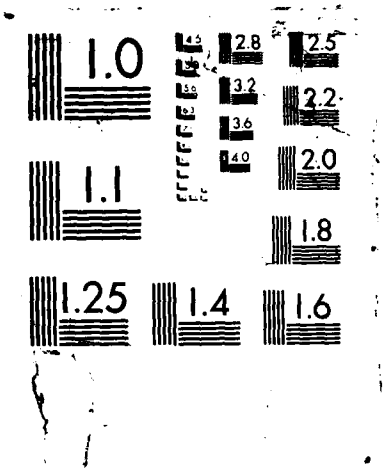
AD-A190 886 HIGH RESOLUTION PROCESS TIMING USER'S MANUAL(U) 1/1
ILLINOIS UNIV AT URBANA CENTER FOR SUPERCOMPUTING
RESEARCH AND DEVELOPMENT A D MALONEY 30 OCT 87
UNCLASSIFIED CSRD-676 AFOSR-TR-87-1970 F49620-86-C-0136 F/G 12/5 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

(2)

AD-A190 886

DTIC ELECTE JAN 19 1988 S D

REPORT DOCUMENTATION PAGE

1b. RESTRICTIVE MARKINGS

DTIC FILE COPY

2a. SECURITY CLASSIFICATION AUTHORITY

3. DISTRIBUTION / AVAILABILITY OF REPORT

Approved for public release; distribution unlimited.

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

CSRD Report No. 676

5. MONITORING ORGANIZATION REPORT NUMBER(S)

AFOSR-TR- 87-1970

6a. NAME OF PERFORMING ORGANIZATION
The Board of Trustees of the University of Illinois

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
AFOSR/NM

6c. ADDRESS (City, State, and ZIP Code)
506 S. Wright St.
Urbana, IL 61801

7b. ADDRESS (City, State, and ZIP Code)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
AFOSR

8b. OFFICE SYMBOL (If applicable)
NM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
~~AFOSR~~ F49620-86-C-0136

8c. ADDRESS (City, State, and ZIP Code)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61102F	2304	A3	

11. TITLE (Include Security Classification)
High Resolution Process Timing User's Manual

12. PERSONAL AUTHOR(S)
Allen D. Malony

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Internal Report	FROM 10/1/86 TO 9/30/87	Oct. 30, 1987	

16. SUPPLEMENTARY NOTATION

COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A high-resolution process timing facility, called HRTIME, has been implemented for the Cedar system. HRTIME is an extension of the Concentrix USER and SYSTEM process time measurements. It times both execution and non-execution process states with 10 μsec accuracy. In addition, HRTIME provides individual processor timing measurements to give a detailed account of the time spent in various states of sequential and concurrent execution. The main purpose of this manual is to explain how to use the HRTIME facility. In particular, the manual discusses how to access the timing data, to correctly time a program section, and to interpret the resulting time measurements. Although a brief overview is given describing what the time measurements are and how they are produced, the user should refer to [BELM87] for a complete discussion of the HRTIME design and implementation. (Keywords: multiprocessors; Fortran; program compilation).

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

22a. NAME OF RESPONSIBLE INDIVIDUAL
Maj. John P. Thomas

22b. TELEPHONE (Include Area Code)
(202)767-5026

22c. OFFICE SYMBOL
NM

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

High Resolution Process Timing User's Manual

Allen D. Malony
Center for Supercomputing
Research and Development

1. Introduction

A high–resolution process timing facility, called *HRTIME*, has been implemented for the Cedar system. *HRTIME* is an extension of the Concentrix™ USER and SYSTEM process time measurements. It times both execution and non–execution process states with 10 μsec accuracy. In addition, *HRTIME* provides individual processor timing measurements to give a detailed account of the time spent in various states of sequential and concurrent execution.

The main purpose of this manual is to explain how to use the *HRTIME* facility. In particular, the manual discusses how to access the timing data, to correctly time a program section, and to interpret the resulting time measurements. Although a brief overview is given describing what the time measurements are and how they are produced, the user should refer to [BELM87] for a complete discussion of the HRTIME design and implementation.

