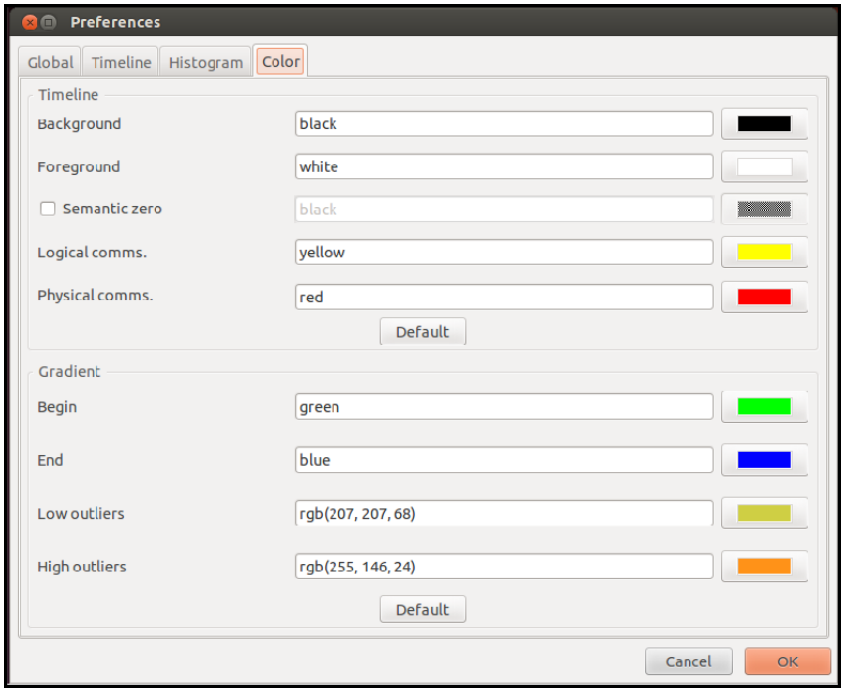
- **Main Menu**

  Το Main Menu αποτελείται από δύο επιλογές Menu. Το **File Menu** και το **Help Menu**.

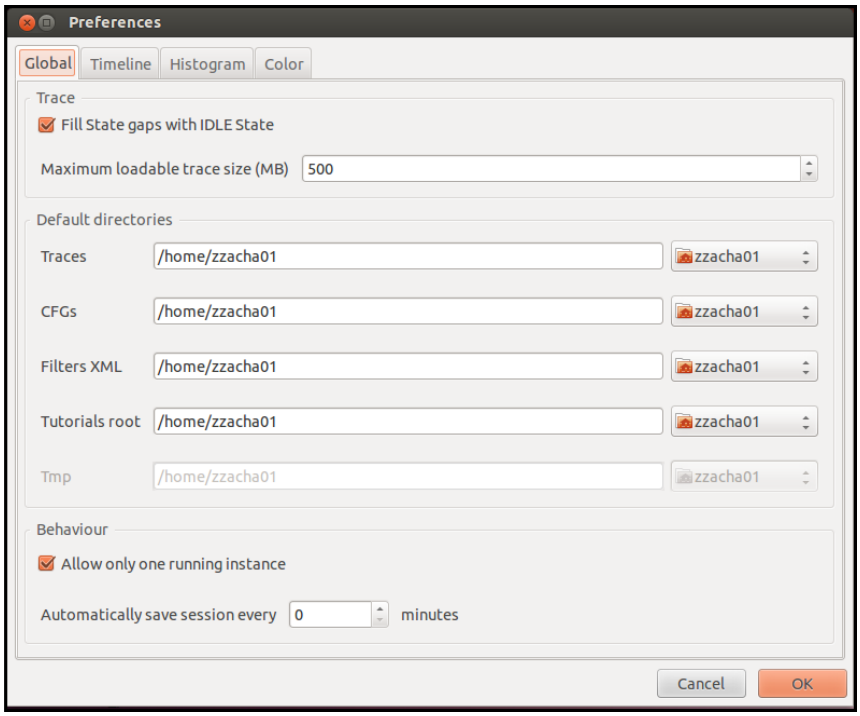


Εικόνα 6.5.2

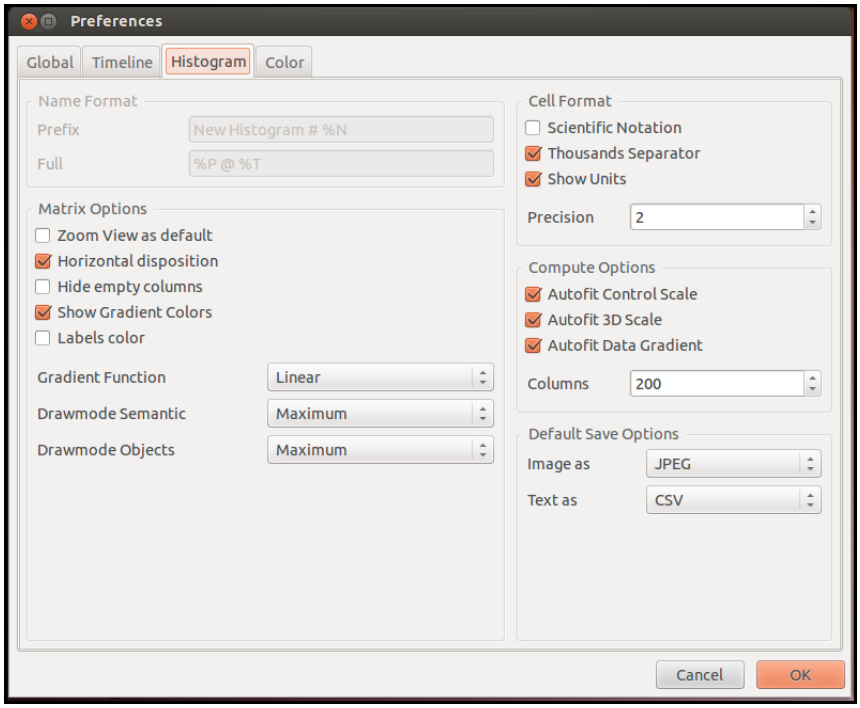
- ✓ **File Menu** Όπως φαίνεται και στην Εικόνα 6.5.2, το File Menu μας παρέχει τις πιο κάτω επιλογές:
  - **Load Trace** Ανοίγει ένα νέο παράθυρο στο οποίο επιλέγουμε το μονοπάτι του trace αρχείου που θέλουμε να ανοίξουμε.
  - **Previous Traces** Μια λίστα με τα trace αρχεία που φορτώθηκαν προηγουμένως. Επιτρέπει το γρήγορο άνοιγμα ενός trace αρχείου που φορτώθηκε προηγουμένως.
  - **Unload Traces** Ανοίγει ένα νέο παράθυρο στο οποίο επιλέγουμε τα trace αρχεία που θέλουμε να κάνουμε unload.
  - **Load Configuration** Για να φορτώσουμε ένα window configuration αρχείο.
  - **Save Configuration** Για να αποθηκεύσουμε ένα window configuration αρχείο.
  - **Preferences** Επιτρέπει στον χρήστη να αλλάξει κάποιες ρυθμίσεις που αφορούν τα χρώματα που χρησιμοποιεί το PARAVER (Εικόνα 6.5.3), τους φακέλους στους οποίους ψάχνει το PARAVER για να βρει τα αρχεία που χρησιμοποιεί (Εικόνα 6.5.4), την εμφάνιση του Analyzer Module – Histogram (Εικόνα 6.5.5) και την εμφάνιση του Visualization Module – Display Window (Εικόνα 6.5.6).
  - **Quit** Έξοδος από το εργαλείο PARAVER.



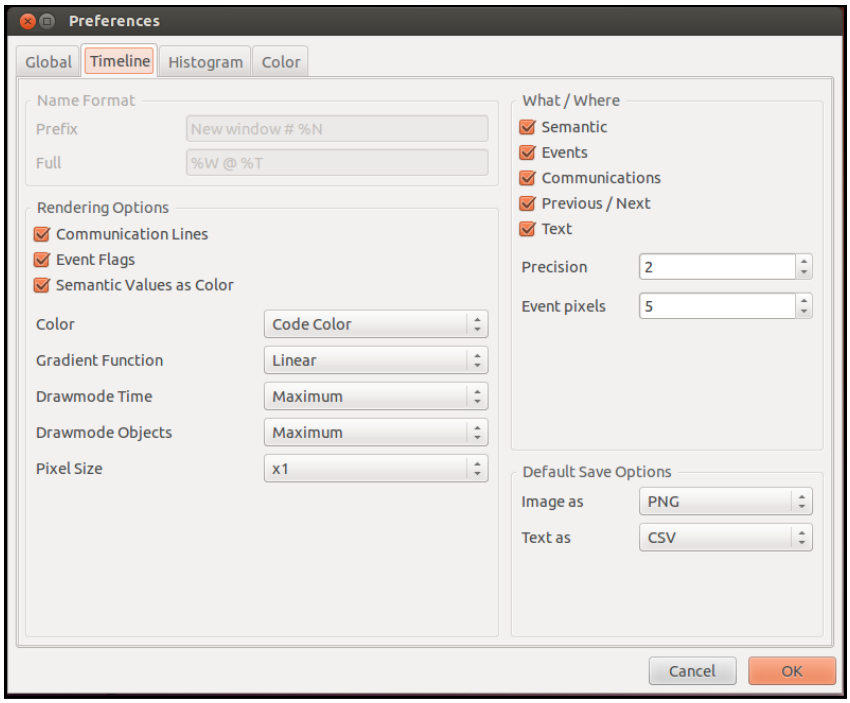
Εικόνα 6.5.3



Εικόνα 6.5.4



Εικόνα 6.5.5

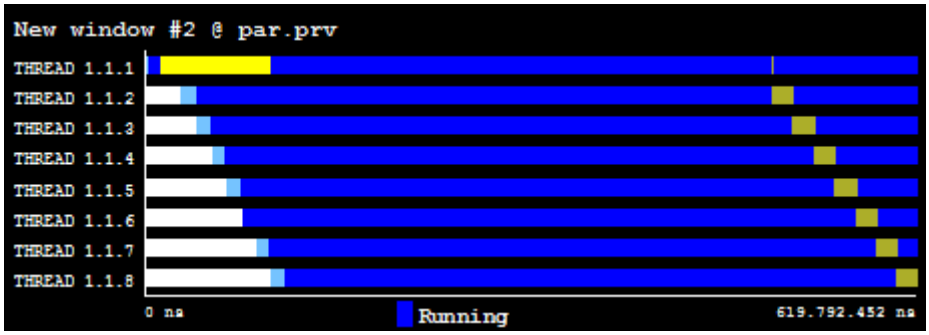


Εικόνα 6.5.6

- ✓ **Help Menu** Ο χρήστης στο μενού αυτό μπορεί να βρει κάποια tutorials (επιλογή "Tutorials") που θα τον βοηθήσουν να μάθει να λειτουργεί το εργαλείο PARAVER. Επίσης, μπορεί να δει τα PARAVER credits (επιλογή "About").

- **Display Window**

  Πατώντας πάνω στο εικονίδιο Display Window, εμφανίζεται στην οθόνη το Visualization Module το οποίο περιέχει την γραφική αναπαράσταση του trace αρχείου (Εικόνα 6.5.7).



Εικόνα 6.5.7

- **Histogram**

  Πατώντας πάνω στο εικονίδιο Histogram, εμφανίζεται στην οθόνη το Analyzer Module το οποίο περιέχει στατιστικά δεδομένα για το trace αρχείο (Εικόνα 6.5.8).

	Running	Not created	I/O	Others
THREAD 1.1.1	423.375.499 ns	222.469.886.852.801 ns	6.705 ns	489 ns
THREAD 2.1.1	14.202.651 ns	222.470.297.689.924 ns	4.819 ns	489 ns
Total	437.578.150 ns	444.940.184.542.725 ns	11.524 ns	978 ns
Average	218.789.075 ns	222.470.092.271.362,50 ns	5.762 ns	489 ns
Maximum	423.375.499 ns	222.470.297.689.924 ns	6.705 ns	489 ns
Minimum	14.202.651 ns	222.469.886.852.801 ns	4.819 ns	489 ns
StDev	204.586.424,00 ns	205.413.870,58 ns	943 ns	0 ns
Avg/Max	0,52	1,00	0,86	1

Εικόνα 6.5.8

- **Filter Module**

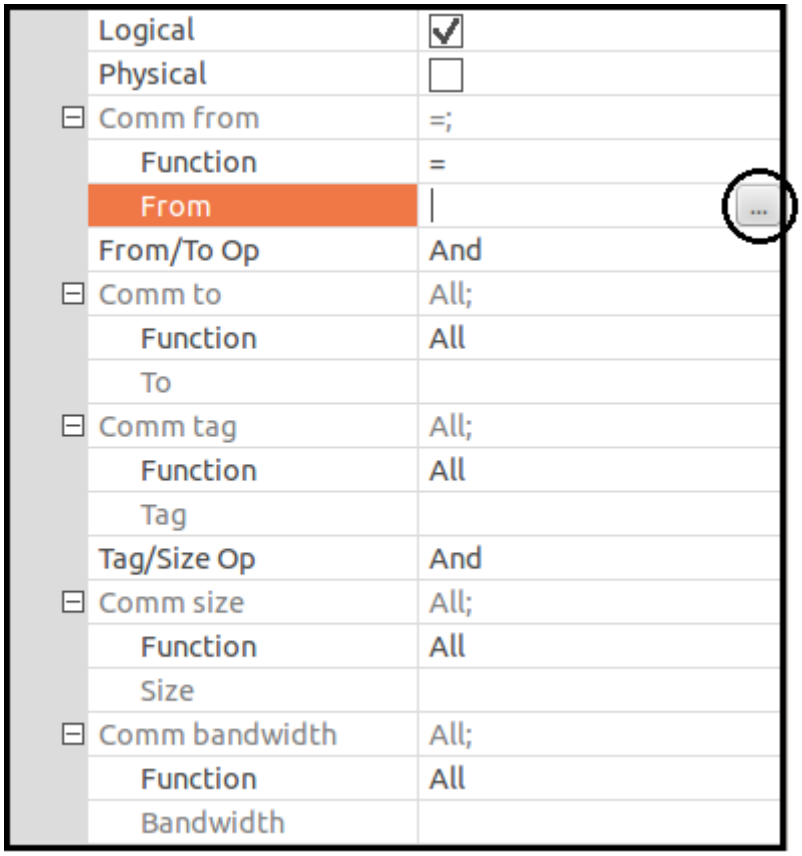
  Πατώντας πάνω στο εικονίδιο «+» που βρίσκεται αριστερά του Filter (Εικόνα 6.5.1), ο χρήστης μπορεί να επιλέξει ποιες πληροφορίες θέλει να αντλούνται από το trace αρχείο. Περισσότερες πληροφορίες για το Filter Module μπορείτε να διαβάσετε στο **Μέρος 6.6** αυτού του Κεφαλαίου.

- **Semantic Module**

  Πατώντας πάνω στο εικονίδιο «+» που βρίσκεται αριστερά του Semantic (Εικόνα 6.5.1), ο χρήστης μπορεί να επιλέξει με ποιο τρόπο θα παρουσιάσει το PARAVER τις πληροφορίες που έλαβε από το Filter Module. Περισσότερες πληροφορίες για το Semantic Module μπορείτε να διαβάσετε στο **Μέρος 6.7** αυτού του Κεφαλαίου.

6.6 Filter Module

Όπως θα έχετε ήδη διαβάσει με την χρήση του Filter Module ο χρήστης μπορεί να επιλέξει ποιες πληροφορίες θέλει να αντλούνται από το trace αρχείο. Πιο συγκεκριμένα μπορεί να επιλέξει ποια events και ποιες επικοινωνίες θα περάσουν στις επόμενες δύο μονάδες (Semantic και Representation Module).



Εικόνα 6.6.1

- **Φιλτράρισμα Επικοινωνιών (Εικόνα 6.6.1)**

  Ο χρήστης μπορεί να επιλέξει το είδος της επικοινωνίας που θέλει να περάσει στις επόμενες δύο μονάδες. Οι επιλογές που έχει είναι ανάμεσα σε:

  - ✓ **Logical** Πότε καλούνται οι συναρτήσεις send/receive μέσα στον κώδικα. Η προκαθορισμένη λειτουργία του PARAVER είναι να δουλεύει με τις logical επικοινωνίες.
  - ✓ **Physical** Πότε πραγματικά αποστέλλεται ή παραλαμβάνεται το μήνυμα.

  Επίσης, ο χρήστης μπορεί να επιλέξει μεταξύ ποιων ζευγαριών γίνεται η επικοινωνία που θέλει να περάσει στις επόμενες δύο μονάδες. Στο πεδίο "From" εισάγει το αντικείμενο αποστολέα και στο πεδίο "To" το αντικείμενο παραλήπτη. Οι πράξεις που μπορούν να εφαρμοστούν μεταξύ του "From" και του "To" είναι:

  - ✓ **Logical AND**
  - ✓ **Logical OR**

  Στα πεδία "From" και "To" ο χρήστης μπορεί να βάλει τα διάφορα αντικείμενα που θέλει να φιλτράρει με δύο τρόπους:

  - ✓ Γράφοντας τα ένα προς ένα, χωρισμένα με κόμμα.
  - ✓ Πατώντας πάνω στο εικονίδιο που είναι κυκλωμένο στην Εικόνα 6.6.1. Το εικονίδιο αυτό ανοίγει ένα παράθυρο το οποίο περιέχει όλα τα αντικείμενα του trace αρχείου. Ο χρήστης μπορεί να επιλέξει ποια από αυτά τα αντικείμενα επιθυμεί να περάσει στις επόμενες δύο μονάδες.

  Οι πράξεις που μπορούν να εφαρμοστούν στα πεδία "From" και "To" είναι οι ακόλουθες:

  - ✓ **All** Όλα τα αντικείμενα.
  - ✓ **!=** Όλα τα αντικείμενα, εκτός από αυτά που είναι γραμμένα στο πεδίο.
  - ✓ > Τα αντικείμενα που είναι μεγαλύτερα από την τιμή στο πεδίο.
  - ✓ < Τα αντικείμενα που είναι μικρότερα από την τιμή στο πεδίο.
  - ✓ = Τα αντικείμενα που είναι γραμμένα στο πεδίο.
  - ✓ **None** Κανένα αντικείμενο.

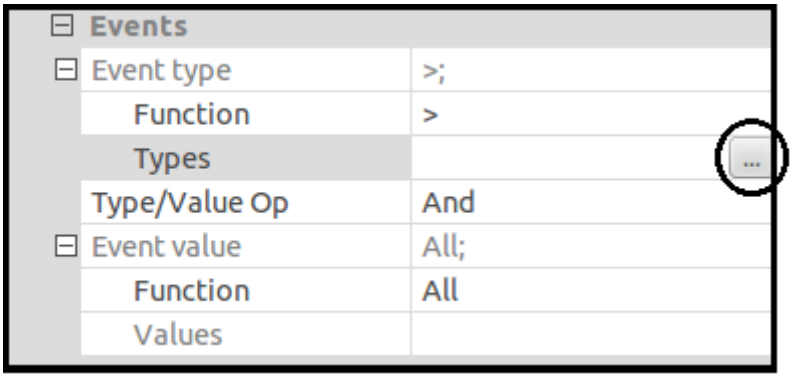
  Τέλος, ο χρήστης μπορεί να επιλέξει τι χαρακτηριστικά πρέπει να έχει η επικοινωνία που θέλει να περάσει στις επόμενες δύο μονάδες. Στο πεδίο "tag" εισάγει το tag και στο πεδίο "size" το μέγεθος της επικοινωνίας. Οι πράξεις που μπορούν να εφαρμοστούν μεταξύ του "tag" και του "size" είναι:

- ✓ **Logical AND**
- ✓ **Logical OR**

Στα πεδία “tag” και “size” ο χρήστης μπορεί να βάλει τα διάφορα tags και sizes που θέλει να φιλτράρει γράφοντας τα ένα προς ένα, χωρισμένα με το ελληνικό ερωτηματικό (**;**).

Οι πράξεις που μπορούν να εφαρμοστούν στα πεδία “tag” και “size” είναι οι ακόλουθες:

- ✓ **All** Όλες οι επικοινωνίες.
- ✓ **!=** Όλες οι επικοινωνίες, εκτός αυτών που είναι γραμμένες στο πεδίο.
- ✓ > Οι επικοινωνίες που είναι μεγαλύτερες από την τιμή στο πεδίο.
- ✓ < Οι επικοινωνίες που είναι μικρότερες από την τιμή στο πεδίο.
- ✓ = Οι επικοινωνίες που είναι γραμμένες στο πεδίο.
- ✓ **None** Καμιά επικοινωνία.



