

# A New Approach to CICS Application Optimization

## Inspect-CPU

### Look Beyond Transaction Performance to Optimize Your CICS Applications

Conventional CICS® performance monitors identify CICS performance issues at the transaction level. However, these monitors give little insight into the consumption of CPU, your most expensive and critical resource.

Inspect-CPU (ICPU) is an innovative, real time CICS monitor that tracks CPU consumption at the program level and ranks programs by percentage of total CPU consumed. ICPU implements a unique low-overhead sampling approach that can be employed for long periods in a production environment without impacting performance—even during peak CPU usage! These extended sampling runs provide new insights into opportunities to save CPU resource and improve throughput.

ICPU delivers an interactive experience that lets users view results even as sampling runs. ICPU is user-friendly, integrates easily with the application development lifecycle and is used easily by everyone – from system programmers to developers – at any time.

### The ICPU Difference

Conventional CICS sampling products identify performance bottlenecks at the transaction level - bottlenecks that can often be eliminated by modifying code. To achieve their objectives, conventional sampling products collect a huge amount of performance information using a high-overhead machine instruction trace. Unfortunately, high sampling overhead limits their usefulness in production. They cannot be left running – nor can they be used in production during peak hours. As a result, they all share a common shortcoming – their use is typically limited to postmortem attempts to identify problems.

When a performance issue occurs in production, a system expert may turn on sampling in the corresponding test environment and try to replicate the transaction mix, or perhaps turn on sampling in production during a low use period in hopes the problem can be recreated there. After a few minutes, sampling is stopped and a batch job is run to generate an analysis. This after-the-fact approach optimistically assumes that the same problem will occur again in production during an arbitrary sampling window or that it can be forced to occur in test. Unfortunately, this is not always the case.

ICPU employs a unique sampling approach. Sampling is performed using a low-cost CICS subtask with standard low-cost CICS global user exits used to collect CICS service requests and TCB switching statistics. Interim results are efficiently stored in memory buffers rather than written to file. In short, ICPU generates very little overhead – around two percent regardless of the load on the system. By virtue of its very low overhead, ICPU can be run in production for long periods of time. This unique and very significant capability lends itself to two important use cases:

**Problem Capture in Real Time:** ICPU can be used for real time – as opposed to postmortem – problem identification. ICPU can run continuously to identify performance issues in production as they occur – eliminating the costly and time-consuming task of trying to recreate them after the fact. ICPU generates alerts based on user



defined CPU consumption thresholds, records the offending programs and the offsets within them, matches offsets to source listings, and facilitates immediate root cause analysis and optimization.

**Application Survey and Optimization:** Conventional sampling products identify performance hot spots at the transaction not the program level. However, a poor performing program can consume excessive CPU and create erratic response times across several CICS transaction types without creating a single transaction hotspot that sets off alarms!

How do you save CPU cycles and improve overall CICS application behavior when transaction monitoring is not helpful? You need to find the programs that inefficiently consume excess CPU and tune them. ICPU is the only sampling product that identifies the programs and the code sequences that monopolize CPU regardless of the mix of transactions using the programs. ICPU is the only sampling product that can run for an unlimited time in production aggregating CPU consumption numbers across all programs and code slices within them to create a 'big picture' not available with other tools.


ICPU uncovers hidden CPU savings opportunities for even well-performing applications. Consider, for example, a program that runs a million times a day and consumes 10 milliseconds of CPU per run. This adds up to almost 3 hours of CPU per day. Reducing the CPU time of this program by a third will save one hour of CPU time per day, or more than 15,000 minutes per year. At a rate of \$12 per CPU minute (insert your own metric), the yearly savings can be as much as \$180,000.

Conventional sampling products are cumbersome to use, complex and difficult to understand, and are generally limited in use to system programmers and other experts. ICPU, on the other hand, is an intuitive and easy to use solution that can be used daily by all personnel involved in the application lifecycle. Developers can profitably make use of ICPU early in the development process to better the odds that only efficient, well behaved programs are moved into production.

The insights provided by ICPU can be used to identify and modify your most inefficient program code resulting in saved CPU cycles, improved response times, postponed upgrades and reduced license costs. ICPU is the only product that delivers this information, presenting CPU saving opportunities that are otherwise invisible to the developers.