2. HRTIME Design Overview

The goal of the *HRTIME* facility is to provide high–resolution, detailed timing measurements of process operation. HRTIME is based on measured process timing with all times measured at 10 μsec resolution [Malo86]. The Concentrix USER and SYSTEM process states are extended to four execution states, and the measured times spent in these states are kept on a per processor basis. Non–execution states are also defined to track the time a process spends ready, blocked or idle.

2.1. Execution States

Four execution states are defined by *HRTIME*: USER, SYSTEM, OVERHEAD, and KERNEL. The USER state is active when user code is being executed. Processing of system calls occurs in the SYSTEM state. Interrupt processing that can be attributed to the current process falls into the OVERHEAD state; this includes interrupts for page faults and general exceptions, such as a floating point exception. Interrupts not directly associated with the current process are processed in the KERNEL state; cross–processor interrupts, device interrupts and timer interrupts are considered part of the KERNEL state. Actually, the KERNEL state is not a "process" execution state at all, but a situation where the operating system itself is executing. For this reason, only the USER, SYSTEM and OVERHEAD states are timed for a process.

In a multiprocessor system such as Cedar, it is possible for a process to be executing sequentially on one processor or concurrently on several processors. To further audit execution time, HRTIME keeps time measurements on a processing resource basis. In Cedar, a sequential process

executes on an IP, a detached-CE, and/or one CE of a cluster. HRTIME maintains a USER, SYSTEM and OVERHEAD timer for each of these "sequential" processing resources as part of the overall process time measurements.

All concurrent processes execute on the computational complex of the Alliant FX/8[1]. However, it is possible for processors participating in a concurrent computation to be in different states of execution. For this reason, the execution states should really be monitored at the processor level. For concurrent processes, *HRTIME* measures execution times per processor per state[2]. USER, SYSTEM and OVERHEAD timers are defined for each of the eight CEs.

Although the execution state timers defined above give detailed timing information, a complicated calculation must be made to determine total elapsed time, especially in the case of a concurrent process. For this purpose, a VIRTUAL execution state is defined; when any processing resource is executing in USER, SYSTEM or OVERHEAD state, the process is in a VIRTUAL execution state. A single timer is defined for the process VIRTUAL execution state.

2.2. Non-execution States

In addition to the execution states, three non-execution process states are recognized: READY, BLOCKED, and IDLE. Obviously, when a process is ready to execute, but not currently running, it is in the READY state. Similarly, a process is in the BLOCKED state when it is blocked from execution. The IDLE state occurs when a process is waiting for something to do. Because the process is not executing, only one timer is needed for each of these non-execution states.

2.3. State Timing

HRTIME works by detecting changes in execution and non-execution states, updating the time spent in the old state, and beginning the measurement of the delta time in the new state. The high-resolution, real-time clock is used to mark the point in time of the beginning of a new state and the end of the old state. Simple calculations are then made to update the old state times to their new values. For further description of *HRTIME* implementation see [BELM87].

3. HRTIME Data Structures

The *HRTIME* facility is always enabled. The operating system maintains the time measurements as part of the process's state. The state timers are stored as 64-bit integer values indicating the number of 10 μsec time units measured. The timer data structures are allocated as part of a larger process measurement structure in the process's read-only address space. This allows reference to any of the timers directly from the user's program.

The process timing data structures are declared in <sys/csrdetc.h>. The C data structure definitions of interest for this document are reproduced below[3]:

---

[1] A process is said to be *concurrent* if it requires more than one CE during its execution.

[2] This is not done by Concentrix. Whenever Concentrix detects any processor to be in SYSTEM state, the whole process is considered to be in SYSTEM state. USER time accumulates only when all processors are in USER state.

[3] The *hrcval* structure contains two 32-bit unsigned integers which together form a single 64-bit unsigned integer.

```
struct exectime
{     struct hrcval   user;      /* user time     */
      struct hrcval   system;    /* system time   */
      struct hrcval   overhead;  /* overhead time */
}
struct crestime
{     struct exectime cce[];     /* CE times          */
      struct exectime ip;        /* IP times          */
      struct exectime dce;       /* detached CE times */
      struct exectime cl;        /* cluster time      */
}
struct hrtimers
{     struct hrcval   pvtime;    /* process virtual time     */
      struct hrcval   tvtime;    /* task virtual time        */
      struct hrcval   notready;  /* time not ready           */
      struct hrcval   ready;     /* time ready, not running */
      struct hrcval   idle;      /* task idle time           */
      struct crestime execution; /* execution times          */
}
```

The *hrtimers* structure defines all time values HRTIME keeps for the process[4].