Εικόνα 6.6.2

- **Φιλτράρισμα Events (Εικόνα 6.6.2)**

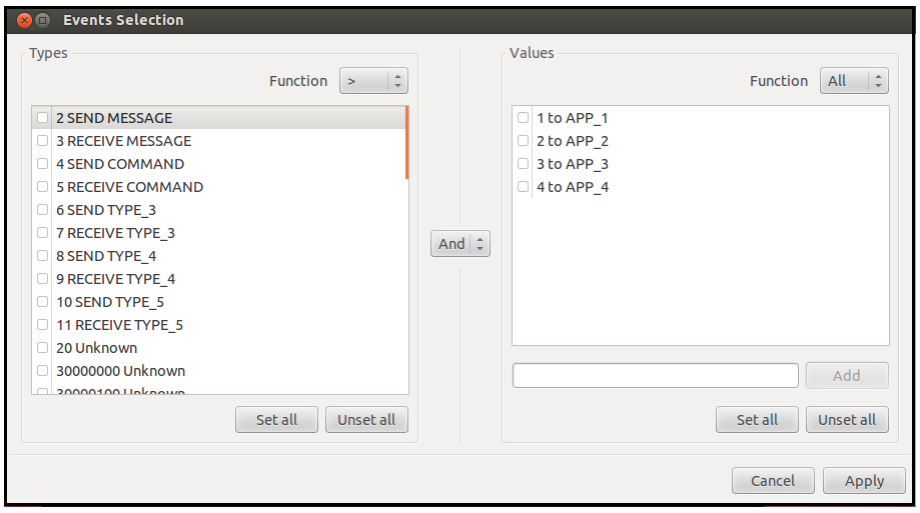
Ο χρήστης μπορεί να επιλέξει το είδος (type) και την τιμή (value) των events που θέλει να περάσει στις επόμενες δύο μονάδες. Στο πεδίο “Types” εισάγει τον τύπο και στο πεδίο “Values” την τιμή. Οι πράξεις που μπορούν να εφαρμοστούν μεταξύ του “Types” και του “Values” είναι:

- ✓ **Logical AND**
- ✓ **Logical OR**

Στα πεδία “Types” και “Values” ο χρήστης μπορεί να βάλει τα διάφορα types και values που θέλει να φιλτράρει με δύο τρόπους:

- ✓ Γράφοντας τα ένα προς ένα, χωρισμένα με ερωτηματικό (**;**).

- ✓ Πατώντας πάνω στο εικονίδιο που είναι κυκλωμένο στην Εικόνα 6.6.2. Το εικονίδιο αυτό ανοίγει ένα παράθυρο το οποίο περιέχει όλα τα types και values του trace αρχείου (Εικόνα 6.6.3). Ο χρήστης μπορεί να επιλέξει ποια από αυτά τα types και values επιθυμεί να περάσει στις επόμενες δύο μονάδες, καθώς και τις πράξεις που επιθυμεί να εφαρμοστούν πάνω σε αυτά.



Εικόνα 6.6.3

Οι πράξεις που μπορούν να εφαρμοστούν στα πεδία "Types" και "Values" είναι οι ακόλουθες:

- ✓ **All** Όλα τα events.
- ✓ **!=** Όλα τα events, εκτός από αυτά που είναι γραμμένα στο πεδίο.
- ✓ > Τα events που είναι μεγαλύτερα από την τιμή στο πεδίο.
- ✓ < Τα events που είναι μικρότερα από την τιμή στο πεδίο.
- ✓ = Τα events που είναι γραμμένα στο πεδίο.
- ✓ **None** Κανένα event.

6.7 Semantic Module

Όπως θα έχετε ήδη διαβάσει με την χρήση του Semantic Module ο χρήστης μπορεί να καθορίσει πως θα παρουσιάσει το PARAVER τα values, ανάλογα πάντα και με το μοντέλο που χρησιμοποιεί (Process ή Resource Model). Ακόμη, με συνδυασμό διαφορετικών όψεων (views) ο χρήστης μπορεί να παράγει νέες μετρήσεις. Αν για παράδειγμα έχει ένα view το οποίο μετρά τις εντολές (instructions) και ένα view το οποίο μετρά τους κύκλους (cycles), με μια διαίρεση των δύο views μπορεί να πάρει την τιμή εντολές ανά κύκλο (instructions/cycle).

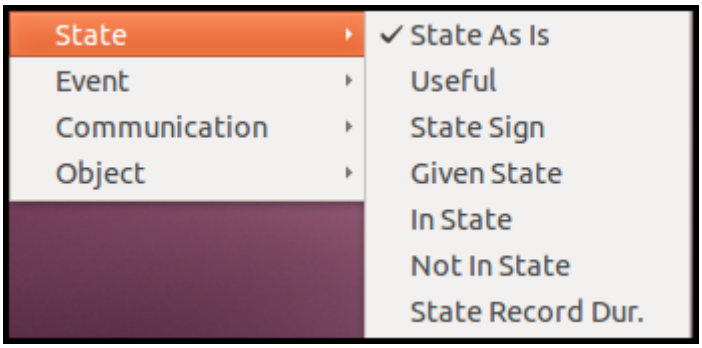
Ανάλογα με το επίπεδο στο οποίο δουλεύει ο χρήστης, θα προσφέρονται και οι συναρτήσεις που βρίσκονται σε αυτό το επίπεδο και όλα τα κατώτερα επίπεδα αυτού. Αν για παράδειγμα ο χρήστης δουλεύει σε THREAD level, θα είναι διαθέσιμες μόνο οι συναρτήσεις για THREAD. Αν δουλεύει σε APPLICATION level, θα είναι διαθέσιμες μόνο οι συναρτήσεις για THREAD, TASK και APPLICATION. Τέλος, αν δουλεύει σε NODE level, θα είναι διαθέσιμες μόνο οι συναρτήσεις για THREAD, CPU και NODE.

Πιο κάτω θα κάνω μια περιγραφή για τις συναρτήσεις που προσφέρει το Semantic Module σε κάθε επίπεδο.

- **Επίπεδο THREAD**

  Στο επίπεδο αυτό οι συναρτήσεις μπορούν να χωριστούν σε τέσσερα μέρη. Στις συναρτήσεις που αφορούν καταστάσεις (states), events, επικοινωνίες και αντικείμενα.

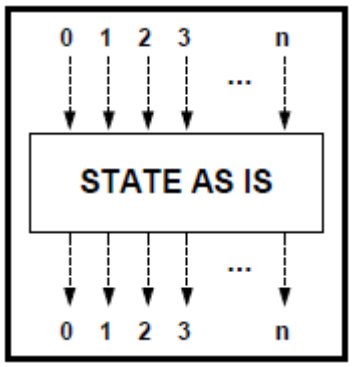
  - ✓ **Καταστάσεις (states)**



Εικόνα 6.7.1

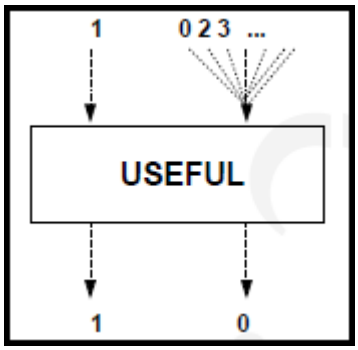
Όπως παρατηρείτε από την Εικόνα 6.7.1 οι συναρτήσεις που προσφέρονται για καταστάσεις (states) είναι οι ακόλουθες:

- **State As Is** Παίρνει την τιμή της κατάστασης (state) και την επιστρέφει αυτούσια, χωρίς να την αλλάξει (Εικόνα 6.7.2).



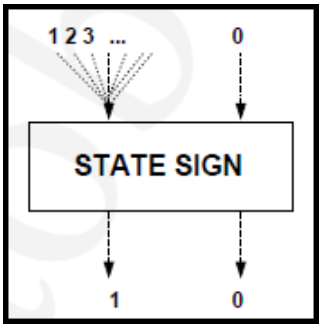
Εικόνα 6.7.2

- **Useful** Παίρνει την τιμή της κατάστασης (state) και αν είναι running (1) την επιστρέφει αυτούσια, χωρίς να την αλλάξει. Αν είναι οποιαδήποτε άλλη τιμή εκτός running (1) επιστρέφει την τιμή idle (0) (Εικόνα 6.7.3).



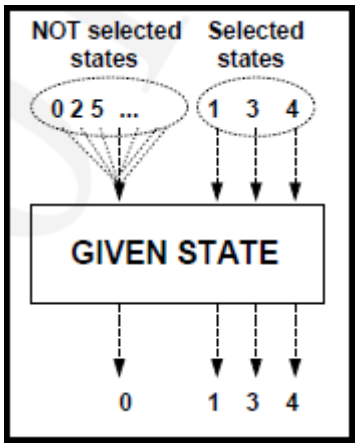
Εικόνα 6.7.3

- **State Sign** Παίρνει την τιμή της κατάστασης (state) και αν δεν είναι idle (0), επιστρέφει τιμή running (1). Αν είναι idle (0), την επιστρέφει αυτούσια χωρίς να την αλλάξει (Εικόνα 6.7.4).



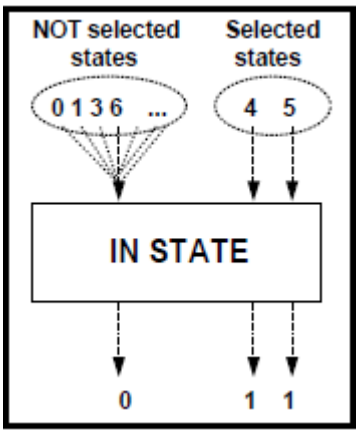
Εικόνα 6.7.4

- **Given State** Παίρνει την τιμή της κατάστασης (state) και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης την επιστρέφει αυτούσια, χωρίς να την αλλάξει. Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή idle (0) (Εικόνα 6.7.5).



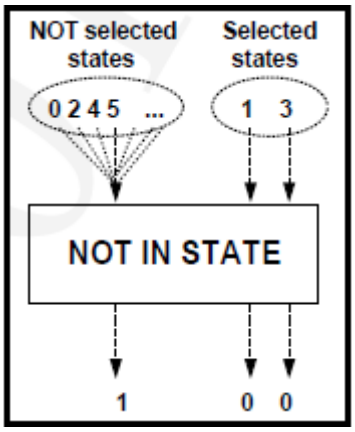
Εικόνα 6.7.5

- **In State** Παίρνει την τιμή της κατάστασης (state) και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή running (1). Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή idle (0) (Εικόνα 6.7.6).



Εικόνα 6.7.6

- **Not In State** Παίρνει την τιμή της κατάστασης (state) και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή idle (0). Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή running (1) (Εικόνα 6.7.7).

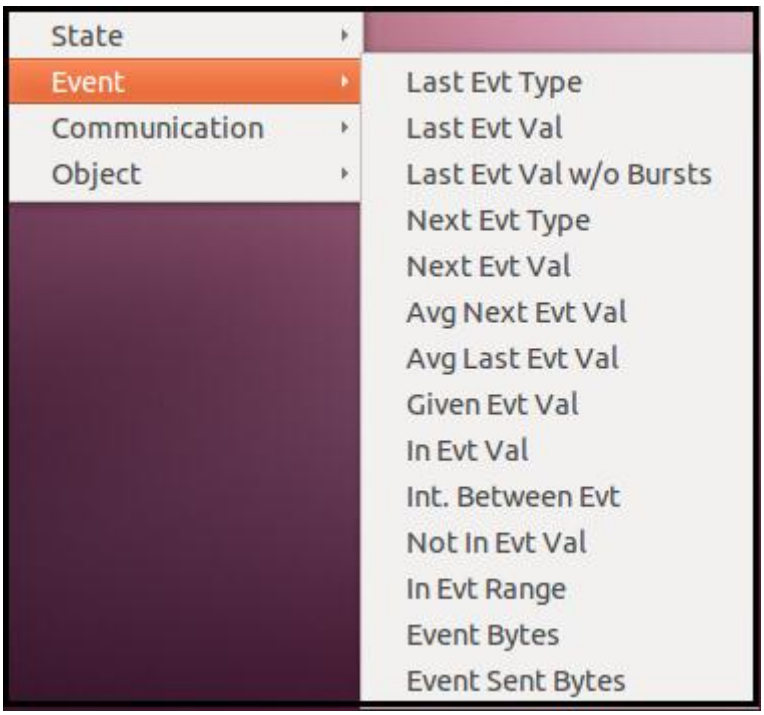


Εικόνα 6.7.7

- **State Record Dur.** Επιστρέφει την χρονική διάρκεια όσων καταστάσεων (states) είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης.

ΠΡΟΣΟΧΗ! Όταν ο χρήστης καθορίζει ένα σύνολο τιμών, χωρίζει τις τιμές μεταξύ τους με ένα ερωτηματικό (**;**).

✓ **Events**



Εικόνα 6.7.8

Όπως παρατηρείτε από την Εικόνα 6.7.8 οι συναρτήσεις που προσφέρονται για events είναι οι ακόλουθες:

- **Last Evt Type** Επιστρέφει το type του τελευταίου event, μέχρι να βρει ένα άλλο event στο trace.
- **Last Evt Value** Επιστρέφει το value του τελευταίου event, μέχρι να βρει ένα άλλο event στο trace.
- **Next Evt Type** Όταν βρει ένα event στο trace, επιστρέφει το type του επόμενου event.
- **Next Evt Val** Όταν βρει ένα event στο trace, επιστρέφει το value του επόμενου event. Αυτό είναι χρήσιμο σε περίπτωση που τα events μας μετρούν cache misses. Τα cache misses ωστόσο που μετρά ένα event αφορούν την περίοδο πριν το event αυτό. Με χρήση της συνάρτηση αυτής θα βλέπουμε τον αριθμό των cache misses που συνέβηκαν κατά την περίοδο αυτή.
- **Avg Next Evt Val** Παρόμοιο με το Next Evt Val (χρησιμοποιεί το value του επόμενου event), αλλά επιστρέφει μια τιμή συναρτήσει της χρονικής περιόδου μεταξύ του τρέχοντος event και του επόμενου event. Η τιμή υπολογίζεται με βάση τον τύπο της Εικόνας 6.7.9.

$$\frac{value_{i+1} * factor}{t_{i+1} - t_i}$$

Εικόνα 6.7.9

Όπου value ίσο με το value του επόμενου event, factor μια τιμή που καθορίζει ο χρήστης, t_{i+1} η χρονική στιγμή του επόμενου event και t_i η χρονική στιγμή του τρέχοντος event. Το factor καθορίζεται με βάση την μονάδα μέτρησης που θέλει να πάρει ο χρήστης. Αν για παράδειγμα το PARAVER χρησιμοποιεί milliseconds και ο χρήστης θέλει το αποτέλεσμα σε seconds, το factor θα πρέπει να ισούται με 1000.

- **Avg Last Evt Val** Παρόμοιο με το Avg Next Evt Val με την διαφορά πως εδώ χρησιμοποιείται το value του τρέχοντος event.

- **Given Evt Val** Παίρνει την τιμή value του event και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης την επιστρέφει αυτούσια, χωρίς να την αλλάξει. Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή idle (0).
- **In Evt Val** Παίρνει την τιμή value του event και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή running (1). Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή idle (0).
- **Int. Between Evt** Επιστρέφει την χρονική περίοδο που μεσολαβεί από ένα event μέχρι το επόμενο event, ανεξαρτήτως του type και value τους.

ΠΡΟΣΟΧΗ! Όταν ο χρήστης καθορίζει ένα σύνολο τιμών, χωρίζει τις τιμές μεταξύ τους με ένα ερωτηματικό (**;**).

✓ **Επικοινωνίες**

Αν και η νέα έκδοση του PARAVER προσφέρει περισσότερες συναρτήσεις απ' ότι η παλιά έκδοση, μόνο δύο από αυτές είναι πραγματικά χρήσιμες:

- **Send BandWidth** Μας επιστρέφει το εύρος ζώνης που χρησιμοποιείται κατά την αποστολή ενός μηνύματος.
- **Recv BandWidth** Μας επιστρέφει το εύρος ζώνης που χρησιμοποιείται κατά την παραλαβή ενός μηνύματος.

Και στις δύο συναρτήσεις η τιμή του byte factor που καθορίζει ο χρήστης πρέπει να είναι ίση με (1), αφού το μέγεθος των μηνυμάτων που χρησιμοποιούμε εκφράζεται σε bytes.

Όσον αφορά τις υπόλοιπες συναρτήσεις, μπορούμε να πάρουμε τις ίδιες πληροφορίες με απλή παρατήρηση στα σημεία όπου γίνεται η επικοινωνία και χωρίς να χρειαστεί να τις χρησιμοποιήσουμε.

- ✓ **Αντικείμενα**

  Οι συναρτήσεις που προσφέρονται για αντικείμενα είναι οι ακόλουθες:

  - **Thread ID** Επιστρέφει την αναγνωριστική τιμή του νήματος. Αυτή η συνάρτηση είναι χρήσιμη όταν θέλουμε να δούμε ποια νήματα τρέχουν σε ένα επεξεργαστή.
  - **Task ID** Επιστρέφει την αναγνωριστική τιμή της διεργασίας στην οποία ανήκει το νήμα. Αυτή η συνάρτηση είναι χρήσιμη όταν θέλουμε να δούμε ποιες διεργασίες τρέχουν σε ένα επεξεργαστή.
  - **Application ID** Επιστρέφει την αναγνωριστική τιμή της εφαρμογής στην οποία ανήκει το νήμα. Αυτή η συνάρτηση είναι χρήσιμη όταν θέλουμε να δούμε ποιες εφαρμογές τρέχουν σε ένα επεξεργαστή.
  - **Cpu ID** Επιστρέφει την αναγνωριστική τιμή του επεξεργαστή στον οποίο τρέχει το νήμα. Αυτή η συνάρτηση είναι χρήσιμη όταν θέλουμε να δούμε σε ποιους επεξεργαστές τρέχει ένα νήμα.
  - **Node ID** Επιστρέφει την αναγνωριστική τιμή του κόμβου στον οποίο τρέχει το νήμα. Αυτή η συνάρτηση είναι χρήσιμη όταν θέλουμε να δούμε σε ποιους κόμβους τρέχει ένα νήμα.

- **Επίπεδο Process Model**

  Στο επίπεδο αυτό οι συναρτήσεις μπορούν να χωριστούν σε τρία μέρη. Στις συναρτήσεις που αφορούν διεργασίες (tasks), εφαρμογές (applications) και workload.

  - ✓ **Διεργασίες (tasks)**

    Οι συναρτήσεις που προσφέρονται για διεργασίες είναι οι ακόλουθες:

    - **Adding** Επιστρέφει το άθροισμα όλων των εισερχόμενων τιμών.
    - **Adding Sign** Επιστρέφει την τιμή (1) αν το άθροισμα όλων των εισερχόμενων τιμών είναι μεγαλύτερο του μηδέν (0). Αλλιώς, επιστρέφει μηδέν (0).
    - **Average** Επιστρέφει το μέσο όρο όλων των εισερχόμενων τιμών.
    - **Maximum** Επιστρέφει την μέγιστη τιμή όλων των εισερχόμενων τιμών.
    - **Minimum** Επιστρέφει την ελάχιστη τιμή όλων των εισερχόμενων τιμών.

- **Thread i** Παίρνει όλες τις εισερχόμενες τιμές και επιστρέφει την τιμή του νήματος **i.** Η τιμή του **i** καθορίζεται από τον χρήστη.

✓ **Εφαρμογές (applications)**

Οι συναρτήσεις που προσφέρονται για εφαρμογές είναι οι ακόλουθες:

- **Adding** Επιστρέφει το άθροισμα όλων των εισερχόμενων τιμών.
- **Adding Sign** Επιστρέφει την τιμή (1) αν το άθροισμα όλων των εισερχόμενων τιμών είναι μεγαλύτερο του μηδέν (0). Αλλιώς, επιστρέφει μηδέν (0).
- **Average** Επιστρέφει το μέσο όρο όλων των εισερχόμενων τιμών.
- **Maximum** Επιστρέφει την μέγιστη τιμή όλων των εισερχόμενων τιμών.
- **Minimum** Επιστρέφει την ελάχιστη τιμή όλων των εισερχόμενων τιμών.
- **Add Tasks** Επιστρέφει το άθροισμα των τιμών όσων διεργασιών (tasks) έχουν επιλεχτεί από το χρήστη.

✓ **Workload**

Οι συναρτήσεις που προσφέρονται για workload είναι οι ακόλουθες:

- **Adding** Επιστρέφει το άθροισμα όλων των εισερχόμενων τιμών.
- **Adding Sign** Επιστρέφει την τιμή (1) αν το άθροισμα όλων των εισερχόμενων τιμών είναι μεγαλύτερο του μηδέν (0). Αλλιώς, επιστρέφει μηδέν (0).
- **Average** Επιστρέφει το μέσο όρο όλων των εισερχόμενων τιμών.
- **Maximum** Επιστρέφει την μέγιστη τιμή όλων των εισερχόμενων τιμών.
- **Minimum** Επιστρέφει την ελάχιστη τιμή όλων των εισερχόμενων τιμών.

- **Επίπεδο Resource Model**

  Στο επίπεδο αυτό οι συναρτήσεις μπορούν να χωριστούν σε τρία μέρη. Στις συναρτήσεις που αφορούν επεξεργαστές (cpu), κόμβους (node) και σύστημα (system).