## ICPU's innovative design offers the following features

1. Real Time – Sampling results are available immediately and can be viewed at any time during the sampling run or after. You don't have to stop sampling to review results, and the view can be refreshed as sampling progresses.
2. Exception Notification – ICPU will alert you to any program that exceeds a predefined CPU usage threshold as soon as it occurs, so you can take a corrective action immediately.
3. Highly Scalable – ICPU can be used by any number of users, in any environment, at any time, for any length of time with no impact on the systems being monitored.
4. User-friendly – ICPU is easy to use and provides meaningful, easy-to-understand results.
5. Unattended Operation – ICPU can be set up to start automatically when CICS starts or at a scheduled time.

- 
6. Source Listing – Matches sampling data directly to COBOL source listings.
  7. Implemented Using Standards – ICPU uses only standard z/OS/CICS interfaces.
  8. Online Help – Available from any screen.
  9. Filtering – By user/transaction/terminal/program.
  10. Check Point – Assures that sampling results are not lost if CICS crashes.
  11. Batch Reports – Can be produced at any time, even while sampling is in progress.
  12. Model Inspections IDs – Any previous Inspection can be used as a model for a new one.
  13. Inline Command – Can be used to start an Inspection from any terminal type.
  14. Dynamic Library Support – For CICS TS 3.2 and above, ICPU supports the dynamic library feature of CICS.
  15. Generic Inspection IDs – A shorthand method of assigning Inspection IDs and referring to them.
  16. CPU Break Down – CPU usage and CICS services break down to the QR TCB and all other TCBs including L8, L9 and TCB switching statistics.
  17. Installation Verification Programs - Distributed in source and load formats. Can be used to verify ICPU installation and to familiarize users with the relative CPU usage of various CICS commands.
  18. Install and Use - A non-SMP/E installation, one load library for all CICS versions, installs in less than an hour.
  19. Effective Self-Learning Environment – Users will learn how to develop better, more efficient programs.

## Using ICPU

Each sampling session is called an ‘Inspection’ and it is identified by a unique ID assigned by the user. Inspection results are recorded to a VSAM file at user-defined time intervals. This file serves as a repository for the results of the current and all prior Inspections. You can easily review previous Inspection results as well as compare results between two different Inspections – for example before and after program code modification.

When the user initiates an Inspection, ICPU acquires buffers and begins collecting CPU usage information by program and program ‘Slice’ and reports CPU usage in order of decreasing significance. For executable programs, the size of a Slice is defined by the user, and for Natural, a Slice is one Natural statement.

During an Inspection, ICPU samples CICS at user-defined time intervals. At the beginning of each interval ICPU executes a ‘Sampling Event’ that identifies the application program and the component using the CPU, be it the application program itself or a service performed on behalf of the application, such as a CICS command, a SQL statement, or another system service, performed, for example, by Language Environment.

There are two types of Inspections: a ‘Regular Inspection’ and an ‘Exception Inspection’. For Regular Inspections, ICPU samples CICS continuously for as long as is desired and the sampling results are available at any point in time. For Exception Inspection sampling runs, ICPU checks for exceptions at each Sampling Event. An Exception Event is raised when the CPU use of any program exceeds the user-defined CPU threshold during the just ended interval. When an Exception Event is raised, ICPU will record the event and generate alerts.

ICPU includes a set of Installation Verification and Demo programs that can be used to both verify ICPU installation and familiarize users as to the relative CPU consumption of various CICS commands.



ICPU is a user-friendly, main menu, PFKey driven system. ICPU operation includes the following options:

- Start Inspection – Used to start an Inspection and set the run time, slice size, and other Inspection parameters.
- Start Exception Inspection - Used to start an Exception Inspection and set the exception CPU threshold and check interval and other Inspection parameters
- Inspections List – Used to display the sampling results of a list of Inspections each one with its highest CPU usage program and slice.
- Inspection Inquiry – Used to display the results of any Inspection, whether active or on file.
- Terminate Inspection – used to terminate an active Inspection and record the Inspection results.
- Delete Inspection – used to delete an Inspection from the ICPU file.
- Customization – used to define system parameters, such as Slice Size, Sample Interval, and Run Time.
- Set Password – used to set ICPU passwords.

A picture is worth a thousand words. The following three ICPU screens illustrate ease of use.

## Inspections List Screen

The Inspections List screen below shows previously executed Inspections by date. Each Inspection on the list highlights the program and the slice within that program using the highest CPU during that Inspection. From this list you can select a specific Inspection and drill down for the full sampling results.