Together with the process's time values, a second set of timers is maintained for the process's children processes. Whenever a child process of the current process completes, it adds its children process times plus its own times to the children process times of its parent, the current process. The current process will likewise propagate is times to its parent when it completes.

4. Using the HRTIME Facility In A Program

The HRTIME facility is intended to be used is the same manner as the standard Concentrix timing facility. That is, routines to access the HRTIME measurements, similar to calling *etime()* and *dtime()* to retrieve USER and SYSTEM times values, are to be used in the same way for timing program sections and measuring overall program execution. However, unlike normal Concentrix timing, HRTIME provide significantly more detailed timing information. Interpreting the HRTIME measurements is the subject of a later section. Using the HRTIME facility is discussed in this section and the next.

4.1. Accessing HRTIME Measurements

Although the timers are in the user's virtual address space, because the process states are dynamically changing, it can be difficult to read a consistent set of timer values directly. Moreover, the user will have to perform update calculations, which the operating system normally does upon a state change, to produce up-to-date time values. The recommended way to access the *HRTIME* measurements is to use the system call *gethrtimers()* provided as part of the *csrd*

[4] *tvtime* is discussed in a later section.

library[5].

The format of the *gethrtimers()* system call is shown below:

```
gethrtimers(type, hrtimes)
int         type;       /* HRTIMERS_SELF (0)      */
                        /* HRTIMERS_CHILDREN (1)  */
struct hrtimers *hrtimes;
```

Being a system call, *gethrtimers()* causes a change of state from USER to SYSTEM on the calling processor. In addition, *gethrtimers()* forces all currently active timers to be updated to a consistent state. The appropriate time values, current process or children, are then transferred to the user's buffer, pointed to by *hrtimes*, and the process continues.

Raw timing data are returned by *gethrtimers()*. If the user wants decimal time values in seconds, the *csrd* library routine *hrtsecs()* can be called. The format of the *hrtsecs()* function call is:

```
double hrtsecs(hrtime)
struct hrcval *hrtime;
```

4.2. Time Measurements of a Program Section

The general procedure for making time measurements of a section of a program is shown below[6]:

```
gethrtimers(type, hrtimes1);
<execute program section>
gethrtimers(type, hrtimes2);
<calculate the time difference between hrtimes2 and hrtimes1>
```

As is obvious from above, the time to execute the program section is really the difference between two *struct hrtimers* samples, hrtimes1 and hrtimes2, determined before and after the program section. Following this procedure, time measurements for various program sections can be made. In fact, if the time samples are saved, a time-sample trace can be kept during program execution and a post-processor used to calculate the desired incremental time values.

4.3. Program Compilation

To use the HRTIME facility, the user program must include the following files:

```
sys/time.h
sys/csrdetc.h
```

In addition, the user program must be compiled with the *csrd* library.

---

[5] This document does not explain how to deal with the consistency problems involved in directly accessing the *HRTIME* data structures. Further discussion of this issue might be documented at a later time depending on need.

[6] Usually, the *type* argument to *gethrtimers()* will be HRTIMERS_SELF; i.e., the current process.

5. Automatic Program Time Measurements

In some cases, the user will want to time a program as a whole. A program, *hrtime*, has been written to run the user's program and print out timing statistics[7] without requiring any program modification. The *hrtime* command format is:

```
hrtime <user program> <user program arguments>
```

The timing results produced by *hrtime* include the execution and non-execution times. The execution times are shown as individual processor times. An example of the *hrtime* output for a concurrently executing program is given below:

```
PROCESS / TASK TIME
     process virtual :     5.89677
     task virtual    :     5.89677
     not ready       :     0.18924
     ready           :     1.64093
     idle            :     0.10224

IP, DETACHED CE and CLUSTER TIME
                             User        System      Overhead
     ip              :       0.00000     0.00000     0.00000
     dce             :       0.00105     0.05764     0.01262
     cluster         :       0.00000     0.00000     0.00000

CE TIME
                             User        System      Overhead
     ce [0]          :       5.73250     0.00092     0.00737
     ce [1]          :       5.73529     0.00000     0.00335
     ce [2]          :       5.73540     0.00000     0.00393
     ce [3]          :       5.73318     0.00416     0.00863
     ce [4]          :       5.73358     0.00000     0.00436
     ce [5]          :       5.73354     0.00000     0.00264
     ce [6]          :       5.73285     0.00000     0.00288
     ce [7]          :       5.71049     0.02005     0.00373
```