  - ✓ **Επεξεργαστές (cpu)**

    Οι συναρτήσεις που προσφέρονται για επεξεργαστές είναι οι ακόλουθες:

    - **Active Thd** Επιστρέφει αυτούσια την τιμή του ενεργού νήματος. Αν δεν εκτελείται κάποιο νήμα στον επεξεργαστή επιστρέφει τη τιμή μηδέν (0).
    - **Active Thd Sign** Επιστρέφει την τιμή (1) αν η τιμή του νήματος είναι μεγαλύτερη του μηδέν (0). Αλλιώς, επιστρέφει μηδέν (0).
    - **Active Thd Val** Παίρνει την τιμή του νήματος και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης την επιστρέφει αυτούσια, χωρίς να την αλλάξει. Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή μηδέν (0).
    - **Active Thd Val Sign** Παίρνει την τιμή του νήματος και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή (1). Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή μηδέν (0).

  - ✓ **Κόμβους (node)**

    Οι συναρτήσεις που προσφέρονται για κόμβους είναι οι ακόλουθες:

    - **Adding** Επιστρέφει το άθροισμα όλων των εισερχόμενων τιμών.
    - **Adding Sign** Επιστρέφει την τιμή (1) αν το άθροισμα όλων των εισερχόμενων τιμών είναι μεγαλύτερο του μηδέν (0). Αλλιώς, επιστρέφει μηδέν (0).
    - **Average** Επιστρέφει το μέσο όρο όλων των εισερχόμενων τιμών.
    - **Maximum** Επιστρέφει την μέγιστη τιμή όλων των εισερχόμενων τιμών.
    - **Minimum** Επιστρέφει την ελάχιστη τιμή όλων των εισερχόμενων τιμών.

- ✓ **Σύστημα (system)**

  Οι συναρτήσεις που προσφέρονται για σύστημα είναι οι ακόλουθες:

  - **Adding** Επιστρέφει το άθροισμα όλων των εισερχόμενων τιμών.
  - **Adding Sign** Επιστρέφει την τιμή (1) αν το άθροισμα όλων των εισερχόμενων τιμών είναι μεγαλύτερο του μηδέν (0). Αλλιώς, επιστρέφει μηδέν (0).
  - **Average** Επιστρέφει το μέσο όρο όλων των εισερχόμενων τιμών.
  - **Maximum** Επιστρέφει την μέγιστη τιμή όλων των εισερχόμενων τιμών.
  - **Minimum** Επιστρέφει την ελάχιστη τιμή όλων των εισερχόμενων τιμών.

- **Επίπεδο COMPOSE**

  Το επίπεδο αυτό βρίσκεται πάνω από τα επίπεδα που περιέγραψα προηγουμένως. Οι συναρτήσεις αυτού του επιπέδου εφαρμόζονται μετά που θα υπολογιστούν οι συναρτήσεις των κατώτερων επιπέδων. Πρώτα εφαρμόζεται η συνάρτηση που έχει επιλεχτεί στο "Compose 2" και έπειτα η συνάρτηση του "Compose 1". Τα "Compose 1" και "Compose 2" προσφέρουν ακριβώς τις ίδιες συναρτήσεις. Με συνδυασμό αυτών των δύο μπορούμε να πάρουμε το επιθυμητό αποτέλεσμα. Οι συναρτήσεις που προσφέρονται στο "Compose 1" και "Compose 2" είναι οι ακόλουθες:

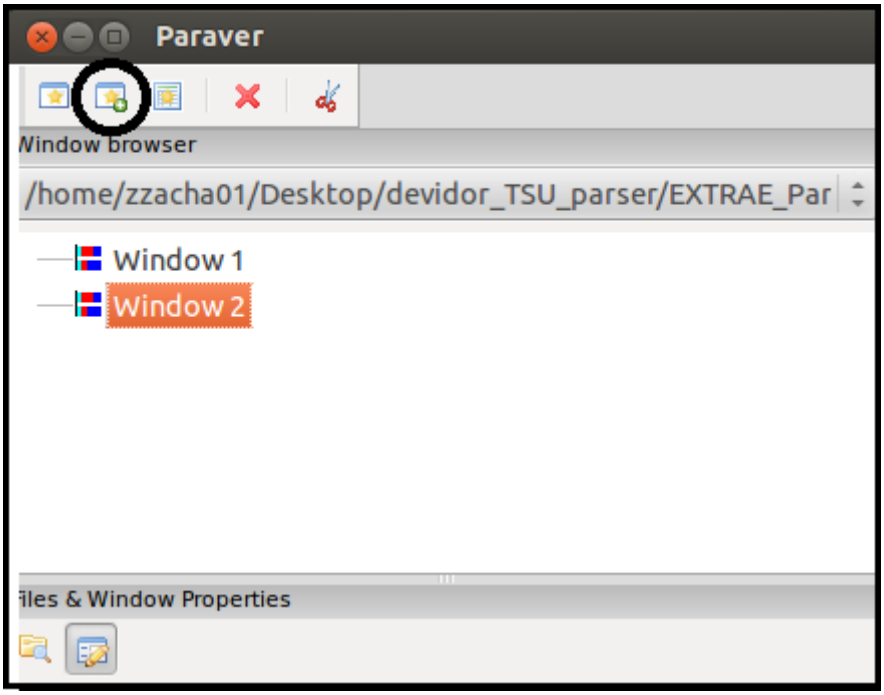
  - ✓ **As Is** Επιστρέφει αυτούσια την τιμή του προηγούμενου επιπέδου, χωρίς να την αλλάξει.
  - ✓ **Sign** Επιστρέφει την τιμή (1) αν η τιμή του προηγούμενου επιπέδου είναι μεγαλύτερη του μηδέν (0). Αλλιώς, επιστρέφει μηδέν (0).
  - ✓ **1-Sign** Συμπληρωματική συνάρτηση του Sign. Επιστρέφει την τιμή μηδέν (0) αν η τιμή του προηγούμενου επιπέδου είναι μεγαλύτερη του μηδέν (0). Αλλιώς, επιστρέφει (1).
  - ✓ **Mod** Επιστρέφει το υπόλοιπο που προκύπτει από την διαίρεση της τιμής του προηγούμενου επιπέδου με μια τιμή που καθορίζει ο χρήστης (divider). Η τιμή επιστροφής κυμαίνεται από [0 – (divider-1)].
  - ✓ **Mod+1** Παρόμοιο με το Mod, απλά προσθέτει την μονάδα (1) στην τιμή που επιστρέφει. Η τιμή επιστροφής κυμαίνεται από [1 - divider].

- ✓ **Div** Επιστρέφει το πηλίκο που προκύπτει από την διαίρεση της τιμής του προηγούμενου επιπέδου με μια τιμή που καθορίζει ο χρήστης (divider).
- ✓ **Select Range** Παίρνει την τιμή του προηγούμενου επιπέδου και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης την επιστρέφει αυτούσια, χωρίς να την αλλάξει. Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή μηδέν (0).
- ✓ **Is In Range** Παίρνει την τιμή του προηγούμενου επιπέδου και αν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή (1). Αν δεν είναι μέσα στο σύνολο τιμών που καθόρισε ο χρήστης επιστρέφει την τιμή μηδέν (0).

Όπως ανέφερα στην αρχή αυτού του Μέρους, το Semantic Module μας παρέχει μια πολύ μεγάλη δυνατότητα. Αυτή η δυνατότητα ονομάζεται "Derived Window" και προκύπτει από τον συνδυασμό δύο διαφορετικών όψεων του ίδιου trace αρχείου. Με τον συνδυασμό αυτό μπορούμε να πάρουμε πληροφορίες που δεν μπορούσαμε να πάρουμε κατευθείαν από την ανάλυση ενός "Display Window". Για παράδειγμα, αν έχουμε μια όψη που μετρά τις εντολές και μια άλλη που μετρά τους κύκλους (cycles), εφαρμόζοντας την πράξη διαίρεσης ανάμεσα στις δύο αυτές όψεις προκύπτει η πληροφορία εντολές ανά κύκλο (Instructions/Cycle). Για να το πετύχουμε αυτό πρέπει να ακολουθήσουμε κάποια βήματα.

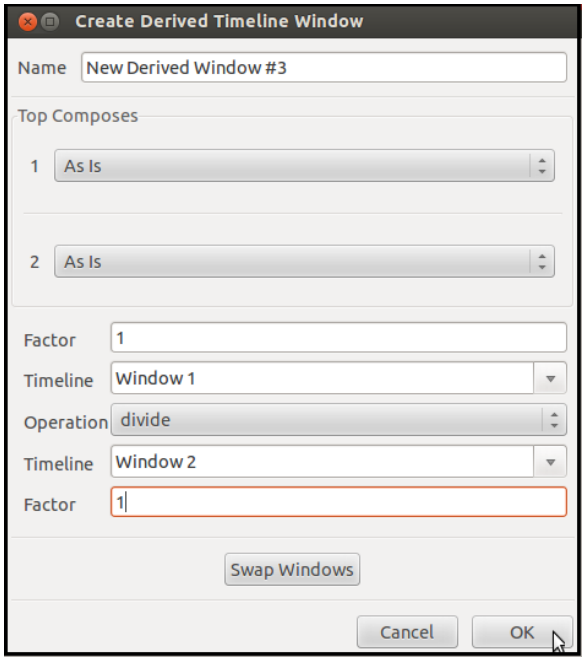
Αρχικά, πρέπει να ανοίξουμε δύο Display Windows, το "Window 1" και το "Window 2". Στο Filter Module για το "Window 1" θα επιλέξουμε να παρουσιάζονται μόνο όσα event αφορούν εντολές (Instructions) και στο Filter Module για το "Window 2" θα επιλέξουμε να παρουσιάζονται μόνο όσα event αφορούν κύκλους (Cycles). Ακολούθως, θα επιλέξουμε και στα δύο Windows την συνάρτηση "Next Event Value" του Semantic Module.

Αφού κάνουμε τα πιο πάνω, ανοίγουμε το "Derived Window" πατώντας πάνω στο εικονίδιο που είναι κυκλωμένο στην Εικόνα 6.7.10.



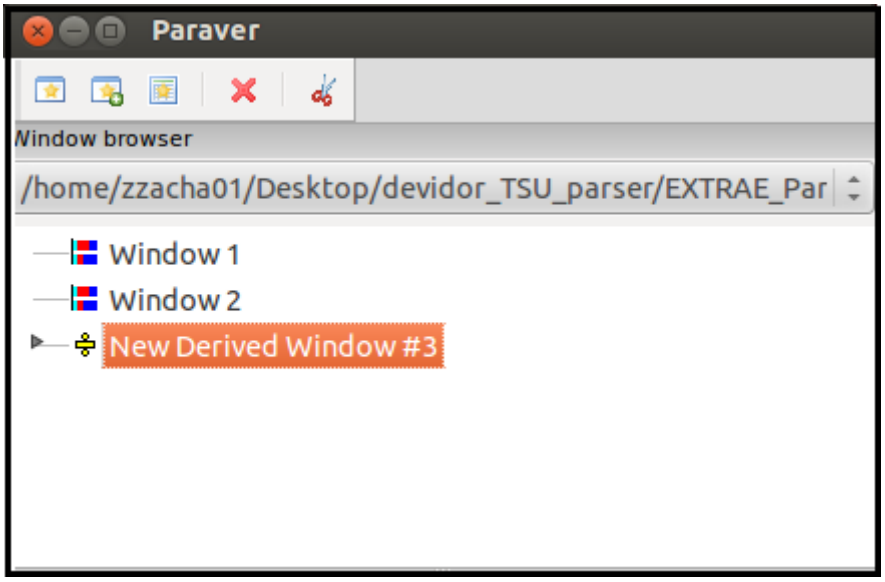
Εικόνα 6.7.10

Στο "Derived Window" που ανοίξαμε επιλέγουμε τις δύο όψεις, ανάμεσα στις οποίες θέλουμε να εφαρμόσουμε την πράξη. Όπως βλέπεται στην Εικόνα 6.7.11, η πάνω όψη αφορά το "Window 1" (Instructions) και η κάτω όψη το "Window 2" (Cycles).



Εικόνα 6.7.11

Ακολούθως, επιλέγουμε την πράξη διαίρεσης (divide) και αναθέτουμε την τιμή (1) στα πεδία “factor”. Τέλος, πατώντας το “OK” προκύπτει το “Derived Window” της Εικόνας 6.7.12.

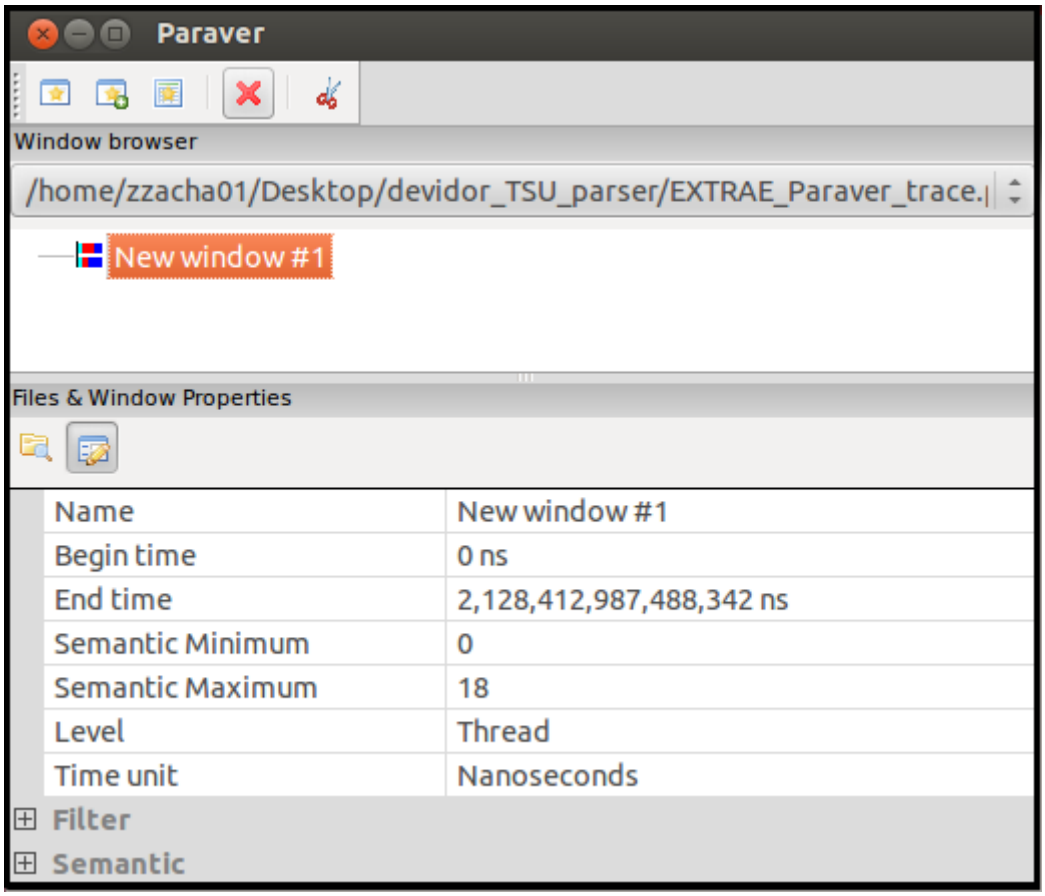


Εικόνα 6.7.12

6.8 Visualization Module

Στο Μέρος αυτό θα γίνει μια περιγραφή του Visualization Module, ή Display Window, το οποίο αποτελεί την πρώτη από τις δύο μονάδες αναπαράστασης που προσφέρει το PARAVER. Για να ανοίξουμε ένα Display Window, πατάμε πάνω στο εικονίδιο “Display Window” της Εικόνας 6.5.1, αφού πρώτα έχουμε φορτώσει ένα trace αρχείο.

Αφού ανοίξουμε το Display Window, το PARAVER μας παρέχει αρχικά κάποιες δυνατότητες για ρύθμιση του τρόπου εμφάνισής του (Εικόνα 6.8.1).



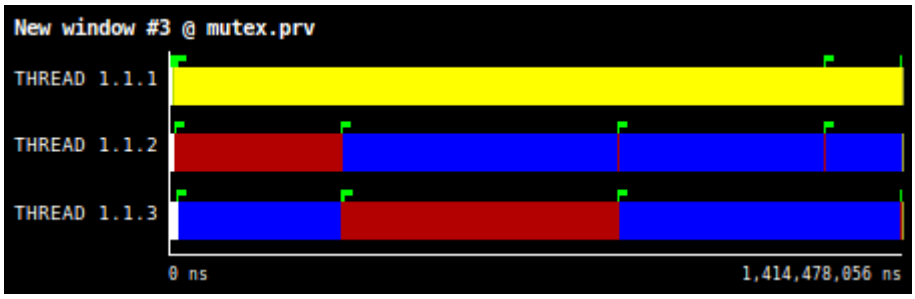
Εικόνα 6.8.1

Η ρύθμιση του τρόπου εμφάνισής του γίνεται μέσω των πιο κάτω πεδίων:

- **Name** Το όνομα του Display Window.
- **Begin Time** Η χρονική στιγμή από την οποία θα ξεκινά η προβολή του trace.
- **End Time** Η χρονική στιγμή στην οποία θα σταματά η προβολή του trace.
- **Semantic Minimum** Η ελάχιστη semantic τιμή που θα προβάλλει το PARAVER.

- **Semantic Maximum** Η μέγιστη semantic τιμή που θα προβάλλει το PARAVER.
- **Level** Το επίπεδο ανάλυσης του trace (THREAD, TASK, NODE, SYSTEM κτλ).
- **Time Unit** Η μονάδα χρόνου που θα χρησιμοποιεί το PARAVER (seconds, milliseconds, nanoseconds κτλ).

Αφού επιλέξουμε τις ρυθμίσεις που επιθυμούμε για τον τρόπο εμφάνισης του Display Window, μεταφερόμαστε στο παράθυρο αυτό. Αρχικά, το Display Window έχει την μορφή που βλέπετε στην Εικόνα 6.8.2.

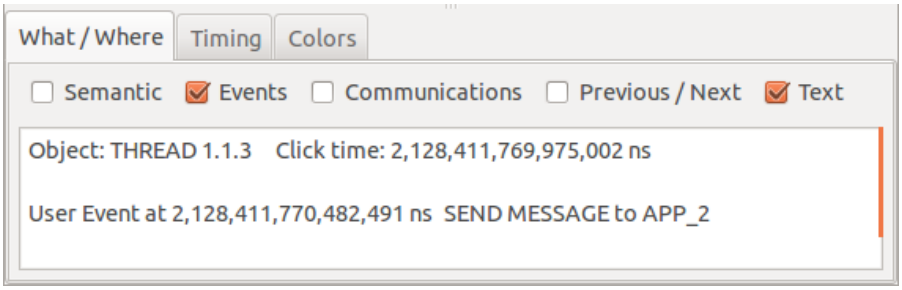


Εικόνα 6.8.2

Στο Display Window το PARAVER μας παρέχει αρκετές δυνατότητες κάνοντας αριστερό ή δεξί κλικ πάνω σε αυτό. Επίσης, όταν τοποθετήσουμε απλά τον δείκτη του mouse μας πάνω σε ένα αντικείμενο που προβάλλεται στο Display Window, το PARAVER μας εμφανίζει την πληροφορία για την semantic τιμή του αντικειμένου στην συγκεκριμένη χρονική στιγμή.

Οι λειτουργικότητες που μας προσφέρει το PARAVER κάνοντας αριστερό ή δεξί κλικ πάνω στο Display Window, είναι οι ακόλουθες:

- **Αριστερό κλικ**
  - ✓ **Μεγέθυνση (Zoom in)** Επιλέγοντας το μέρος του Display Window στο οποίο θέλουμε να επικεντρωθούμε.
  - ✓ **Εμφάνιση Info Panel** Κάνοντας διπλό αριστερό κλικ πάνω σε ένα αντικείμενο που προβάλλεται στο Display Window. Το Info Panel παρέχει πληροφορίες στον χρήστη για τα events, τις επικοινωνίες και την semantic τιμή του αντικειμένου κατά την χρονική στιγμή στην οποία έκανε το διπλό αριστερό κλικ (Εικόνα 6.8.3).