CICSTEST			ICPU						MAP0030		
			Inspections List						PAGE: 0001		
Command: __											
Line Commands: S&I = Inquiry, T=Terminate, D=Delete											
<u>LC</u>	<u>Userid</u>	<u>Inspection</u>	<u>S</u>	<u>T</u>	<u>Start</u>	<u>Start</u>	<u>Run</u>	<u>Highest</u>	<u>Highest</u>		
		<u>Id</u>			<u>Date</u>	<u>Time</u>	<u>Time</u>	<u>Program</u>	<u>CPU%</u>	<u>Slice</u>	<u>CPU%</u>
_	ICPU	DEMO1	A	R	12/26/14	09:12:27	08:00:00	TEST0000	78.7	002AF460	12.3
_	ICPU	EXC00012	A	E	12/26/14	09:31:55	08:00:00	TEST0000	58.4	0000C860	14.3
_	ICPU	EXC00011	F	E	12/26/14	09:30:55	00:01:00	NATAPP01	78.7	2010	44.5
_	ICPU	EXC00009	F	E	12/26/14	09:28:55	00:02:00	TEST1000	71.1	000060A0	50.0
_	ICPU	EXC00008	F	E	12/26/14	09:27:55	00:01:00	TEST0000	73.5	002AF460	11.3
_	ICPU	EXC00007	F	E	12/26/14	09:26:55	00:01:00	TEST1000	90.7	00002E20	50.3
_	ICPU	EXC00006	F	E	12/26/14	09:25:55	00:01:00	NATAPP02	70.1	2920	33.3
_	ICPU	EXC00005	F	E	12/26/14	09:24:55	00:01:00	TEST2000	73.5	0000BA40	11.3
_	ICPU	EXC00004	F	E	12/26/14	09:23:55	00:01:00	TEST1000	80.4	000068C0	48.6
_	ICPU	EXC00003	F	E	12/26/14	09:22:55	00:01:00	TEST0000	98.5	000D3C60	08.3
_	ICPU	EXC00002	F	E	12/26/14	09:21:55	00:01:00	TEST0000	73.6	002AF460	27.2
_	ICPU	EXC00001	F	E	12/26/14	09:20:55	00:01:00	TEST0000	73.5	002AF460	26.3
S=Status: A=Active, F=Filed, P=Pending   T=Type: R=Regular, E=Exception											
KEYS: ENTER=Refresh, 7=Prev, 8=Next											



When the drill down is on a Regular Inspection, the Inspection Results Summary screen is displayed (1st of the two screens below). When the drill down is on an Exception Inspection, the Exception Results Summary screen is shown (2nd of the two screens below). There are similarities and differences between the two as we will see.

## Inspection Results Summary Screen

When drilling down in on a Regular Inspection, the Inspection Results Summary screen is displayed. The screen lists programs in descending order by CPU usage, and for each program, a list of slices, in descending order by CPU usage.

**CICSTEST**

**ICPU**

**MAP0070**

**Inspection Results Summary**  
**All TCBs**

**Command:** \_\_\_\_

**PAGE:** 01 **OF:** 05

**Userid:** ICPU    **Inspection Id:** DEMO  
**Status:** Active    **Start:** 12/29/14 07:12:27  
**End:** 12/29/14 15:12:27

**Run Time:** 03:13:25  
**Issued:** 12/29/14 07:12:27  
**Sample Interval:** 010

**Slice size:** 0032  
**Repeat:** N 007

**Inspection Filters:**  
**Userids:** \_\_\_\_\_  
**Pgmids:** \_\_\_\_\_  
**Tranids:** \_\_\_\_\_  
**Termids:** \_\_\_\_\_

**DB2: 85.9 AS: 4.2 TS: 3.9 SP: 2.8 SPL: 2.7 PC: 1.8 FC: .6 SC: .6 OT: .2**

Program Name	Lang/Conn	Hits	Appl	CICS	Osrsv	Reqs	QR CMDS	No of TCB	No of SW Slices	Slice-CPU% (01)
NATNUC01	AS/Q	46,651	4.3	15.2	4.7	284,831	35,051	52,926	20	002AF460- 16.4
NATAPP01	NAT	26,602	2.8	7.9	3.1				20	2010 - 12.3
TST20000	CO/T	17,927	2.1	4.3	2.9	19,123	3,817	3,204	10	00000460- 62.3
TST30000	CO/Q	11,759	1.8	2.3	2.0	8,368	8,368	0	03	00004C58- 12.1
===Totals===		192,771	29.7	43.9	26.4	486,716	46,702	60,474		

**KEYS: ENTER=Refresh, 2=SW TCB, 4=BWD, 5=FWD, 7=Prev, 8=Next, 10=Left, 11=Right**

Four programs are visible in the list above. To see additional programs in the list the user hits the PF5 key to move forward. The "Totals" line at the bottom of the list of programs give the total CPU breakdown for the Inspection across all programs including those not currently visible in the list. In our example above, the Totals line shows that across all programs executing during the Inspection, 29.7% of the CPU time was used by application code, 43.9% was used by CICS and 26.4% was attributed to other services.