6. HRTIME and Fortran

Although the above discussion uses C for describing HRTIME data structures and routines, accessing the HRTIME facility from Fortran is no problem. The user only has to declare an equivalent *struct hrtimers* data structure whose address will be passed to the *gethrtimers()* routine. The following shows a suggested Fortran declaration and a call to *gethrtimers()*:

```
integer   type
integer*4 hrtimes(2,38)
```

[7] A manual page is available for the *hrtime* program on the CSRD Alliant machines.

```
call gethrtimers(type,hrtimes)
```

In this case, the *hrtimes* array is organized as:

```
process virtual time     : hrtimes(1:2,1)
task virtual time        : hrtimes(1:2,2)
time not ready           : hrtimes(1:2,3)
time ready, not running  : hrtimes(1:2,4)
task idle time           : hrtimes(1:2,5)

          user                 system               overhead
ce[0]  : hrtimes(1:2,6),   hrtimes(1:2,7),   hrtimes(1:2,8)
ce[1]  : hrtimes(1:2,9),   hrtimes(1:2,10),  hrtimes(1:2,11)
ce[2]  : hrtimes(1:2,12),  hrtimes(1:2,13),  hrtimes(1:2,14)
ce[3]  : hrtimes(1:2,15),  hrtimes(1:2,16),  hrtimes(1:2,17)
ce[4]  : hrtimes(1:2,18),  hrtimes(1:2,19),  hrtimes(1:2,20)
ce[5]  : hrtimes(1:2,21),  hrtimes(1:2,22),  hrtimes(1:2,23)
ce[6]  : hrtimes(1:2,24),  hrtimes(1:2,25),  hrtimes(1:2,26)
ce[7]  : hrtimes(1:2,27),  hrtimes(1:2,28),  hrtimes(1:2,29)

                 user                 system               overhead
ip             : hrtimes(1:2,30),  hrtimes(1:2,31),  hrtimes(1:2,32)
detached ce    : hrtimes(1:2,33),  hrtimes(1:2,34),  hrtimes(1:2,35)
cluster        : hrtimes(1:2,36),  hrtimes(1:2,37),  hrtimes(1:2,38)
```

As before, all times are 64–bit unsigned integer values indicating the number of 10 μsec time units measured. Again, the *hrtsecs()* function can be called to convert the integer time values to floating point values.

7. HRTIME and Xylem

Xylem supports multitasking of a process for concurrent execution across multiple Cedar clusters. Concentrix runs on each individual cluster and schedules Xylem tasks for execution. The question is how the HRTIME facility performs time measurements of Xylem tasks. Because Xylem tasks are essentially equivalent to Concentrix processes, the time measurement mechanism is exactly the same. Hence, all previous discussion also applies to Xylem tasks. However, some additional HRTIME measurement mechanisms for Xylem processes are provided.

A Xylem process is made up of one or more tasks. A Xylem process represents a logical execution environment; that is, only the tasks actually execute. However, HRTIME also measures time for a Xylem process. This is done by adding the execution time results of the Xylem process's tasks to its HRTIME time totals when the tasks complete. Thus, the execution time measurements for a Xylem process are the sum of all its tasks' execution times.

For a normal Concentrix process, the process VIRTUAL time, *pvtime*, is equivalent to the task VIRTUAL time, *tvtime*. However, for Xylem tasks and processes, these two VIRTUAL times take on different meanings. Task VIRTUAL time for a Xylem task is analogous to process VIRTUAL time for a Concentrix process. That is, task VIRTUAL time accumulates whenever any processing resource used by the task is in USER, SYSTEM or OVERHEAD state. Xylem

tasks do not use the process VIRTUAL timer.

On the other hand, Xylem processes do use the process VIRTUAL timer for determining total elapsed execution time for the Xylem process. A Xylem process is in a VIRTUAL execution state whenever any of its tasks are executing in USER, SYSTEM or OVERHEAD state; this is analogous to the Concentrix process VIRTUAL state definition except on at higher level.