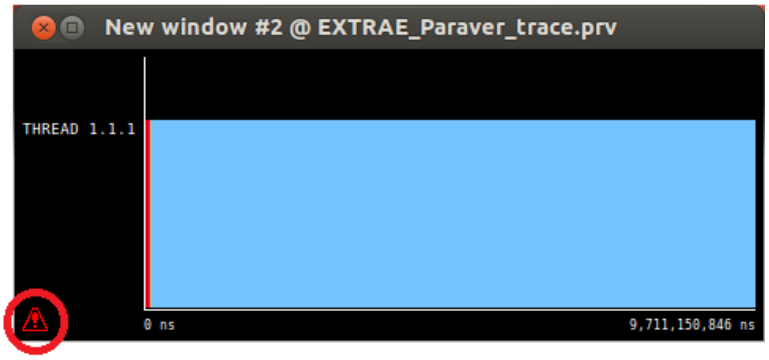


Εικόνα 6.8.3

- **Δεξί κλικ**

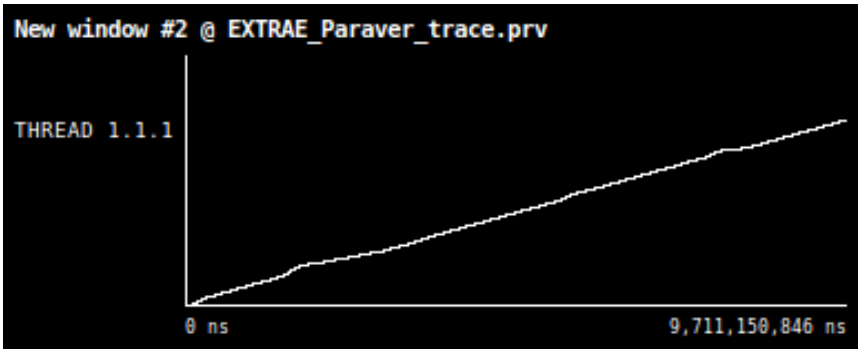
  Κάνοντας δεξί κλικ σε οποιαδήποτε περιοχή πάνω στο Display Window, εμφανίζεται ένα pop-up menu το οποίο μας παρέχει τις ακόλουθες δυνατότητες:

  - ✓ **Copy & Paste** Ο χρήστης μπορεί να αντιγράψει την κλίμακα χρόνου (Time Scale), την κλίμακα Semantic (Semantic Scale) και το μέγεθος (Size) ενός Display Window σε ένα άλλο.
  - ✓ **Clone** Δημιουργεί ένα πανομοιότυπο αντίγραφο του Display Window. Το αντίγραφο αυτό έχει τις ίδιες κλίμακες και τις ίδιες semantic συναρτήσεις που είχε το αρχικό Display Window.
  - ✓ **Undo Zoom** Αναιρεί την τελευταία μεγέθυνση (zoom in) που έκανε ο χρήστης στο Display Window.
  - ✓ **Fit Time Scale** Προσαρμογή του Display Window στην χρονική διάρκεια του trace αρχείου. Χρήσιμο στην περίπτωση που ο χρήστης έχει μεγεθύνει σε μεγάλο βαθμό μια περιοχή του trace και επιθυμεί να μεταβεί στην αρχική εικόνα του trace με μόνο ένα βήμα.
  - ✓ **Fit Semantic Scale** Για υπολογισμό της νέας ανώτατης και κατώτατης semantic τιμής. Αυτό είναι χρήσιμο στην περίπτωση που οι semantic συναρτήσεις που επιλέξαμε επιστρέφουν τιμές που είναι εκτός του Semantic Scale που ορίσαμε αρχικά. Σε μια τέτοια περίπτωση το PARAVER εμφανίζει το εικονίδιο που είναι κυκλωμένο στην Εικόνα 6.8.4, το οποίο υποδεικνύει πως συνέβηκε κάποιο σφάλμα. Κάνοντας κλικ πάνω σε αυτό το εικονίδιο, εμφανίζεται ένα μήνυμα με την περιγραφή του σφάλματος.



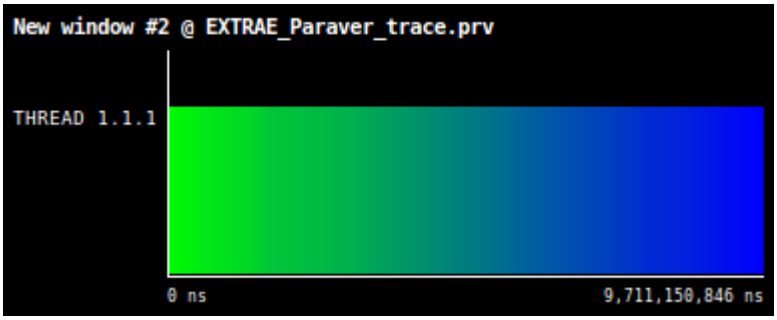
Εικόνα 6.8.4

- ✓ **View** Χωρίζεται σε τρεις επιλογές τις οποίες μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε ανά πάσα στιγμή. Οι επιλογές αυτές είναι:
  - **Communication Lines** Με ενεργοποιημένη αυτή την επιλογή, το PARAVER ζωγραφίζει γραμμές μεταξύ των αντικειμένων που επικοινωνούν μεταξύ τους. Η γραμμή έχει κίτρινο χρώμα αν αφορά λογική (logical) επικοινωνία και κόκκινο χρώμα αν αφορά φυσική (physical) επικοινωνία.
  - **Event Flags** Με ενεργοποιημένη αυτή την επιλογή, το PARAVER ζωγραφίζει από ένα σημαιάκι σε κάθε σημείο που υπάρχει event.
  - **Function Line With Color** Με ενεργοποιημένη αυτή την επιλογή, το PARAVER ζωγραφίζει τα αντικείμενα χρησιμοποιώντας χρώματα. Ένα τέτοιο παράδειγμα αποτελεί η Εικόνα 6.8.2. Αν η επιλογή αυτή είναι απενεργοποιημένη, το PARAVER ζωγραφίζει τα αντικείμενα σε μορφή timeline (Εικόνα 6.8.5). Αυτή η μορφή είναι χρήσιμη σε περίπτωση που θέλουμε να δούμε την εξέλιξη της τιμής μιας μεταβλητής στον χρόνο, ή αν έχουμε μεγάλο φάσμα τιμών.



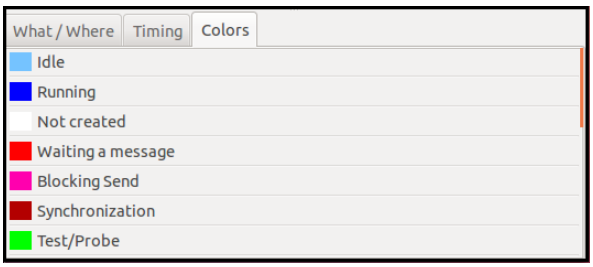
Εικόνα 6.8.5

- ✓ **Color** Χωρίζεται σε δύο επιλογές από τις οποίες μόνο μία μπορούμε να επιλέξουμε ανά πάσα στιγμή. Οι επιλογές αυτές είναι:
  - **Code Color** Ζωγραφίζει τα αντικείμενα με τα χρώματα που έχει καθορίσει ο χρήστης στο *.pcf αρχείο (Εικόνα 6.8.2).
  - **Gradient Color** Ζωγραφίζει τα αντικείμενα βάσει μιας διαβάθμισης χρωμάτων. Μεγαλύτερες τιμές ζωγραφίζονται με σκούρο χρώμα (μπλε) και μικρότερες τιμές με ανοιχτό χρώμα (πράσινο). Στην Εικόνα 6.8.6 βλέπεται την εμφάνιση που θα έχει το trace αρχείο της Εικόνας 6.8.5 αν εφαρμοστούν σε αυτό οι επιλογές "Function Line With Color" και "Gradient Color".



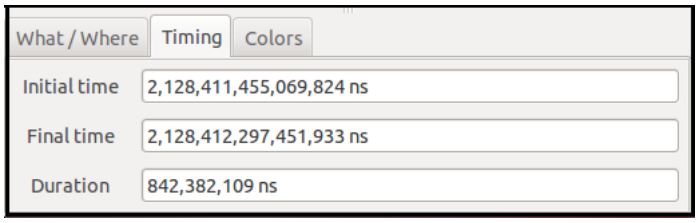
Εικόνα 6.8.6

- ✓ **Pixel Size** Ο χρήστης μπορεί να μεγαλώσει ή να μικρύνει το μέγεθος των pixels που χρησιμοποιεί το Display Window για αναπαράσταση του trace αρχείου.
- ✓ **Select Rows** Ο χρήστης μπορεί να επιλέξει ποια αντικείμενα επιθυμεί να παρουσιάσει το Display Window και ποια όχι.
- ✓ **Save Image** Αποθήκευση της εικόνας που αναπαριστά το Display Window.
- ✓ **Info Panel** Εμφάνιση ή απόκρυψη του Info Panel, το οποίο παρέχει χρήσιμες πληροφορίες στον χρήστη. Το Info Panel αποτελείται από τρία tabs:
  - **Colors** Έχει τα χρώματα και την ετικέτα που εμφανίζει το PARAVER για κάθε ένα από αυτά (Εικόνα 6.8.7).



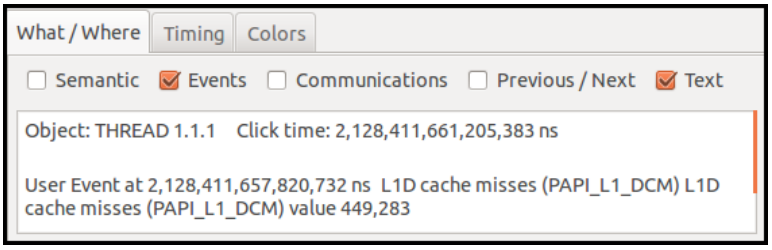
Εικόνα 6.8.7

- **Timing** Σε περίπτωση που ο χρήστης μεγεθύνει (zoom in) ένα μέρος του Display Window, στο tab αυτό μπορεί να δει σε ποια χρονική στιγμή ξεκινά το μέρος που μεγέθυνε (Initial time), σε ποια χρονική στιγμή σταματά (Final time) και την χρονική διάρκεια μεταξύ των δύο (Duration). Μπορείτε να δείτε ένα παράδειγμα στην Εικόνα 6.8.8.

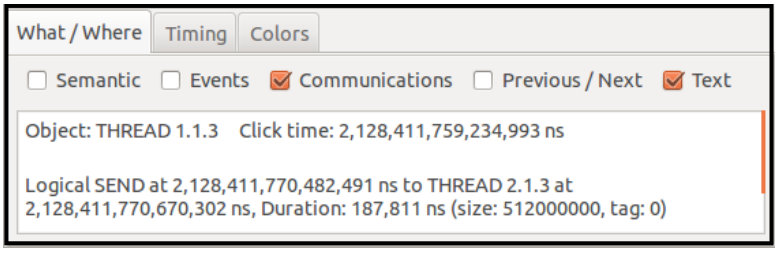


Εικόνα 6.8.8

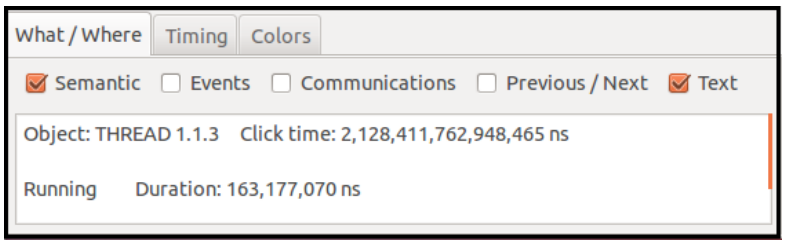
- **What/Where** Παρέχει πληροφορίες στον χρήστη για τα events (Εικόνα 6.8.9), τις επικοινωνίες (Εικόνα 6.8.10) και την semantic τιμή (Εικόνα 6.8.11) ενός αντικειμένου, κατά την χρονική στιγμή στην οποία έκανε το διπλό αριστερό κλικ.



Εικόνα 6.8.9

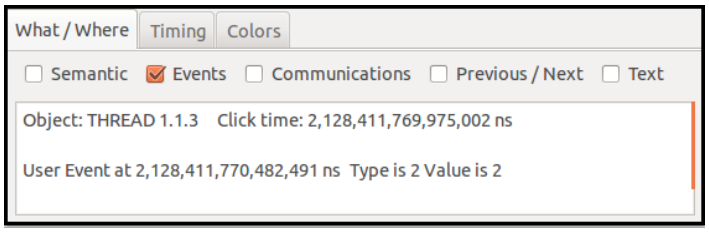


Εικόνα 6.8.10

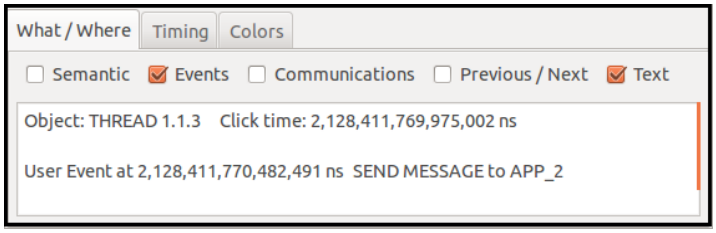


Εικόνα 6.8.11

Όπως μπορείτε να δείτε από τις Εικόνες 6.8.9, 6.8.10 και 6.8.11 η πληροφορία που εμφανίζεται μέσα στο tab What/Where «φιλτράρεται» με βάση κάποια check boxes (Semantic, Events, Communications, Previous/Next και Text). Αν για παράδειγμα δεν επιλέγαμε το Semantic check box, το tab What/Where δεν θα εμφάνιζε την Semantic τιμή.



Εικόνα 6.8.12



Εικόνα 6.8.13

Στην Εικόνα 6.8.12 βλέπετε μια περίπτωση όπου το Text check box δεν έχει επιλεχτεί, με αποτέλεσμα το tab What/Where να παρουσιάζει αυτούσιο τον τύπο (2) και την τιμή (2) ενός event. Όταν όμως το Text check box επιλεχτεί (Εικόνα 6.8.13), το tab What/Where μετατρέπει τον τύπο και την τιμή του event στα labels που έχει καθορίσει ο χρήστης μέσα στο *.pcf αρχείο.

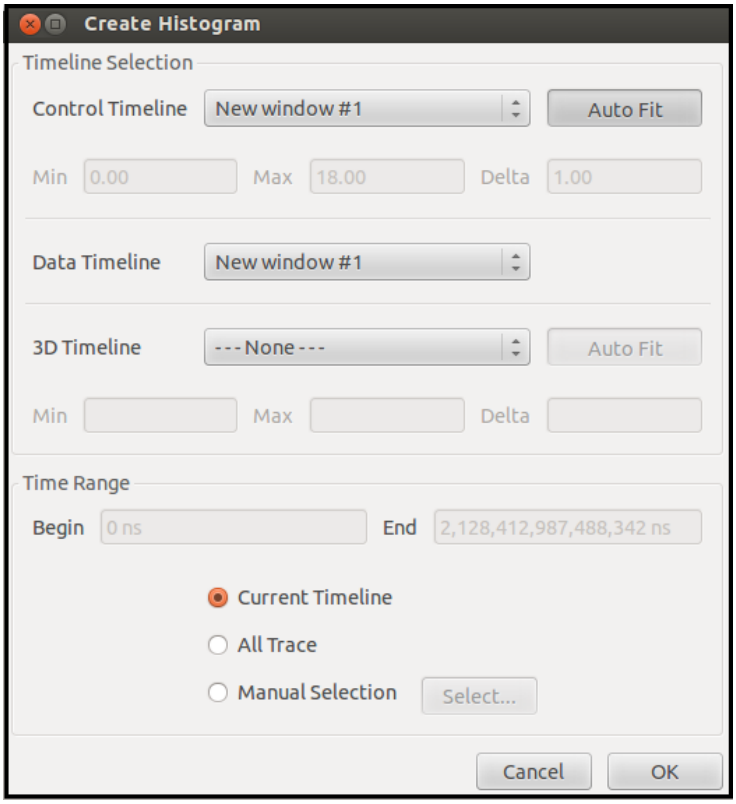
Όταν ο χρήστης πάρει την επιθυμητή εικόνα ανάλυσης του trace αρχείου του, έχοντας κάνει όλες τις απαραίτητες ρυθμίσεις στα Filter και Semantic Modules, μπορεί να αποθηκεύσει αυτές τις ρυθμίσεις σε ένα configuration (*.cfg) αρχείο, επιλέγοντας “Save Configuration” στο μενού “File”. Με αυτό τον τρόπο, αν χρειαστεί ποτέ να εφαρμόσει στο ίδιο ή διαφορετικό trace αρχείο τις ίδιες ρυθμίσεις, δεν θα πρέπει να τις ορίσει ξανά. Απλά θα φορτώσει το συγκεκριμένο configuration (*.cfg) αρχείο, επιλέγοντας “Load Configuration” στο μενού “File”. Αυτή η διαδικασία είναι πολύ χρήσιμη, ιδιαίτερα στην περίπτωση των Derived Windows.

6.9 Analyzer Module

Έχοντας διαβάσει κάποιος μέχρι το σημείο αυτό, θα είναι πλέον σε θέση να δημιουργήσει ένα trace αρχείο με το EXTRAE, να το ανοίξει με το PARAVER και αφού εφαρμόσει τις κατάλληλες ρυθμίσεις στα Filter και Semantic Modules, να πάρει την επιθυμητή όψη.

Σε αυτό το Μέρος θα πάω ένα βήμα παρακάτω και θα προσπαθήσω να περιγράψω τον τρόπο με τον οποίο ο χρήστης μπορεί να πάρει κάποια στατιστικά στοιχεία για την όψη του trace αρχείου που έχει δημιουργήσει. Για τη συλλογή στατιστικών στοιχείων στο PARAVER υπεύθυνο είναι το Analyzer Module – Histogram.

Για να χρησιμοποιήσει ωστόσο ο χρήστης το Analyzer Module – Histogram, θα πρέπει να έχει ήδη δημιουργήσει ένα Visualization Module - Display Window, με την όψη που επιθυμεί. Εφόσον η όψη για την οποία θέλει να πάρει κάποια στατιστικά στοιχεία είναι έτοιμη, ο χρήστης ανοίγει ένα Histogram πατώντας πάνω στο εικονίδιο “Histogram” που είναι κυκλωμένο στην Εικόνα 6.5.1. Αυτό θα έχει ως αποτέλεσμα να ανοιχτεί το παράθυρο “Create Histogram” στην οθόνη (Εικόνα 6.9.1).

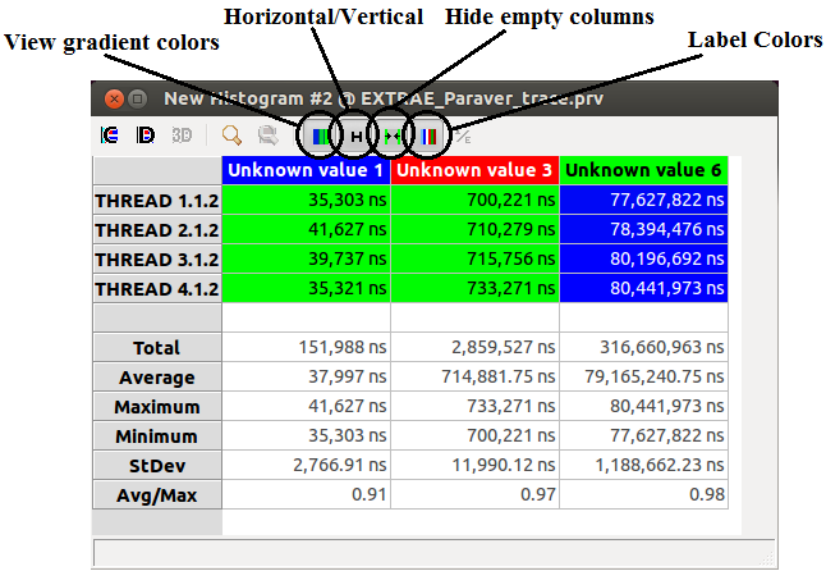


Εικόνα 6.9.1

Στο παράθυρο αυτό ο χρήστης κάνει τις ακόλουθες επιλογές:

- **Control Timeline** Εδώ ο χρήστης επιλέγει το Display Window – Timeline στο οποίο θα γίνει η στατιστική ανάλυση.
- **Data Timeline** Εδώ ο χρήστης επιλέγει το Display Window – Timeline από το οποίο θα ληφθούν τα δεδομένα προς ανάλυση.
- **Time Range** Εδώ ο χρήστης επιλέγει το χρονικό διάστημα για το οποίο θέλει να πάρει στατιστικές μετρήσεις. Μπορεί να επιλέξει μεταξύ τριών επιλογών:
    - ✓ **Current Timeline** Όση και η χρονική διάρκεια της όψης στο Display Window – Timeline.
    - ✓ **All Trace** Όση και η χρονική διάρκεια ολόκληρου του trace αρχείου.
    - ✓ **Manual Selection** Μπορεί ο ίδιος ο χρήστης να καθορίσει το χρονικό διάστημα. Για αυτό τον σκοπό υπάρχουν δύο επιλογές:
        - Βάζοντας την αρχική τιμή στο πεδίο “Begin” και την τελική τιμή στο πεδίο “End”.
        - Πατώντας πάνω στην επιλογή “Select”, με την οποία μπορεί να επιλέξει το μέρος του Display Window – Timeline που θέλει να αναλύσει, χρησιμοποιώντας το mouse.

Στο παράθυρο αυτό υπάρχει και η επιλογή “Auto Fit”, η οποία καθορίζει αυτόματα τις τιμές για τα πεδία “Min”, “Max” και “Delta”, με βάση τις πληροφορίες που παίρνει από το Display Window – Timeline. Αφού ο χρήστης κάνει τις πιο πάνω επιλογές, επιλέγει “OK” και εμφανίζεται το Histogram το οποίο περιέχει τις στατιστικές μετρήσεις (Εικόνα 6.9.2).