The CICS services breakdown line (the line above the list of programs) shows that 85.9% of the services were DB2 requests, and 4.2% of the services were CICS Assign commands. Examining the Natural nucleus program NATNUC01 (first line in the list of programs), we see that the application code used 4.3% of the CPU time, CICS modules servicing application requests used 15.2% of the CPU time, and other system services, such as Language Environment, used 4.7% of the CPU time. The Natural statement "2010" used 12.3% of the CPU time used by NATAPP01.





## Exception Results Summary Screen

When the drill down is on an Exception Inspection, the Exception Results Summary screen is shown. Just as with our Inspection Results Summary, the Exception Results Summary screen lists programs in descending order by CPU usage, and for each program, a list of slices, in descending order by CPU usage.

CICSTEST
ICPU
MAP0070

Exception Results Summary

Command: \_\_\_\_
All TCBs
PAGE: 01 OF: 01

**Userid: ICPU**  
**Status: Filed**

**Inspection Id: EXC00010 G**  
**Start: 12/26/14 09:22:35**  
**End: 12/26/14 09:24:47**

**Run Time: 00:02:12**  
**Issued: 12/26/14 09:20:35**  
**Sample Interval: 010**

**Slice size: 0032**  
**Repeat: N 007**

**Inspection Filters:**  
**Userids:**  
**Pgmids:**  
**Exception Criteria: CPU Threshold: 65% Check Interval: 010 Recording Limit: 0015**

**DB2: 27.7 SC: 20.7 KC: 8.4 AS: 6.9 SP: 5.9 TD: 5.6 TS: 4.5 FC: 4.2**

Program Name	Lang /Conc	Hits	CPU%			QR	No of	No of	Slice-CPU%
			Appl	CICS	Osrsv	Reqs	CMDs	TCB SW	Slices (01)
NATNUC01	AS/Q	1,505	24.6	24.5	19.6	3,238	398	601	15 002AF460- 12.3
NATAPP01	NAT	519	4.4	15.4	3.9				12 0800 - 16.4
TST20000	CO/T	92	.5	2.3	1.4	217	43	36	03 00000060- 79.4
TST30000	CO/Q	74	.2	1.7	1.5	95	95	0	11 00004C58- 12.1
===Totals===		2,191	29.7	43.9	26.4	5,534	531	687	

**KEYS: 2=SW TCB, 4=BWD, 5=FWD, 6=First, 7=Prev, 8=Next, 10=Left, 11=Right**

This example above shows that during the 60 seconds exception Check Interval ended at 09.24.47, the program NATNUC01 was using 68.7% (24.6% + 24.5% + 19.6%) of the CPU used by all applications, which exceeds the 65% threshold set for this Inspection.

When an exception occurs ICPU invokes a transaction which notifies about the event as soon as it occurs. This feature is completely customizable and can be modified to any desired user logic. The supplied program writes the three-line message below:

Inspection Id	User Id	Exception Date	CICSTEST Exception Time	ICPU Exception Event DD/LIB	Program/CSECT Name	Lang	CPU%	Slice Offset	CPU%
SAMPLE	ICPU	05/01/18	12.23.34	ICPULOAD	TEST0000	COB	71.1	00000160	50.0





## Summary

By virtue of its ease of use and low overhead, ICPU can be used to identify CPU consumption issues in your CICS applications across all phases of the application lifecycle - from development through production. A system with tuned applications is more stable, causes fewer problems and can help delay expensive capacity upgrades.

Of course, upgrading your CPU can solve CPU related performance problems, but with ICPU you can postpone CPU upgrades, reduce costs and improve response time and overall performance.

ICPU can be installed in less than an hour; it uses standard CICS and MVS services and is available for all versions of CICS TS.

Tuning CICS is most often accomplished by trading off resources. With ICPU you are in a win-win situation – you can improve the efficiency of your online applications without giving up anything.