8. Interpreting HRTIME Measurements

HRTIME provides significantly more detailed timing information than the standard Concentrix USER and SYSTEM times. In fact, in nearly all cases, the HRTIME facility can take the place of Concentrix timing for process time measurements; certainly, HRTIME will be used for non–execution timing and Xylem process/task timing. However, together with the benefit of more information comes the problem of understanding what it all means. This section makes a few comments on interpretating HRTIME measurements. An all–inclusive discussion is impossible since the time measurements will be used for different purposes. However, as users gain more experience with HRTIME, they will become more confident in their understanding of the measurements, and further consensus on standard timing measurement methodology is expected.

One aspect of HRTIME that might be curious is why the definition of USER, SYSTEM and OVERHEAD states instead of staying with USER and SYSTEM. Some system processing actually occurs on behalf of the user and, therefore, should be measured separately from the overhead of general OS operations. Doing so provides the user with a measurement of how much overhead processing the program is really experiencing during its execution and also how much system support the program is requiring. By monitoring the OVERHEAD times, as well as the USER and SYSTEM times, the user can get a sense of how vulnerable the program's performance is to the overhead processing.

Non–execution time measurements might be regarded as superfluous by some users. However, these times have real meaning and can reflect interesting process operation behavior. For instance, the READY time is a good indication of the amount of waiting a process experiences in the scheduling queue. The BLOCKED time is even more versatile in that it encompasses all dependent waiting (blocking) time encountered by the process. This time not only includes blocking due to I/O operations but also represents waiting due to inter–process and inter–task synchronization. IDLE time is particularly interesting for Xylem tasks because it can be used to compute a task utilization measure indicating the percentage of time a task was executing some portion of the user's program.

For the most part, the execution time measurements have clear definitions; e.g., the time spent executing in SYSTEM state on CE 5 or the total process virtual time. The complication comes when trying to work back from the measurements to what the program is actually doing. For sequential processes, the HRTIME measurements are not too difficult to understand. The breakdown across the different sequential processing resources is interesting because it shows how the process was scheduled during its execution. To determine the total USER, SYSTEM and OVERHEAD times the user could sum the IP, detached CE and cluster values. However, when the execution takes place on different physical processing resources, as in this case IPs and CEs, the user may want to regard the execution times differently.

The HRTIME measurements for concurrent processes are in some ways easy and in other ways difficult to interpret. On the one hand, the measurements are just a simple extension of the one CE cluster execution times to multiple CEs. On the other hand, the actions of multiple CEs

have to be determined instead of just one. The goals of the CE execution time measurements is to give some indication of CE resource usage. Ideally, a single global state space would be defined where each point describes a different combination of the CE execution states. Time spent in each global state could then be measured. However, the implementation of this measurement model is impractical. Thus, the individual CE measurements are provided.

Using the individual CE measurements, it is difficult to determine the amount of time CE 0 is in USER state when CE 1 in SYSTEM state, and so on, with the other CE state combinations. However, it is unclear whether such a time value has much meaning. Because the computational complex is assigned to a process as a single resource, it is more important how the individual CEs themselves are utilized. The HRTIME measurements show this as a breakdown between execution states for each CE. Unfortunately, HRTIME is unable to detect when a CE is idling. Otherwise, the time spent in a non–execution state could be measured for each CE and complete CE utilization statistics calculated. As it is, when a CE is idling, the CE appears to be in USER state and USER time is being accumulated.

9. Conclusion

The HRTIME facility is a new tool for timing programs on the Cedar multiprocessor. It is possible that significantly more questions will be raised about the HRTIME facility than were answered above. Being a new tool, this is to be expected. However, the additional detail of the HRTIME measurement should be a benefit over the simple USER and SYSTEM times currently reported by Concentrix. As user experience with HRTIME grows, it is hoped that standard timing practices will be developed.

References

[Malo86] Allen Malony. *Virtual High-Resolution Process Timing.* **CSRD Document #616**, Oct. 1986.

[BELM87] R. Barton, P. Emrath, D. Lawrie, A. Malony, R. McGrath. *New Approaches to Measuring Process Execution Time in the Cedar Multiprocessor System.* **CSRD Document #622**, Jan. 1987.

END

DATE

FILMED

5-88

DTIC