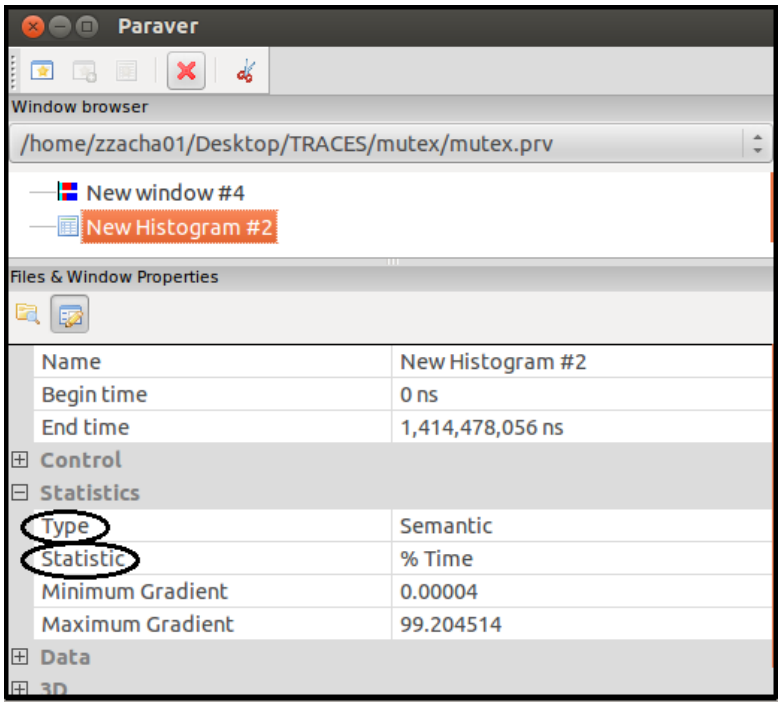
	Unknown value 1	Unknown value 3	Unknown value 6
THREAD 1.1.2	35,303 ns	700,221 ns	77,627,822 ns
THREAD 2.1.2	41,627 ns	710,279 ns	78,394,476 ns
THREAD 3.1.2	39,737 ns	715,756 ns	80,196,692 ns
THREAD 4.1.2	35,321 ns	733,271 ns	80,441,973 ns
Total	151,988 ns	2,859,527 ns	316,660,963 ns
Average	37,997 ns	714,881.75 ns	79,165,240.75 ns
Maximum	41,627 ns	733,271 ns	80,441,973 ns
Minimum	35,303 ns	700,221 ns	77,627,822 ns
StDev	2,766.91 ns	11,990.12 ns	1,188,662.23 ns
Avg/Max	0.91	0.97	0.98

Εικόνα 6.9.2

Στο παράθυρο Histogram, όπως βλέπετε και από την Εικόνα 6.9.2, υπάρχουν τέσσερα κουμπιά τα οποία εξυπηρετούν κάποιες λειτουργικότητες. Αυτά τα κουμπιά και οι λειτουργικότητές τους είναι:

- **View gradient colors** Προσθέτει χρώμα στα πεδία που περιέχουν τα αποτελέσματα των στατιστικών μετρήσεων. Σκούρο (μπλε) χρώμα για τις μεγάλες τιμές και ανοιχτό (πράσινο) χρώμα για τις μικρές τιμές.
- **Horizontal/Vertical** Εναλλαγή των γραμμών με τις στήλες. Όταν είναι επιλεγμένο τα αντικείμενα του PARAVER (threads, tasks, κτλ) εμφανίζονται στις γραμμές και τα ονόματα των στατιστικών μετρήσεων στις στήλες.
- **Hide empty columns** Αποκρύπτει όσες στήλες είναι άδειες.
- **Label Colors** Προσθέτει χρώμα στις ετικέτες (labels) με τα ονόματα των στατιστικών μετρήσεων.

Τέλος, όπως και τα υπόλοιπα modules (Semantic και Filter), έτσι και το Analyzer Module προσφέρει κάποιες στατιστικές συναρτήσεις. Οι συναρτήσεις αυτές χωρίζονται σε δύο τύπους. Ανάλογα με τον τύπο που θα επιλέξει ο χρήστης, το PARAVER του προσφέρει και τις ανάλογες συναρτήσεις. Η επιλογή τύπου και συνάρτησης γίνεται στο κυρίως παράθυρο του PARAVER (Εικόνα 6.9.3).



Εικόνα 6.9.3

Οι δύο τύποι συναρτήσεων που προσφέρει το Analyzer Module και οι συναρτήσεις αυτών, είναι:

- **Τύπος Semantic**
  - ✓ **Time** Υπολογίζει το συνολικό χρόνο που διαρκεί μια τιμή στο επιλεγμένο χρονικό διάστημα.
  - ✓ **% Time** Υπολογίζει το ποσοστό του συνολικού χρόνου που διαρκεί μια τιμή στο επιλεγμένο χρονικό διάστημα.
- **Τύπος Communication**
  - ✓ **# Sends** Υπολογίζει το άθροισμα των εξερχόμενων επικοινωνιών στο επιλεγμένο χρονικό διάστημα.
  - ✓ **# Receives** Υπολογίζει το άθροισμα των εισερχόμενων επικοινωνιών στο επιλεγμένο χρονικό διάστημα.
  - ✓ **Bytes sent** Υπολογίζει το άθροισμα των bytes που στάλισαν στο επιλεγμένο χρονικό διάστημα.
  - ✓ **Bytes received** Υπολογίζει το άθροισμα των bytes που παραλήφθηκαν στο επιλεγμένο χρονικό διάστημα.

Κεφάλαιο 7

Μεθοδολογία ανάλυσης DDM εφαρμογών

7.1 Προσθήκη EXTRAE συναρτήσεων στην DDM εφαρμογή.
7.2 Προσθήκη EXTRAE συναρτήσεων στον κώδικα του TSU.

7.1 Προσθήκη EXTRAE συναρτήσεων στην DDM εφαρμογή

Για καταγραφή της συμπεριφοράς των Worker threads τα οποία είναι υπεύθυνα για την εκτέλεση των διάφορων DDM εφαρμογών, ο προγραμματιστής πρέπει να τοποθετήσει κάποιες συναρτήσεις του EXTRAE API μέσα στον κώδικα της DDM εφαρμογής. Τα βήματα που ακολούθησα είναι τα εξής:

- **Συνάρτηση main()**
  Μέσα στην συνάρτηση main της DDM εφαρμογής τοποθετώ τις δύο συναρτήσεις του EXTRAE API, Extrae_init() και Extrae_fini(). Οι συναρτήσεις αυτές δηλώνουν πως θέλω να καταγράψω στο trace αρχείο που θα παραχθεί από την εκτέλεση της DDM εφαρμογής με το εργαλείο EXTRAE όλα τα γεγονότα, τις επικοινωνίες και τις καταστάσεις που συμβαίνουν μέσα στην εφαρμογή.
  Όπως βλέπετε και από την Εικόνα 7.1.1 το Extrae_init() τοποθετείται εντελώς στην αρχή της main και το Extrae_fini() στο τέλος της, πριν την εντολή return.

```
int main(int argc, char **argv)
{
      Extrae_init();
{
      myIP = (char *) malloc(sizeof(char)*24);
      myName = (char *) malloc(sizeof(char)*24);
       ◆
       ◆
       ◆
       ◆
      free(A);
      free(B);
      free(C);

      Extrae_fini();

      return 1 ;
}
```

Εικόνα 7.1.1

- **Συνάρτηση threads_main()**

  Μέσα στην συνάρτηση threads_main της DDM εφαρμογής όπου υπάρχει κώδικας που εσωκλείεται μέσα σε DVM_THREAD_START και DVM_THREAD_END, τοποθετώ από δύο φορές τη συνάρτηση Extrae_event του EXTRAE API. Η πρώτη μπαίνει ακριβώς μετά το DVM_THREAD_START και η δεύτερη ακριβώς πριν το DVM_THREAD_END (Εικόνα 7.1.2).

```
DVM_THREAD_START(THREAD_FRONT);
Extrae_event(24,4);
#ifdef STARTING_TIME
      if (counters[speid][2]==0){
            counters[speid][2] = gtod_micro();
      }
#endif
      GET_CONTEXT_D(DVM_CONTEXT,cntx,indx);

      pA  = (DATA_TYPE *)DVM_LOOKUP(0);
      pB  = (DATA_TYPE *)DVM_LOOKUP(1);

      //fwd((float*)A[cntx][cntx],(float*)A[cnt
      fwd(pA,pB);

      DVM_UPDATE_MULTIPLE_FIN(CONS_1,MAKE_CONTE
(cntx,DIM-1,indx));
Extrae_event(24,0);
DVM_THREAD_END();
```

Εικόνα 7.1.2

Ως πρώτη παράμετρο (type) του Extrae_event βάζω πάντα την ίδια τιμή. Αυτό δηλώνει πως όσα events καταγράψει το EXTRAE με αυτή την τιμή type, αφορούν τα Worker threads της DDM εφαρμογής (στην Εικόνα 7.1.2 η τιμή αυτή είναι 24). Ως δεύτερη παράμετρο (value) του Extrae_event βάζω πάντα την ίδια τιμή αν πρόκειται για το Extrae_event που βρίσκεται πριν το DVM_THREAD_END. Με αυτό τον τρόπο, το PARAVER όταν το Worker thread εκτελεί κώδικα που δεν βρίσκεται μεταξύ των DVM_THREAD_START και DVM_THREAD_END (άρα ο Worker δεν έχει κάτι να κάνει και αναμένει το TSU να του δώσει εργασία) θα ζωγραφίζει το Worker thread με το ίδιο χρώμα. Στο παράδειγμα της Εικόνας 7.1.2 η τιμή αυτή είναι μηδέν (0), και έτσι το PARAVER όταν το Worker thread δεν έχει κάτι να κάνει και αναμένει το TSU να του δώσει εργασία θα ζωγραφίζει με γαλάζιο το Worker thread. Για περισσότερες πληροφορίες σχετικά με τα χρώματα που προσφέρει το PARAVER μπορείτε να διαβάσετε το **Κεφάλαιο 4 – Τμήμα "STATES_COLOR"** (σελ. 45).

Όταν πρόκειται για το Extrae_event που βρίσκεται ακριβώς μετά το DVM_THREAD_START, ως δεύτερη παράμετρο (value) του Extrae_event βάζω μια διαφορετική τιμή κάθε φορά. Με αυτό τον τρόπο, όταν το PARAVER θα εμφανίσει το Worker thread θα μπορώ να διακρίνω ανάλογα με το χρώμα που χρησιμοποιεί σε ποιο κομμάτι κώδικα DVM_THREAD_START και DVM_THREAD_END βρίσκεται ο Worker. Στο παράδειγμα της Εικόνας 7.1.2 όταν το Worker thread βρίσκεται μεταξύ του DVM_THREAD_START και DVM_THREAD_END κώδικα, το PARAVER θα εμφανίζει στο Worker thread την τιμή (4), η οποία αντιστοιχεί στο ροζ χρώμα.

7.2 Προσθήκη EXTRAE συναρτήσεων στον κώδικα του TSU

Στον κώδικα του TSU χρησιμοποίησα την ίδια λογική που χρησιμοποίησα και για τους κώδικες των DDM εφαρμογών, μόνο που αυτή την φορά αντί να «σημαδεύω» κώδικες μου εσωκλείονται μεταξύ DVM_THREAD_START και DVM_THREAD_END, «σημάδευα» τα σημεία του TSU κώδικα που αναφέρονταν σε σημαντικές κλήσεις συναρτήσεων-λειτουργειών του TSU. Για παράδειγμα, «σημάδεψα» συναρτήσεις-λειτουργίες όπως CacheFlow, DecrementConsumers και Dequeue, βάζοντας ένα Extrae_event πριν την κλήση της συνάρτησης και ένα Extrae_event μετά την κλήση της συνάρτησης (Εικόνα 7.2.1).

```
Extrae_event(20,7);
CacheFlow(nIndexTSU,nDCDSpace);
Extrae_event(20,0);
```

Εικόνα 7.2.1

Ως πρώτη παράμετρο (type) του Extrae_event βάζω πάντα την ίδια τιμή. Αυτό δηλώνει πως όσα events καταγράψει το EXTRAE με αυτή την τιμή type, αφορούν τα TSU threads (στην Εικόνα 7.2.1 η τιμή αυτή είναι 20). Ως δεύτερη παράμετρο (value) του Extrae_event βάζω πάντα την ίδια τιμή αν πρόκειται για το Extrae_event που βρίσκεται μετά την κλήση της συνάρτησης. Με αυτό τον τρόπο, το PARAVER θα ζωγραφίζει το TSU thread με το ίδιο χρώμα όταν το TSU thread εκτελεί κώδικα που βρίσκεται εκτός

των σημαντικών λειτουργιών-συναρτήσεών του. Στο παράδειγμα της Εικόνας 7.2.1 η τιμή αυτή είναι μηδέν (0), και έτσι το PARAVER όταν το TSU thread βρίσκεται εκτός των σημαντικών λειτουργιών-συναρτήσεών του θα ζωγραφίζει με γαλάζιο το TSU thread. Όταν πρόκειται για το Extrae_event που βρίσκεται πριν την κλήση της συνάρτησης, ως δεύτερη παράμετρο (value) του Extrae_event βάζω μια διαφορετική τιμή κάθε φορά. Με αυτό τον τρόπο, όταν το PARAVER θα εμφανίσει το TSU thread θα μπορώ να διακρίνω ανάλογα με το χρώμα που χρησιμοποιεί σε ποια σημαντική συνάρτηση-λειτουργία βρίσκεται το TSU. Στο παράδειγμα της Εικόνας 7.2.1 όταν το TSU thread βρίσκεται μέσα στην CacheFlow συνάρτηση-λειτουργία, το PARAVER θα εμφανίζει στο TSU thread την τιμή (7), η οποία αντιστοιχεί στο κίτρινο χρώμα.

Επιπλέον, στον κώδικα του TSU έπρεπε με κάποιο τρόπο να παρακολουθώ την εξέλιξη του μεγέθους κάθε μιας από τις τρεις ουρές που έχει ένας Worker, Acknowledgment Queue (AQ), Waiting Queue (WQ) και Firing Queue (FQ). Αυτό έγινε κατορθωτό με την χρήση της συνάρτησης Extrae_event, στην οποία η πρώτη παράμετρος (type) καθορίζει το σε ποιου Worker την ουρά (AQ, WQ ή FQ) αναφέρομαι και η δεύτερη παράμετρος (value) το μέγεθος αυτής της ουράς την δεδομένη χρονική στιγμή. Αποφάσισα πως για τα WQ των Workers θα χρησιμοποιούνται οι τιμές 1000 ως 1999 για type, για τα AQ οι τιμές 2000 ως 2999 και για τα FQ οι τιμές 3000 ως 3999. Αν για παράδειγμα θέλω να δω την εξέλιξη του μεγέθους του WQ του Worker 1, θα επιλέξω για type την τιμή 1000, για το WQ του Worker 2 την τιμή 1001, για το WQ του Worker 3 την τιμή 1002 και ούτω καθεξής. Με την επιλογή αυτού του φάσματος χιλίων τιμών για κάθε είδος ουράς (AQ, WQ και FQ), μπορώ να υποστηρίξω την ύπαρξη μέχρι και 1000 Workers σε κάθε TSU.

Για την παρακολούθηση της εξέλιξης του μεγέθους των WQs των Workers, αποφάσισα όπως χρησιμοποιήσω ένα δυναμικό πίνακα (zacharias_WQ) τύπου long long int ο οποίος θα δεσμεύει τόσες θέσεις όσες και ο αριθμός των Workers του TSU και θα αρχικοποιεί την κάθε θέση με την τιμή μηδέν (0). Στην συνέχεια, τοποθέτησα μέσα στον κώδικα του TSU.c συναρτήσεις Extrae_event μετά από τα σημεία όπου προσθέτονταν ή αφαιρούνταν μια εγγραφή από το WQ (Εικόνα 7.2.2). Προσθήκη νέας εγγραφής στο WQ είχαμε όταν η κλήση στην συνάρτηση UpdateSM επέστρεφε τιμή μεγαλύτερη ή ίση με δύο (2) και αφαίρεση όταν είχαμε κλήση της συνάρτησης Dequeue για WQ.

```
if (!Dequeue (g_WQ[nKernelNum],....,WQ_SIZE)) {
     printf("[TSU] Failure Peeking from WQ\n");
     return 0;
}
//Zacharias
Extrae_event(1000+nKernelNum,--zacharias_WQ[nKernelNum]);
```

```
nRes = UpdateSM(AqTNum,UpdateContext,UpdateValue);

// 2 : success, a thread RC = 0
if( nRes >= 2 ){
     Extrae_event(1000+nRes-2,++zacharias_WQ[nRes-2]);
}
```

Εικόνα 7.2.2

Όταν έχω αφαίρεση μιας εγγραφής από το WQ (Εικόνα 7.2.2 πάνω μέρος), ως πρώτη παράμετρο (type) στην συνάρτηση Extrae_event περνώ την τιμή 1000+nKernelNum (όπου nKernelNum ο αριθμός του Worker thread [0,N-1]). Έτσι καθορίζω πως το event αυτό αφορά το μέγεθος του WQ του Worker thread nKernelNum. Ως δεύτερη παράμετρο (value) στην συνάρτηση Extrae_event περνώ την τιμή της θέσης του δυναμικού πίνακα που αφορά το συγκεκριμένο Worker thread (nKernelNum) μειωμένη κατά μία (1) μονάδα.

Όταν έχω προσθήκη νέας εγγραφής στο WQ (Εικόνα 7.2.2 κάτω μέρος), ως πρώτη παράμετρο (type) στην συνάρτηση Extrae_event περνώ την τιμή 1000+nRes-2 (όπου nRes ο αριθμός του Worker thread [0,N-1] αυξημένος κατά δύο μονάδες). Έτσι καθορίζω πως το event αυτό αφορά το μέγεθος του WQ του Worker thread nRes-2. Ως δεύτερη παράμετρο (value) στην συνάρτηση Extrae_event περνώ την τιμή της θέσης του δυναμικού πίνακα που αφορά το συγκεκριμένο Worker thread (nRes-2) αυξημένη κατά μία (1) μονάδα.

Για την παρακολούθηση της εξέλιξης του μεγέθους των AQs των Workers είχα το πρόβλημα πως προσθήκη μιας νέας εγγραφής κάνει το Worker thread και αφαίρεση μιας εγγραφής το TSU thread. Έτσι αποφάσισα όπως αυτή την φορά χρησιμοποιήσω δύο δυναμικούς πίνακες (zacharias_AQ_Enqueue και zacharias_AQ_Dequeue) τύπου long long int οι οποίοι θα δεσμεύουν τόσες θέσεις όσες και ο αριθμός των Workers του TSU και θα αρχικοποιούν την κάθε θέση με την τιμή μηδέν (0). Προσθήκη μιας νέας εγγραφής στο AQ έχουμε όταν γίνεται κλήση της συνάρτησης AQEnqueue και αφαίρεση όταν

έχουμε κλήση της συνάρτησης AQDequeue. Και οι δύο αυτές συναρτήσεις βρίσκονται στο αρχείο TSUUtility.c.

Έτσι, όταν γίνεται προσθήκη μιας εγγραφής στο AQ (Εικόνα 7.2.3 πάνω μέρος), απλά αυξάνω κατά μία (1) μονάδα την τιμή της θέσης του δυναμικού πίνακα zacharias_AQ_Enqueue που αφορά το συγκεκριμένο Worker thread (speid), ενώ όταν γίνεται αφαίρεση μιας εγγραφής από το AQ (Εικόνα 7.2.3 κάτω μέρος), απλά αυξάνω κατά μία (1) μονάδα την τιμή της θέσης του δυναμικού πίνακα zacharias_AQ_Dequeue που αφορά το συγκεκριμένο Worker thread (live_AQ). Επιπλέον, στην περίπτωση που γίνεται αφαίρεση μιας εγγραφής από το AQ, τοποθετώ και μια συνάρτηση Extrae_event στην οποία περνώ ως πρώτη παράμετρο (type) την τιμή 2000+live_AQ (όπου live_AQ ο αριθμός του Worker thread [0,N-1]). Έτσι καθορίζω πως το event αυτό αφορά το μέγεθος του AQ του Worker thread live_AQ. Ως δεύτερη παράμετρο (value) στην συνάρτηση Extrae_event περνώ την διαφορά που προκύπτει από την αφαίρεση της τιμής της θέσης του δυναμικού πίνακα zacharias_AQ_Dequeue που αφορά το συγκεκριμένο Worker thread (live_AQ) από την τιμή της θέσης του δυναμικού πίνακα zacharias_AQ_Enqueue που αφορά το συγκεκριμένο Worker thread (live_AQ). Μπορείτε να δείτε την διαδικασία που ακολουθείται για την παρακολούθηση της εξέλιξης του μεγέθους των AQs των Workers στην Εικόνα 7.2.3.

```
//Zacharias
++zacharias_AQ_Enqueue[speid];
```

```
//Zacharias
++zacharias_AQ_Dequeue[live_AQ];
Extrae_event(2000+live_AQ,
zacharias_AQ_Enqueue[live_AQ]-zacharias_AQ_Dequeue[live_AQ]);
```

Εικόνα 7.2.3

Για την παρακολούθηση της εξέλιξης του μεγέθους των FQs των Workers είχα το πρόβλημα πως προσθήκη μιας νέας εγγραφής κάνει το TSU thread και αφαίρεση μιας εγγραφής το Worker thread. Έτσι αποφάσισα όπως αυτή την φορά χρησιμοποιήσω δύο δυναμικούς πίνακες (zacharias_FQ_Enqueue και zacharias_FQ_Dequeue) τύπου long long int οι οποίοι θα δεσμεύουν τόσες θέσεις όσες και ο αριθμός των Workers του TSU και θα αρχικοποιούν την κάθε θέση με την τιμή μηδέν (0). Προσθήκη μιας νέας εγγραφής στο FQ έχουμε όταν γίνεται κλήση της συνάρτησης Add_DCD_Entry του αρχείου

CacheFlowCMP και αφαίρεση όταν έχουμε κλήση της συνάρτησης GetNextThread του αρχείου CMPAuxiliary.c.

Έτσι, όταν γίνεται προσθήκη μιας εγγραφής στο FQ (Εικόνα 7.2.4 πάνω μέρος), απλά αυξάνω κατά μία (1) μονάδα την τιμή της θέσης του δυναμικού πίνακα zacharias_FQ_Enqueue που αφορά το συγκεκριμένο Worker thread (nKernelNum), ενώ όταν γίνεται αφαίρεση μιας εγγραφής από το FQ (Εικόνα 7.2.4 κάτω μέρος), απλά αυξάνω κατά μία (1) μονάδα την τιμή της θέσης του δυναμικού πίνακα zacharias_FQ_Dequeue που αφορά το συγκεκριμένο Worker thread (speid). Επιπλέον, στην περίπτωση που γίνεται αφαίρεση μιας εγγραφής από το FQ, τοποθετώ και μια συνάρτηση Extrae_event στην οποία περνώ ως πρώτη παράμετρο (type) την τιμή 3000+speid (όπου speid ο αριθμός του Worker thread [0,N-1]). Έτσι καθορίζω πως το event αυτό αφορά το μέγεθος του FQ του Worker thread speid. Ως δεύτερη παράμετρο (value) στην συνάρτηση Extrae_event περνώ την διαφορά που προκύπτει από την αφαίρεση της τιμής της θέσης του δυναμικού πίνακα zacharias_FQ_Dequeue που αφορά το συγκεκριμένο Worker thread (speid) από την τιμή της θέσης του δυναμικού πίνακα zacharias_FQ_Enqueue που αφορά το συγκεκριμένο Worker thread (speid). Μπορείτε να δείτε την διαδικασία που ακολουθείται για την παρακολούθηση της εξέλιξης του μεγέθους των FQs των Workers στην Εικόνα 7.2.4.

```
//Zacharias
++zacharias_FQ_Enqueue[nKernelNum];
```

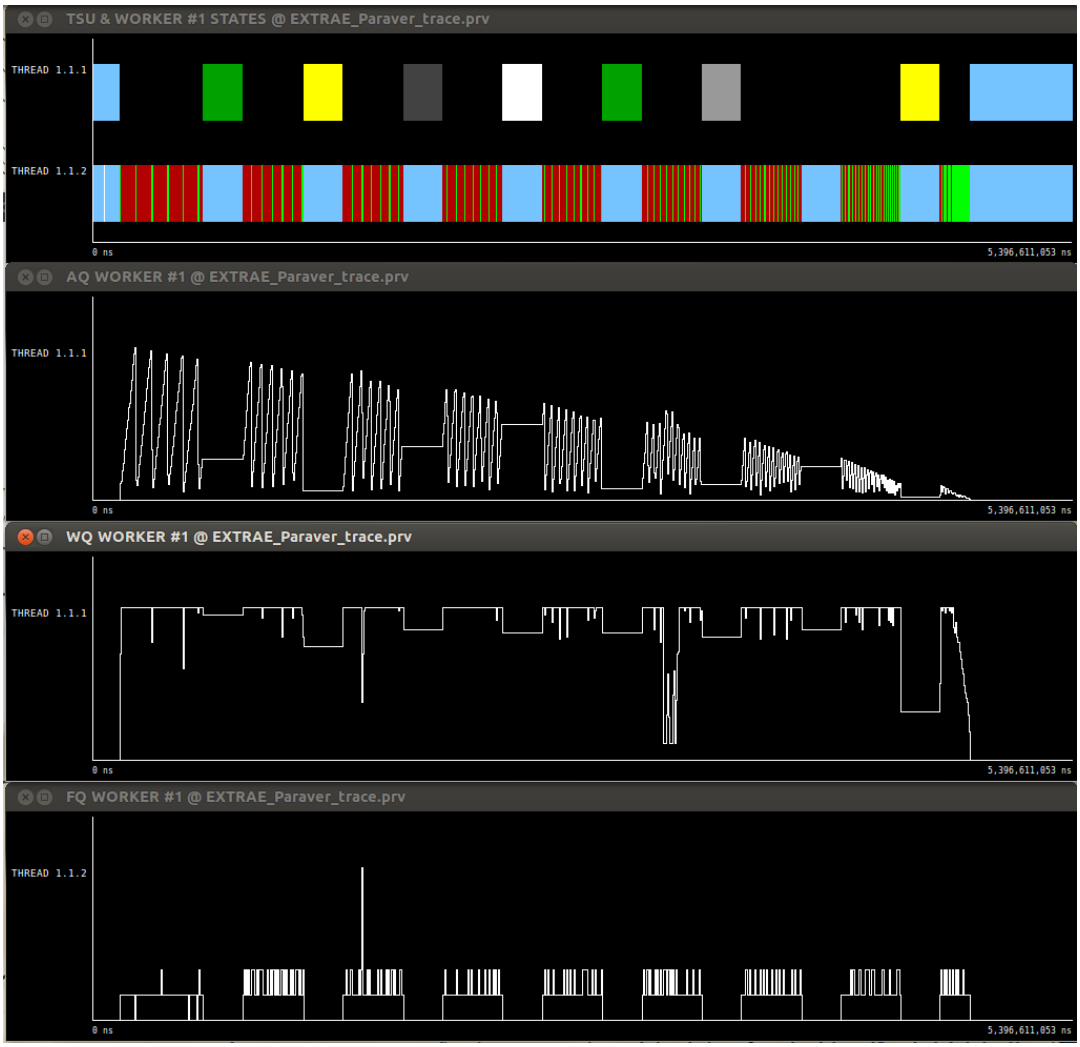
```
//Zacharias
++zacharias_FQ_Dequeue[speid];
Extrae_event(3000+speid,
zacharias_FQ_Enqueue[speid]-zacharias_FQ_Dequeue[speid]);
```

Εικόνα 7.2.4

Κεφάλαιο 8

Περιγραφή Screenshots και Συμπεράσματα

8.1 Single TSU – Cholesky



Εικόνα 8.1.1

Η Εικόνα 8.1.1 είναι ένα στιγμιότυπο της εκτέλεσης της εφαρμογής Cholesky με παραμέτρους NROWS=512, BLOCK_SIZE=4 και KERNEL_NUM=7. Το εντελώς πάνω παράθυρο απεικονίζει στην πρώτη γραμμή τις καταστάσεις του TSU thread (Πίνακας 8.1.1) και στην δεύτερη γραμμή τις καταστάσεις του Worker 1 thread (Πίνακας 8.1.2). Για να προκύψει η εικόνα του πρώτου παραθύρου έκανα δεξί κλικ πάνω σε αυτό και στην επιλογή "Select Rows" επέλεξα το πρώτο (TSU) και δεύτερο (Worker 1) thread. Ακολούθως, επέλεξα στο "Filter" όπως το PARAVER παρουσιάζει μόνο τα events με type 20 (TSU events) και 24 (Workers events). Τέλος, στο "Semantic" επέλεξα το "Last Event Value" και ακολούθως έκανα δεξί κλικ πάνω στο παράθυρο και επέλεξα "Fit Semantic Scale" → "Fit Both".

Unknown Value 0	Οποιαδήποτε άλλη λειτουργία του TSU.
Unknown Value 1	Λειτουργίες του FQ (Get DCD Entry).
Unknown Value 2	Decrement Consumers.
Unknown Value 3	Peek Head.
Unknown Value 4	Get Size (για να δει σε τι κατάσταση είναι τα queues).
Unknown Value 6	Get Graph Memory Entry.
Unknown Value 7	Cache Flow (Μεταφορά από WQ σε FQ).
Unknown Value 9	Λειτουργίες του FQ (Init DCD Entry).
Unknown Value 14	Dequeue.
Unknown Value 16	Gen Fetch List.
Unknown Value 18	Λειτουργίες του FQ (Add DCD Line).
Unknown Value 19	Λειτουργίες του FQ (Add DCD Entry).

Πίνακας 8.1.1

Unknown Value 0	Ο χρόνος που ο Worker περιμένει το TSU να του πει τι να κάνει.
Unknown Value 1	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower triangular matrix & conjugate transpose).
Unknown Value 2	Συναρτήσεις αρχικοποίησης Worker (καταγραφή IFP, Worker ID).
Unknown Value 3	Καθορισμός των threads της εφαρμογής που θα απελευθερωθούν.
Unknown Value 4	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower triangular matrix & conjugate transpose).
Unknown Value 5	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower triangular matrix & conjugate transpose).
Unknown Value 6	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower triangular matrix & conjugate transpose).

Πίνακας 8.1.2

Το δεύτερο από πάνω παράθυρο απεικονίζει τις τιμές του AQ SIZE του Worker 1 thread. Για να προκύψει η εικόνα αυτή έκανα δεξί κλικ πάνω σε αυτό και στην επιλογή "Select Rows" επέλεξα το πρώτο (TSU) thread. Ακολούθως, επέλεξα στο "Filter" όπως το PARAVER παρουσιάζει μόνο τα events με type 2000 (Worker 1 AQ). Επίσης, στο "Semantic" επέλεξα το "Last Event Value" και ακολούθως έκανα δεξί κλικ πάνω στο παράθυρο και απενεργοποίησα την επιλογή "View" → "Function Line With Color". Τέλος, έκανα δεξί κλικ πάνω στο παράθυρο και επέλεξα "Fit Semantic Scale" → "Fit Both".

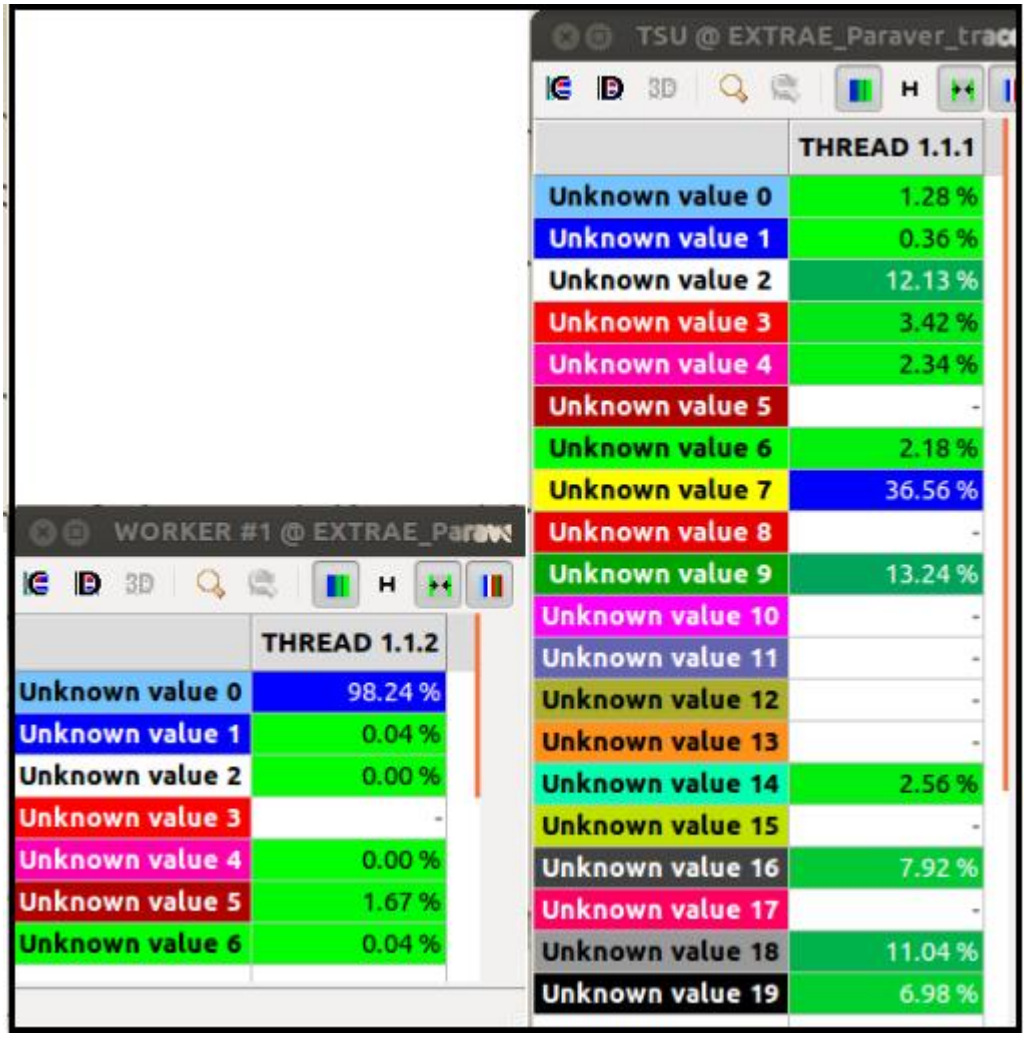
Το τρίτο από πάνω παράθυρο απεικονίζει τις τιμές του WQ SIZE του Worker 1 thread. Για να προκύψει η εικόνα αυτή έκανα δεξί κλικ πάνω σε αυτό και στην επιλογή "Select Rows" επέλεξα το πρώτο (TSU) thread. Ακολούθως, επέλεξα στο "Filter" όπως το PARAVER παρουσιάζει μόνο τα events με type 1000 (Worker 1 WQ). Επίσης, στο "Semantic" επέλεξα το "Last Event Value" και ακολούθως έκανα δεξί κλικ πάνω στο παράθυρο και απενεργοποίησα την επιλογή "View" → "Function Line With Color". Τέλος, έκανα δεξί κλικ πάνω στο παράθυρο και επέλεξα "Fit Semantic Scale" → "Fit Both".

Το τελευταίο παράθυρο απεικονίζει τις τιμές του FQ SIZE του Worker 1 thread. Για να προκύψει η εικόνα αυτή έκανα δεξί κλικ πάνω σε αυτό και στην επιλογή "Select Rows" επέλεξα το δεύτερο (Worker 1) thread. Ακολούθως, επέλεξα στο "Filter" όπως το PARAVER παρουσιάζει μόνο τα events με type 3000 (Worker 1 FQ). Επίσης, στο "Semantic" επέλεξα το "Last Event Value" και ακολούθως έκανα δεξί κλικ πάνω στο παράθυρο και απενεργοποίησα την επιλογή "View" → "Function Line With Color". Τέλος, έκανα δεξί κλικ πάνω στο παράθυρο και επέλεξα "Fit Semantic Scale" → "Fit Both".

Από την Εικόνα 8.1.1 παρατηρώ πως το μέγεθος του AQ αυξάνεται κάθε φορά που το Worker thread εκτελεί κάποια threads. Αυτή είναι αναμενόμενη συμπεριφορά, καθώς όσα threads εκτελούνται από τον Worker πρέπει να μπαίνουν στο AQ έτσι ώστε να ενημερωθούν αργότερα τα Ready Counts των consumers τους. Επίσης, κάνοντας μια σύγκριση ανάμεσα στα μεγέθη του WQ και του FQ, παρατηρώ πως όταν υπάρχει μείωση στο μέγεθος του WQ, το μέγεθος του FQ αυξάνεται. Αυτό είναι αναμενόμενο καθώς με βάση το Cache Flow policy, οτιδήποτε υπάρχει στο WQ, προτού εκτελεστεί από τον Worker μπαίνει στο FQ. Ακόμη, παρατηρώντας τη διακύμανση του μεγέθους του FQ σε σχέση με τις καταστάσεις του Worker, παρατηρώ πως όταν το μέγεθος του FQ μειώνεται, ο Worker ξεκινά να εκτελεί τις πράξεις του decomposition. Αυτό συμβαίνει για το λόγο ότι ο Worker παίρνει από το FQ όλα τα έτοιμα προς εκτέλεση threads και τα εκτελεί. Μια άλλη σημαντική παρατήρηση για την Εικόνα 8.1.1 είναι ότι το μέγεθος του WQ σε κάποια χρονικά σημεία παραμένει flat και δεν αυξάνεται περισσότερο. Αυτό οφείλεται στο γεγονός πως το WQ έχει φτάσει στο μέγιστο μέγεθος threads (63) που μπορεί να κρατήσει και ως αποτέλεσμα δεν μπορεί να δεχτεί άλλα threads. Τέλος, αξίζει να αναφέρω πως σε αρκετές περιπτώσεις ενώ το WQ περιέχει κάποια threads, το FQ παραμένει άδειο με αποτέλεσμα ο Worker να μην εκτελεί τίποτα, έστω και αν υπήρχαν έτοιμα προς εκτέλεση threads.

Η Εικόνα 8.1.2 περιέχει κάποια στατιστικά στοιχεία σχετικά με το ποσοστό του χρόνου που καταναλώνει ο Worker 1 (αριστερά) μέσα στις διάφορες καταστάσεις του όπως αυτές φαίνονται στον Πίνακα 8.1.2. Επίσης, περιέχει κάποια στατιστικά στοιχεία σχετικά με το ποσοστό του χρόνου που καταναλώνει το TSU (δεξιά) μέσα στις διάφορες καταστάσεις του όπως αυτές φαίνονται στον Πίνακα 8.1.1.

Για να προκύψουν τα δύο παράθυρα της Εικόνας 8.1.2 πάτησα πάνω στο εικονίδιο "Histogram" που είναι κυκλωμένο στην Εικόνα 6.5.1. Ως αποτέλεσμα ανοίχτηκε το παράθυρο "Create Histogram" (Εικόνα 6.9.1) στην οθόνη στο οποίο επέλεξα για ποιο Display Window (TSU ή Worker 1) επιθυμώ να πάρω στατιστικές μετρήσεις.



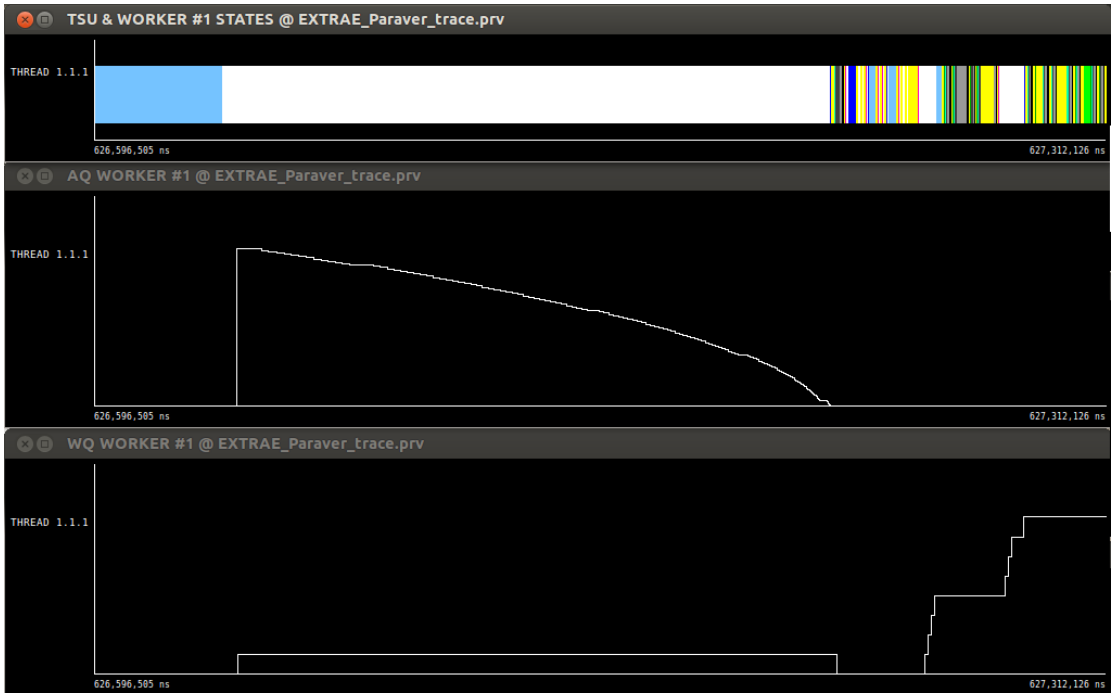
Εικόνα 8.1.2

Πατώντας το κουμπί “OK” στο παράθυρο “Create Histogram”, έχει ως αποτέλεσμα την εμφάνιση της Εικόνας 8.1.2, της οποίας η τελική μορφή προκύπτει ενεργοποιώντας τα κουμπιά “Label Colors” και “Hide empty columns”, και απενεργοποιώντας το κουμπί Horizontal/Vertical.

Με μια απλή παρατήρηση από το αριστερά παράθυρο της Εικόνας 8.1.2, φαίνεται πως το ποσοστό του χρόνου για το οποίο ο Worker 1 εκτελεί πράξεις decomposition είναι πολύ μικρό (σύνολο περίπου 1,75%), ενώ καταναλώνει το 98,24% του χρόνου του σε κατάσταση αναμονής μέχρι να του δώσει εργασία το TSU. Ένας λόγος που συμβαίνει αυτό είναι όπως ανέφερα και προηγουμένως, το γεγονός πως ενώ το WQ είχε έτοιμα προς εκτέλεση threads, το FQ παρέμενε άδειο με αποτέλεσμα ο Worker 1 να μην εκτελεί πράξεις decomposition και απλά να αναμένει το TSU να του δώσει εργασία. Από το δεξιά παράθυρο της Εικόνας 8.1.2, φαίνεται πως το μεγαλύτερο ποσοστό του χρόνου για το TSU (σύνολο 68%) καταναλώνεται για Cache Flow λειτουργίες (μεταφορά των έτοιμων

threads από το WQ στο FQ). Τέλος, ένα ποσοστό γύρω στο 14,5% καταναλώνεται για λειτουργίες ανεύρεσης των consumers ενός thread (Get Graph Memory Entry) και μείωση των Ready Counters τους (Decrement Consumers).

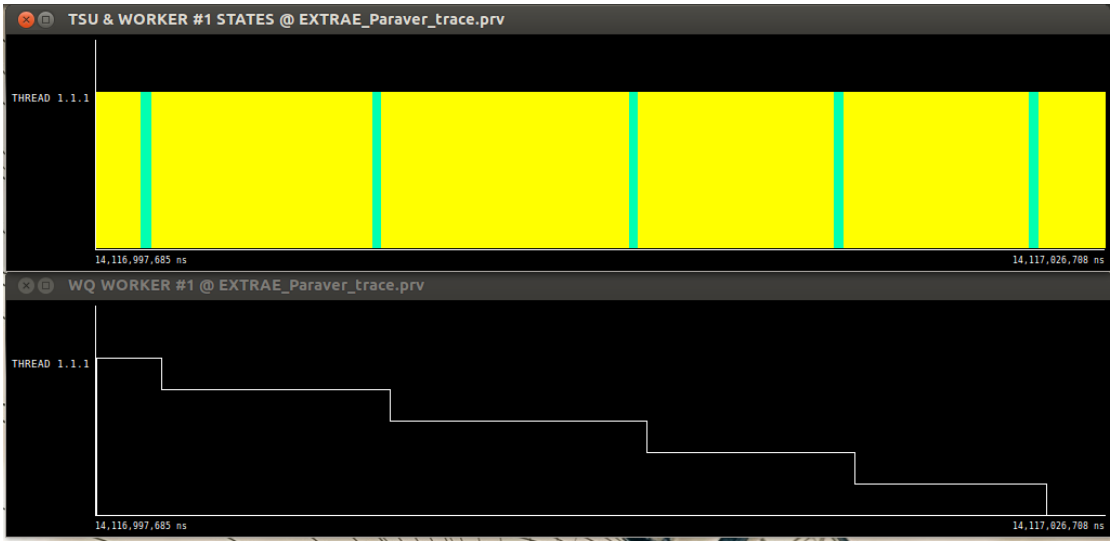
Η Εικόνα 8.1.3 είναι ένα στιγμιότυπο της εκτέλεσης της εφαρμογής Cholesky με παραμέτρους NROWS=2048, BLOCK_SIZE=16 και KERNEL_NUM=7. Το εντελώς πάνω παράθυρο απεικονίζει τις καταστάσεις του TSU thread (Πίνακας 8.1.1), το μεσαίο το μέγεθος του AQ και το κάτω παράθυρο το μέγεθος του WQ του Worker 1. Ο τρόπος με τον οποίο προέκυψε η πιο κάτω εικόνα, είναι ακριβώς ο ίδιος με τον τρόπο που προέκυψε η Εικόνα 8.1.1.



Εικόνα 8.1.3

Στην εικόνα αυτή θέλω να δείξω πως την στιγμή που το TSU εκτελεί λειτουργίες Decrement Consumers (άσπρο χρώμα), το μέγεθος του AQ μειώνεται καθώς απομακρύνονται από αυτό όσα threads είχαν εκτελεστεί και ενημερώθηκαν τα Ready Counts των consumers τους. Ταυτόχρονα, παρατηρώ πως αυξάνεται το μέγεθος του WQ καθώς με την λειτουργία Decrement Consumers προκύπτουν threads με Ready Count ίσο με μηδέν (0). Αυτά τα threads μεταφέρονται στο WQ.

8.2 Single TSU - LU



Εικόνα 8.2.1

Η Εικόνα 8.2.1 είναι ένα στιγμιότυπο της εκτέλεσης της εφαρμογής LU με παραμέτρους NROWS=2048, BLOCK_SIZE=16 και KERNEL_NUM=7. Το πάνω παράθυρο απεικονίζει τις καταστάσεις του TSU thread (Πίνακας 8.1.1). Η εικόνα του πρώτου παραθύρου προκύπτει με τον ίδιο τρόπο που προέκυψε και η Εικόνα 8.1.1. Το κάτω παράθυρο απεικονίζει τις τιμές του WQ SIZE του Worker 1 thread.

Στην εικόνα αυτή θέλω να δείξω πως την στιγμή που το TSU εκτελεί λειτουργίες Dequeue WQ (πράσινο χρώμα), το μέγεθος του WQ μειώνεται καθώς απομακρύνονται από αυτό όσα threads μεταφέρονται στο FQ.

Εικόνα 8.2.2

Η Εικόνα 8.2.2 αποτελεί και πάλι ένα στιγμιότυπο της εκτέλεσης της εφαρμογής LU με παραμέτρους NROWS=2048, BLOCK_SIZE=16 και KERNEL_NUM=7. Το πάνω παράθυρο απεικονίζει τις καταστάσεις του TSU thread (Πίνακας 8.1.1). Η εικόνα του πρώτου παραθύρου προκύπτει με τον ίδιο τρόπο που προέκυψε και η Εικόνα 8.1.1. Το κάτω παράθυρο απεικονίζει τις τιμές του FQ SIZE του Worker 1 thread.
Στην εικόνα αυτή θέλω να δείξω πως την στιγμή που το TSU εκτελεί λειτουργίες Add DCD Line (γκρίζο χρώμα), δηλαδή προσθέτει στο FQ τα έτοιμα προς εκτέλεση threads, το μέγεθος του FQ αυξάνεται.

Unknown Value 0	Ο χρόνος που ο Worker περιμένει το TSU να του πει τι να κάνει.
Unknown Value 1	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower & upper triangular matrix).
Unknown Value 2	Συναρτήσεις αρχικοποίησης Worker (καταγραφή IFP, Worker ID).
Unknown Value 3	Καθορισμός των threads της εφαρμογής που θα απελευθερωθούν.
Unknown Value 4	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower & upper triangular matrix).
Unknown Value 5	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower & upper triangular matrix).
Unknown Value 6	Πράξεις για decomposition πίνακα σε γινόμενο δύο πινάκων (lower & upper triangular matrix).

Πίνακας 8.2.1

Στον Πίνακα 8.2.1 μπορείτε να δείτε τις πιθανές καταστάσεις στις οποίες μπορεί να βρίσκεται ένα Worker thread της εφαρμογής LU.

	THREAD 1.1.2		THREAD 1.1.2		THREAD 1.1.2
Unknown value 0	98.41 %	Unknown value 0	96.04 %	Unknown value 0	79.57 %
Unknown value 1	0.00 %	Unknown value 1	0.00 %	Unknown value 1	0.00 %
Unknown value 2	0.00 %	Unknown value 2	0.00 %	Unknown value 2	0.00 %
Unknown value 3	0.00 %	Unknown value 3	0.00 %	Unknown value 3	0.00 %
Unknown value 4	0.02 %	Unknown value 4	0.03 %	Unknown value 4	0.11 %
Unknown value 5	0.02 %	Unknown value 5	0.04 %	Unknown value 5	0.14 %
Unknown value 6	1.55 %	Unknown value 6	3.89 %	Unknown value 6	20.18 %

Εικόνα 8.2.3

Στην Εικόνα 8.2.3 βλέπουμε το ποσοστό του χρόνου που καταναλώνει ο Worker 1 μέσα στις διάφορες πιθανές καταστάσεις της εφαρμογής LU (Πίνακας 8.2.1). Στο αριστερά παράθυρο βλέπουμε το ποσοστό χρόνου για εκτέλεση με παραμέτρους NROWS=512, BLOCK_SIZE=4 και KERNEL_NUM=7, στο μεσαίο με παραμέτρους NROWS=1024, BLOCK_SIZE=8 και KERNEL_NUM=7 και στο δεξιά με παραμέτρους NROWS=2048, BLOCK_SIZE=16 και KERNEL_NUM=7.

Με μια σύγκριση των τιμών των τριών παραθύρων, φαίνεται πως το TSU αποδίδει καλύτερα όσο μεγαλώνει το μέγεθος προβλήματος, καθώς όπως παρατηρούμε το ποσοστό του χρόνου για το οποίο ο Worker 1 εκτελεί πράξεις decomposition αυξάνεται από 1,59% για NROWS=512, BLOCK_SIZE=4, σε 3,96% για NROWS=1024, BLOCK_SIZE=8 και σε 20,43% για NROWS=2048, BLOCK_SIZE=16.

8.3 Single TSU – Mult-or

Unknown Value 0	Ο χρόνος που ο Worker περιμένει το TSU να του πει τι να κάνει.
Unknown Value 1	Παίρνει δύο blocks και καθορίζει το γινόμενό τους.
Unknown Value 2	Συναρτήσεις αρχικοποίησης Worker (καταγραφή IFP, Worker ID).
Unknown Value 3	Καθορισμός των threads της εφαρμογής που θα απελευθερωθούν.

Πίνακας 8.3.1

Στον Πίνακα 8.3.1 μπορείτε να δείτε τις πιθανές καταστάσεις στις οποίες μπορεί να βρίσκεται ένα Worker thread της εφαρμογής Mult-or.

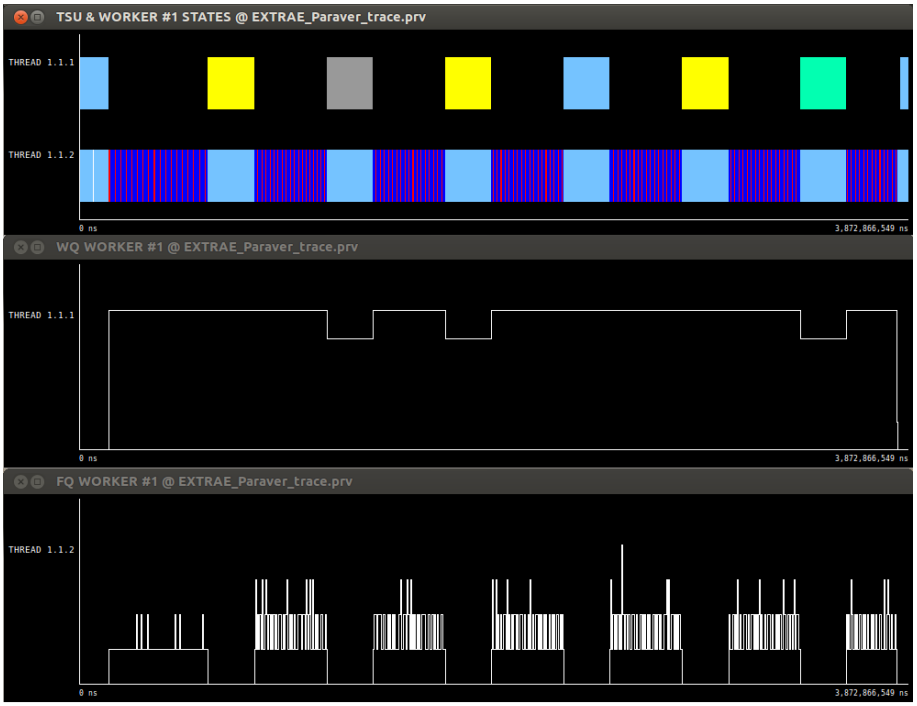
	THREAD 1.1.2		THREAD 1.1.2
Unknown value 0	95.48 %	Unknown value 0	49.13 %
Unknown value 1	4.50 %	Unknown value 1	50.87 %
Unknown value 2	0.00 %	Unknown value 2	0.00 %
Unknown value 3	0.02 %	Unknown value 3	0.00 %
Total	100.00 %	Total	99.00 %
Average	25.00 %	Average	24.00 %
Maximum	95.48 %	Maximum	50.87 %
Minimum	0.00 %	Minimum	0.00 %
StDev	40.73 %	StDev	25.01 %
Avg/Max	0.26	Avg/Max	0.49

Εικόνα 8.3.1

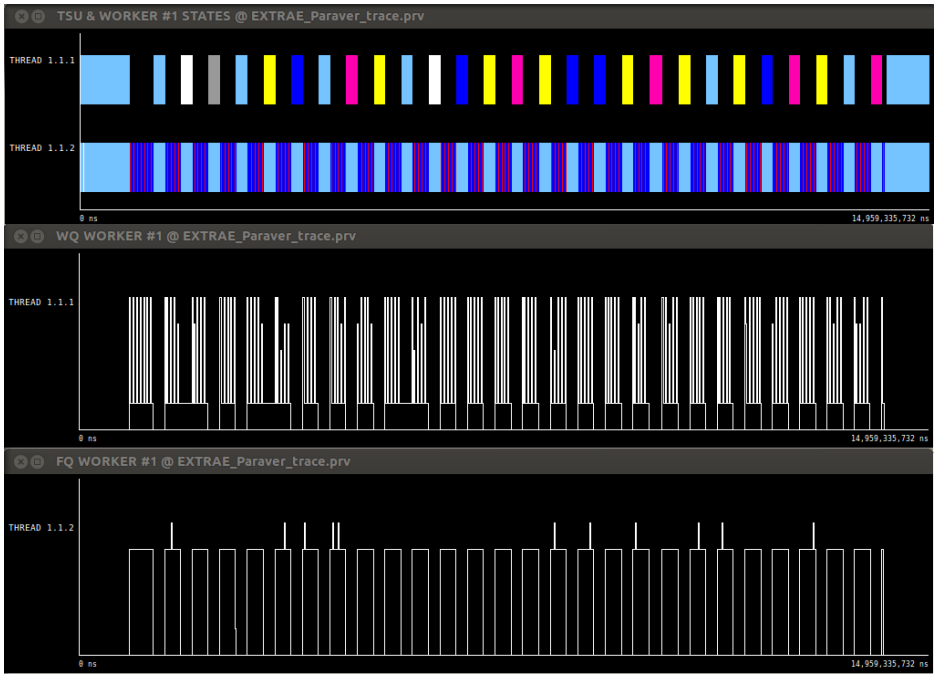
Στην Εικόνα 8.3.1 βλέπουμε το ποσοστό του χρόνου που καταναλώνει ο Worker 1 μέσα στις διάφορες πιθανές καταστάσεις της εφαρμογής Mult-or (Πίνακας 8.3.1). Στο αριστερά παράθυρο βλέπουμε το ποσοστό χρόνου για εκτέλεση με παραμέτρους NROWS=512, BLOCK_SIZE=8 και KERNEL_NUM=7 και στο δεξιά με παραμέτρους NROWS=2048, BLOCK_SIZE=32 και KERNEL_NUM=7.

Με μια σύγκριση των τιμών των δύο παραθύρων, φαίνεται πως το TSU αποδίδει καλύτερα όσο μεγαλώνει το μέγεθος προβλήματος, καθώς όπως παρατηρούμε το ποσοστό του χρόνου για το οποίο ο Worker 1 εκτελεί πράξεις πολλαπλασιασμού δύο blocks αυξάνεται από 4,5% για NROWS=512, BLOCK_SIZE=8 σε 50,87% για NROWS=2048, BLOCK_SIZE=32.

Αυτή η τεράστια βελτίωση της απόδοσης του TSU μπορεί να δικαιολογηθεί συγκρίνοντας τις Εικόνες 8.3.2 και 8.3.3. Στην Εικόνα 8.3.2 (NROWS=512, BLOCK_SIZE=8) παρατηρούμε πως αν και το WQ περιέχει κάποια threads, το FQ παραμένει άδειο με αποτέλεσμα ο Worker να μην εκτελεί τίποτα, έστω και αν υπήρχαν έτοιμα προς εκτέλεση threads. Αυτό το πρόβλημα δεν παρουσιάζεται στην Εικόνα 8.3.3 (NROWS=2048, BLOCK_SIZE=32), καθώς όταν το WQ περιέχει κάποια threads το FQ δεν μένει άδειο, με αποτέλεσμα ο Worker να εκτελεί πράξεις πολλαπλασιασμού δύο blocks.



Εικόνα 8.3.2



Εικόνα 8.3.3

Τέλος, στην Εικόνα 8.3.4 παρατηρούμε πως κάθε Worker εκτελεί πράξεις πολλαπλασιασμού δύο blocks για το ίδιο ποσοστό χρόνου (50-51%). Αυτό αποτελεί μια απόδειξη πως το TSU πετυχαίνει αρκετά καλό load balance μεταξύ των 7 Workers του.

	THREAD 1.1.2	THREAD 1.1.3	THREAD 1.1.4	THREAD 1.1.5	THREAD 1.1.6	THREAD 1.1.7	THREAD 1.1.8
Unknown value 0	49.26 %	49.45 %	49.36 %	49.36 %	49.25 %	49.24 %	48.93 %
Unknown value 1	50.74 %	50.55 %	50.64 %	50.64 %	50.75 %	50.75 %	51.07 %
Unknown value 2	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Unknown value 3	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Total	100 %	100 %	100 %	100 %	100 %	100.00 %	100 %
Average	25 %	25 %	25 %	25 %	25 %	25.00 %	25 %
Maximum	50.74 %	50.55 %	50.64 %	50.64 %	50.75 %	50.75 %	51.07 %
Minimum	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
StDev	25.00 %	25.00 %	25.00 %	25.00 %	25.00 %	25.00 %	25.01 %
Avg/Max	0.49	0.49	0.49	0.49	0.49	0.49	0.49

Εικόνα 8.3.4

8.4 Four TSUs – Mult-or

Τα screenshots αυτού του μέρους αφορούν εκτέλεση της Mult-or εφαρμογής με τέσσερα (4) TSUs, που το καθένα έχει από ένα Worker. Οι παράμετροι της εκτέλεσης είναι NROWS=512 και BLOCK_SIZE=32.

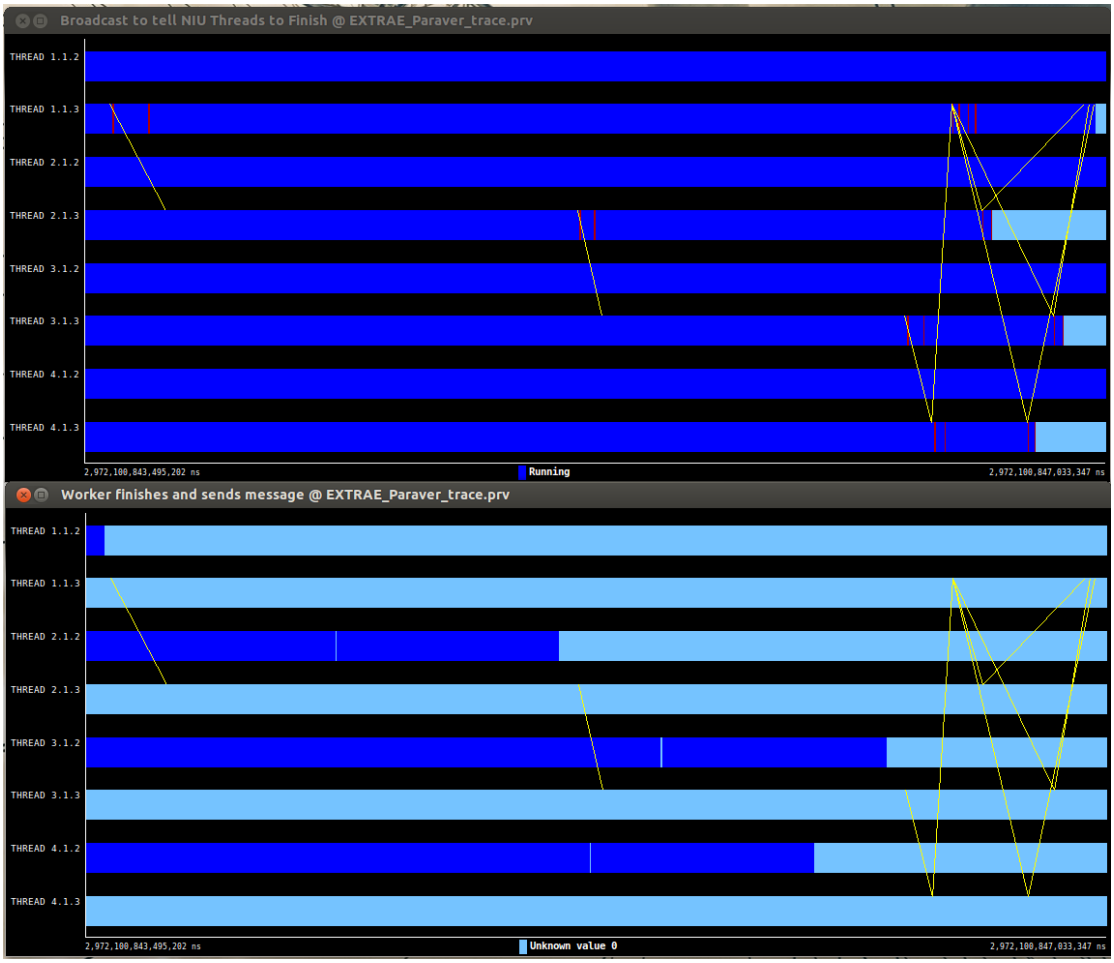
Στην Εικόνα 8.4.1 βλέπουμε το ποσοστό του χρόνου που καταναλώνει ο κάθε Worker μέσα στις διάφορες πιθανές καταστάσεις της εφαρμογής Mult-or (Πίνακας 8.3.1). Από την εικόνα αυτή παρατηρούμε πως και τα τέσσερα TSUs πετυχαίνουν αρκετά καλή απόδοση, καθώς το Worker thread του καθενός εκτελεί πράξεις πολλαπλασιασμού δύο blocks για ποσοστό γύρω στο 85,5% του χρόνου.

	THREAD 1.1.2		THREAD 2.1.2		THREAD 3.1.2		THREAD 4.1.2
Unknown value 0	14.52 %	Unknown value 0	14.40 %	Unknown value 0	14.41 %	Unknown value 0	14.50 %
Unknown value 1	85.43 %	Unknown value 1	85.53 %	Unknown value 1	85.52 %	Unknown value 1	85.43 %
Unknown value 2	0.02 %	Unknown value 2	0.02 %	Unknown value 2	0.02 %	Unknown value 2	0.02 %
Unknown value 3	0.04 %	Unknown value 3	0.05 %	Unknown value 3	0.05 %	Unknown value 3	0.04 %
Total	100 %	Total	100 %	Total	100 %	Total	99.00 %
Average	25 %	Average	25 %	Average	25 %	Average	24.00 %
Maximum	85.43 %	Maximum	85.53 %	Maximum	85.52 %	Maximum	85.43 %
Minimum	0.02 %	Minimum	0.02 %	Minimum	0.02 %	Minimum	0.02 %
StDev	35.38 %	StDev	35.44 %	StDev	35.43 %	StDev	35.39 %
Avg/Max	0.29	Avg/Max	0.29	Avg/Max	0.29	Avg/Max	0.29

Εικόνα 8.4.1

Στην Εικόνα 8.4.2 στο κάτω παράθυρο, αν πάρουμε ανά δύο τα ζευγάρια, το πρώτο thread σε κάθε ζευγάρι είναι ο Worker και το δεύτερο το NIU thread. Παρατηρούμε πως μόλις τελειώσουν οι Workers 1, 2 και 3 με τους υπολογισμούς τους, το αντίστοιχο NIU thread (1,2,3) στέλλει μήνυμα στο επόμενο NIU thread. Το τέταρτο NIU thread επειδή όταν παραλάβει το μήνυμα από το τρίτο NIU thread, ο Worker του έχει ήδη τελειώσει με τους υπολογισμούς του, στέλλει αμέσως μήνυμα πίσω στο πρώτο NIU thread. Μόλις παραλάβει το πρώτο NIU thread το μήνυμα, κάνει broadcast στα υπόλοιπα NIU threads το μήνυμα όπως τερματίσουν. Μόλις παραλάβει το κάθε NIU thread (2,3,4) το μήνυμα αυτό, στέλλει επιβεβαίωση πίσω στο πρώτο NIU thread πως παρέλαβε το μήνυμα και τερματίζει. Όταν το πρώτο NIU thread παραλάβει και τις 3 επιβεβαιώσεις, τερματίζει και αυτό.

Τα σημεία που τερματίζουν τα NIU threads φαίνονται στο πάνω παράθυρο της Εικόνας 8.4.2, όταν το χρώμα στις λωρίδες 2, 4, 6 και 8 από μπλε γίνεται γαλάζιο.



Εικόνα 8.4.2

Τέλος, να αναφέρω πως για όλες μου τις μετρήσεις-εκτελέσεις χρησιμοποίησα την μηχανή teras.

ΠΑΡΑΡΤΗΜΑ Α

Email 1ο

Ζαχαρίας Ζαχαρίου

Good morning!

Thank you very much for your immediate response!

I am working on my dissertation, which is about testing profiling tools for multithreaded applications.
I am also wondering if there is a way of creating a tracefile, by merging intermediate trace files which are produced from different applications.
For example, at the moment I am trying to calculate matrix multiplication by using pthreads and sockets in a small group of computers.

Thank you in advance.

Zacharias

Barcelona Supercomputing Center (BSC)

Hello Zacharias,

Some time ago we supported doing the merge (i.e. use mpi2prv) of several applications, but as this feature hasn't been used too much I feel it rather untested. However, I suggesst you to give it a try. Just use -- in mpi2prv to separate applications. For instance

```
$EXTRAE_HOME/bin/mpi2prv -f TRACE1.mpits -- -f TRACE2.mpits
```

Regards.

Email 2o

Ζαχαρίας Ζαχαρίου

Mr. Harald,

I have tried to write two examples in which I use the extrae_UserCommunication struct.
However, I get some errors for both of them during the merge process.

The first one has to do with two different executables (the client and the server). You can find the Makefile, xml file and C file at the following paths:
1) For the server ---> email/diffApps
2) For the client ---> email/diffApps/Client
After the execution of these two I get two ".mpits" files:
1) For the server ---> TRACE_SERVER.mpits
2) For the client ---> TRACE_CLIENT.mpits
Then, while I am in the diffApps directory I execute the command "make merge", which is:
$(EXTRAE_HOME)/bin/mpi2prv -f TRACE_SERVER.mpits -e ./server -- -f TRACE_CLIENT.mpits -e Client/client

During the merge process I get the following error:
"mpi2prv: Progress 2 of 2 ... 5.7% 11.4% 17.1% 22.9% 25.7% 31.4% 37.1% 42.9% 45.7% 51.4% 57.1% 62.9% mpi2prv: Error! Found unmatched communication! Continuing...
65.7% 71.4% 77.1% 82.9% 85.7% 91.4% 97.1% done
mpi2prv: Error! Found 1 unmatched communications. Resulting tracefile may be inconsistent."

Also, if you open the ".prv" file which is located at the path
email/diffApps/Paraver_Files
you will notice that there are no any communication lines between the two applications.

The second one has to do with one executable which creates two threads (the client and the server). You
can find the Makefile, xml file and C file at the following path:
email/sameApp

During the merge process I get the following error:
"mpi2prv: Progress 2 of 2 ... 5.2% 10.4% 15.6% 20.1% 25.3% 30.5% 35.1% 40.3% 45.5% 50.6% 55.2% 60.4% 65.6% 70.1% 75.3% 80.5% 85.1% 90.3% 95.5%
mpi2prv: Error! Found an unfinished state! Continuing...

done
mpi2prv: Error! Found 2 incomplete states. Resulting tracefile may be inconsistent."

Also, if you open the ".prv" file which is located at the path
email/sameApp/Paraver_Files
you will notice that there is only one communication line between the two threads, instead
of two.

In case you would like to know how the Extrae package was configured you can open the
"package_configuration.txt" file which is located under the "email" folder and contains
the output of the command ${EXTRAE HOME}/etc/configured.sh

I would really appreciate it if you could help me to find out what is going wrong with these
two examples.

Thank you in advance.

Zacharias

Barcelona Supercomputing Center (BSC)

Hello Zaccharias,

I've taken a look at your programs.

I noticed that you put partners 0 and 1 in the example/sameApp. This is the cause why you are not observing the second communication. These partners are considered for MPI applications only, so the 1st MPI process would be partner 0, the 2nd MPI process would be partner 1, and so on.

Regarding the email/diffApp, I'm sorry to say that we don't have implemented communications between different applications. As I commented in a previous mail, the functionality of generating a tracefile consisting of two apps is rather old & untested. Sorry for the inconvenience.

Regards.

Email 3o

Ζαχαρίας Ζαχαρίου

Hello Mr. Harald,

Since you told me that communications between different applications are not implemented, I came
up with another idea and I would appreciate it if you could help me.

I am running again two different applications (the server and client), but this time I have
commented out the extrae communication code and instead of it, I am using the simple function
Extrae_event(type,value) of the EXTRAE API.

So, I call the Extrae_event() function:
-before the write function and
-after the read function

I use the type parameter to specify a pair of communication (write-read) and the value parameter,
to specify if it has to do with a write function(SEND=1) or read function(RECEIVE=0). For example,
if you look at the code of server.c and client.c
- Before the first write function of the client.c there is an "Extrae_event(1,1);" and after the
first read function of the server.c there is an "Extrae_event(1,0)". This means that those two
events are one pair of communication(COMMUNICATION_1) because they have the same type, and the
client.c event is the SEND and the server.c event is the RECEIVE.
- Same with the "Extrae_event(2,1);" of the server.c and the "Extrae_event(2,0);" of client.c
which is the second pair of communication(COMMUNICATION_2).

After I run and merge the .mpit files of these two applications, I modify the .pcf file by adding
the following lines:

```
EVENT_TYPE
1000   1    COMMUNICATION_1 --
VALUES
1      SEND
0      RECEIVE


EVENT_TYPE
1000   2    COMMUNICATION_2 --
```

VALUES
1 SEND
0 RECEIVE

So, when I open the .prv file with the PARAVER, firstly I enable the "View --> Event Flags" and then
I zoom 4 to 5 times at the end of the application trace to see these 4 events.

The question I would like to ask you is:
Can I modify the source code of PARAVER so that when it recognizes that a pair of events is a
communication(SEND-RECEIVE), to draw a communication line between these two events? Just like
the extrae communication struct does, but only a simple line. If yes, can you please tell me where
I can find the part of the PARAVER's source code that has to do with the line drawing?

Also, because I have noticed that some event types are pre-defined, is there any list with the numbers
of these types, so that I won't "overwrite" them with my communication events?

Description of attached file:
- Client C code with .xml and Makefile at the path email/Client
- Server C code with .xml and Makefile at the path email
- Paraver files (.prv, .pcf, .row) at the path email/Paraver_Files

Thank you in advance.

Zacharias

Σε αυτό το email ΔΕΝ πήρα απάντηση!

ΠΑΡΑΡΤΗΜΑ Β

Email 1ο

Ζαχαρίας Ζαχαρίου

Hello Mr. Harald,

I am trying to install paraver on Ubuntu 12.04 LTS but during the "make" command I get the following error:

In file included from ../include/ktracecutter.h:33:0,
from ktracecutter.cpp:45:
../include/ktraceoptions.h:35:30: fatal error: libxml/xmlmemory.h: No such file or directory
compilation terminated.
make[2]: *** [ktracecutter.lo] Error 1
make[1]: *** [all-recursive] Error 1
make: *** [all] Error 2

Is there anything I can do about this?

Thank you in advance.

Zacharias

Barcelona Supercomputing Center (BSC)

Hello Zacharias,

I'll forward the message to our list so the people responsible for Paraver can help you with the issue.

Regards.

Barcelona Supercomputing Center (BSC)

Hello Zacharias,

Youd need to install the libxml2 and libxml2-dev packages in order to build paraver-kernel and wxparaver.

In future versions we'll improve the configure detection of this dependency.

Please don't hesitate to contact us again for any question or comment.

Best regards,

Pedro A. Gonzalez
Paraver Team

Email 2o

Ζαχαρίας Ζαχαρίου

Hello Mr. Gonzalez

I have already done this (see below), but again I still get the same error when I execute "make".

zzacha01@ubuntu:~$ sudo apt-get install libxml2
[sudo] password for zzacha01:
Reading package lists... Done
Building dependency tree
Reading state information... Done
libxml2 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

zzacha01@ubuntu:~$ sudo apt-get install libxml2-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libxml2-dev is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

I don't know if the BOOST is causing the problem. I installed "boost_1_50_0", and during the configuration of PARAVER I execute the following command:

./configure --with-boost=/home/zzacha01/Desktop/BOOST --prefix=/home/zzacha01/Desktop/PARAVER

Thank you in advance.
Zacharias

Barcelona Supercomputing Center (BSC)

Hi Zacharias,

There's a quick trick to solve it:

cd paraver-kernel/include; ln -s /usr/include/libxml2/libxml/

It will create a soft link to let "make" reach the includes of the libxml.

I hope this will help.

Regards,
Pedro

Email 3o

Ζαχαρίας Ζαχαρίου

Hello Mr. Gonzalez

The quick trick to solve the problem worked, and PARAVER is installed.

However, in the PARAVER/bin/ folder there is only "paramedir.bin".
Am I doing something wrong? Is there anything else I have to do?

Thank you in advance.

Zacharias

Barcelona Supercomputing Center (BSC)

Hi Zacharias,

wxparaver tool is composed of two main components and two libraries.

After installing paraver-kernel, you'll have the two libraries (kernel,
api) into lib/paraver-kernel, and the utility paramedir.bin into bin/.

The other main component is the GUI, wxparaver itself.
To run its configure you'll need to tell where wxWidgets and
wxpropertygrid are installed.

As an example:
./configure --prefix=/home/user/install
--with-wx-config=/home/user/libs/wxWidgets/GCCBuildReleaseGTK2DLL/wx-config --with-wxpropgrid=wxcode_gtk2_propgrid-2.8 --with-paraver=/home/user/install

Regards,
Pedro

Email 4o

Ζαχαρίας Ζαχαρίου

Hello Mr. Gonzalez

This time I have a problem with installing PropGrid.

The "configure" command works great, but during the "make" I get the following errors:

./src/advprops.cpp: In member function ‘virtual bool wxDateProperty::StringToValue(wxVariant&, const wxString&, int) const’:
./src/advprops.cpp:2074:100: error: cannot convert ‘bool’ to ‘const char*’ in initialization
In file included from ./src/advprops.cpp:21:0:
/home/zzacha01/Desktop/wxWidgets/include/wx-2.9/wx/object.h: In function ‘T* wxCheckCast(const void*, T*) [with T = wxPGVariantDataColourPropertyValue]’:
./src/advprops.cpp:920:1:   instantiated from here
/home/zzacha01/Desktop/wxWidgets/include/wx-2.9/wx/object.h:161:5: error: invalid static_cast from type ‘wxPGVariantDataColourPropertyValue*’ to type ‘const wxObject*’
/home/zzacha01/Desktop/wxWidgets/include/wx-2.9/wx/object.h:161:5: error: ‘ms_classInfo’ is not a member of ‘wxPGVariantDataColourPropertyValue’
make: *** [propgrid_dll_advprops.o] Error 1

The "configure" command is the following:

./configure --prefix=/home/zzacha01/Desktop/PropGrid --with-wx-config=/home/zzacha01/Desktop/wxWidgets/bin/wx-config

Is there anything I can do to fix this problem?
I apologize for the inconvenience, but I am quite new with all these things.
Thank you in advance.

Zacharias

Barcelona Supercomputing Center (BSC)

Hello Zacharias,

I'm afraid we don't give support for wxpropertygrid, and I'm not quite sure, but it seems to break due to some definitions mismatch between the wxWidgets includes and what wxpropertygrid expects.

Nowadays we are using the wxWidgets 2.8.12 (Current Stable Release http://www.wxwidgets.org/downloads/#latest_stable), and wxPropertyGrid 1.4.15 (latest one; http://wxpropgrid.sourceforge.net/cgi-bin/index) and it compiles and works properly in many platforms.

I see you're trying to compile against wxWidgets 2.9. Maybe you'll need to downgrade it.

We take note of all these issues for future libraries upgrade. We also apologize for these problems; we'll change our web page and our documentation (it's true it just says wxWidgets >= 2.8).

Thank you very much for your patience.

Pedro A. Gonzalez
Paraver Team

Email 5ο

Ζαχαρίας Ζαχαρίου

Hello Mr. Gonzalez

Now I am using the wxWidgets 2.8.12 and wxPropertyGrid 1.4.15 as you said, but I still get a different error message which is:

/usr/bin/ld: cannot find -lwxcode_gtk2u_propgrid-2.8
collect2: ld returned 1 exit status
make[1]: *** [wxparaver.bin] Error 1
make[1]: Leaving directory `/home/zzacha01/Desktop/paraver-source-4.3.4/wxparaver-4.3.4'
make: *** [all] Error 2

The "configure" command is the following:

./configure --prefix=/home/zzacha01/Desktop/PARAVER --with-wx-config=/home/zzacha01/Desktop/wxWidgets/bin/wx-config --with-paraver=/home/zzacha01/Desktop/PARAVER --with-boost=/home/zzacha01/Desktop/BOOST

However, I have noticed in a previous email you sent me that in your configure command you add the following:
--with-wxpropgrid=wxcode_gtk2_propgrid-2.8

What is this wxcode_gtk2_propgrid-2.8? Is it part of the PropertyGrid installation? Because I can't find it.

Thank you in advance.

Zacharias

Barcelona Supercomputing Center (BSC)

Hi Zacharias,

The error that you're getting now is given by the linker, because it can't find the library libwxcode_gtk2u_propgrid-2.8 (the "u" stands for unicode). I think this answers your question: it's part of the installation of wxpropgrid.

So, where is wxpropgrid installed? Which is the parameter "--prefix" of your wxpropgrid configure? The library should be there, under ./lib directory.

Also, please keep tools@bsc.es at CC: or "Reply all" to let our team follow our conversation.

Regards,
Pedro A. Gonzalez
Paraver Team

Email 6ο

Ζαχαρίας Ζαχαρίου

Hello again,

I installed wxpropgrid with --prefix=/home/zzacha01/Desktop/PropGrid.
When I type the command ls -l under the ~/Desktop/PropGrid/lib this is the output:

lrwxrwxrwx 1 zzacha01 zzacha01 32 Aug 7 15:06 libwxcode_gtk2_propgrid-2.8.so -> libwxcode_gtk2_propgrid-2.8.so.0
lrwxrwxrwx 1 zzacha01 zzacha01 36 Aug 7 15:06 libwxcode_gtk2_propgrid-2.8.so.0 -> libwxcode_gtk2_propgrid-2.8.so.0.0.0
-rwxr-xr-x 1 zzacha01 zzacha01 3744418 Aug 7 15:06 libwxcode_gtk2_propgrid-2.8.so.0.0.0

Now, I try to install PARAVER with the following configure command:

./configure --prefix=/home/zzacha01/Desktop/PARAVER --with-wx-config=/home/zzacha01/Desktop/wxWidgets/bin/wx-config --with-wxpropgrid=/home/zzacha01/Desktop/PropGrid/lib/libwxcode_gtk2_propgrid-2.8.so --with-paraver=/home/zzacha01/Desktop/PARAVER --with-boost=/home/zzacha01/Desktop/BOOST

but I get the following error:

/usr/bin/ld: cannot find -l/home/zzacha01/Desktop/PropGrid/lib/libwxcode_gtk2_propgrid-2.8.so
collect2: ld returned 1 exit status
make[1]: *** [wxparaver.bin] Error 1
make[1]: Leaving directory `/home/zzacha01/Desktop/paraver-source-4.3.4/wxparaver-4.3.4'
make: *** [all] Error 2

Is there anything I can do about this one?

Thank you in advance.

Zacharias

Barcelona Supercomputing Center (BSC)

Hi Zacharias,

There's another trick that may help you. When we compile wxpropgrid we install it in the same directory of wxWidgets install.

Our wxpropgrid configure is like this:

./configure --prefix=/home/user/wxWidgets/GCCBuildReleaseGTK2DLL/lib
--with-wx-prefix=../wxWidgets/GCCBuildReleaseGTK2DLL/lib
--with-wx-config=../wxWidgets/GCCBuildReleaseGTK2DLL/wx-config
--with-wx-exec-prefix=/home/user/wxWidgets/GCCBuildReleaseGTK2DLL/lib
--enable-shared

So the --with-wxprogrid parameter in wxparaver configure is without the path, like this:
--with-wxprogrid=wxcode_gtk2_propgrid-2.8

Can you test it?

Thank you,

Pedro A. Gonzalez
Paraver Team

Δοκίμασα και το πιο πάνω, αλλά και πάλι συνέχισα να παίρνω σφάλματα. Έτσι για αποφυγή περεταίρω καθυστέρησης, αποφάσισα να κατεβάσω την εγκατάσταση για την πλατφόρμα Linux-x86 64 bits.