CICS Integration & Optimization: Tales from the Trenches

Or… let’s save some MIPS!

Russ Teubner
HostBridge Technology
russ@hostbridge.com
CICS users are loyal to their apps – and for good reason! However, they also need to integrate these same applications with an ever widening array of web and cloud-based resources. If that weren’t enough, every year they are under pressure to add value, support new workload and reduce the cost of ownership. That’s a tall order. This session will highlight two customers who used a common tactic to enhance the value of their existing CICS investments.
Customer Case Studies

- Customer A
  - Industry: Telecommunications (US)
  - Very high daily/consistent transaction volume
  - Long-standing investment in COBOL-based socket apps

- Customer B
  - Industry: Financial Services (International)
  - Very high transaction volume on one day each month (and in compressed time period)
  - Long-standing investment in PL/I-based socket apps
Common Objectives

- Both customers had common objectives
- Business Objectives
  - Respond to competitive pressures in their industry
  - Lower incremental cost of high-volume CICS application processing (i.e., marginal value > marginal cost)
  - Move new/additional workload to System z and reinforce CICS TS as the most cost effective platform for their business
- Technical Objective (at least their hope)
  - Streamline System z and CICS integration paths
  - Reduce the CPU burn (GP) associated with socket applications and infrastructure
Perfect R&D Situations

- Well defined business objectives
- An initial theory as to what the technical issues might be
- Strong in-house CICS talent
- Load testing infrastructure in place
- Good CICS tools on hand
- Test LPAR/region available
- Had a spare cubicle
Timing Was Opportune

- Customers were continuing to state their concern about doing more for less
- We had just delivered zIIP-enabled versions of our products, and our heads were filled with fun facts related to:
  - z/OS, USS, LE, WLM, SRBs, zIIP
  - CICS TS v4 Open Transaction Environment
  - Sockets
- Other factors:
  - We are zealots regarding integration of CICS apps/data as part of web/cloud-based infrastructure
  - We are committed to delivering functionality under CICS
  - I didn’t want to stop writing code (zIIP project was too fun)
Cut to the Chase

- What we learned was surprising and the results were unexpected (in a good way)
- We ended up exploiting CICS TS v4 OTE and z/OS to create a solution
- I want this to be knowledge you can use:
  - The approach is generally applicable to any CICS customer who has socket apps
  - The higher your volume, the more it matters
- Yes… I’m “a vendor” but please forget that for now – I’m speaking as a CICS developer
Customer A - Initial Conditions

- Typical architecture for CICS-based socket listener/applications
- Persistent connection between Gateway and RX/TX transactions
- Multiple simultaneous Gateway-to-CICS connections
- Volume was VERY HIGH!
Research Focus

- EZASOKET application design patterns, performance, APIs
- CICS Socket Listener design patterns
- CICS Socket Def/Mgmt patterns
- CICS TS v4 OTE exploitation
- z/OS USS exploitation
CICS Socket Support

- Provided as part of z/OS Communications Server
- What it includes:
  - Socket APIs
    - C language API
    - Sockets Extended API (aka, EZASOKET or EZACICSO)
    - Original COBOL API (aka, EZACICAL)
  - Listeners: standard and enhanced (i.e., CSKL); or user-written
  - Definition and management components (e.g., EZAO)
- A well-documented workhorse, but…
- It’s been around a long time (circa 1992)
- Older than CICS OTE
  - Thus… much of it’s original architecture
- Reengineered to support OTE
  - But… the general approach of the original architecture persisted

Thus, I’m NOT referring to CICS TS features which use the CICS Sockets Domain.
Our focus is here…

A CICS Sockets transaction has direct access to the TCP/IP socket and can issue native sockets calls to receive and send data over the socket.

These services are based on the Sockets Extended sockets APIs (provided by Communications Server).

A conversational model - or a request/reply model

A request/reply model

The listeners are the 'servers' as seen from a TCP/IP perspective.

These services are based on the UNIX System Services C/C++ sockets API (provided by Language Environment) and the UNIX System Services callable APIs.

A CICS Sockets Domain transaction does not have direct access to the socket, but communicates with CICS Sockets Domain services to receive a request and to send a reply over a socket.
CICS Sockets Pathway

CICS Sockets Support

CICS Sockets Domain

Application Programs and Subsystems

Pascal API

REXX Sockets

Sockets Extended Call API

Sockets Extended Assembler Macro API

CICS sockets

IMS Sockets

XTI RFC1006

SUN RPC 3.9

NCS RPC

X-Windows System X11R4

SNMP DPI 1.2

XTI XPG4.2

SUN RPC 4.0

DCE RPC

SNMP DPI 2.0

X-Windows System X11R6

z/OS CS TCP/IP C/C++ Sockets

Language Environment (UNIX) C/C++ Sockets

z/OS UNIX System Services Callable BPX Sockets

TCP and UDP Transport Protocol Layer

IP Network Protocol Layer

Network Interface Layer

z/OS Communications Server, IP Sockets Application Programming Interface Guide and Reference
Test Methodology

- Two test harnesses used for comparison
- z/OS-based testing is quick and good for functionality, but not fair for performance (hyper-sockets is too good)
Standard Test Cycle

- Each test cycle caused the gateway to:
  - Open 2 sockets via Listener TX
  - Send/Receive TXs started to handle socket I/O
  - Generate 2,500 request-response iterations (no delays)
  - Each request caused a LINK to a customer program
  - Bytes in/out modeled for average production use case

- Benchmarks run:
  - 1 concurrent test cycle
  - 5 concurrent test cycles (10 sockets and 12,500 iterations)

- Objectives:
  - Measure region-level CPU burn for various configurations
  - Differentiate between CPU burn associated with Socket apps and Socket infrastructure

Selected to keep total region-level CPU use to a manageable level on test LPAR
Tooling Developed

- It’s difficult to get a snapshot of a CICS region’s total resource consumption that is:
  - high-resolution (microseconds)
  - low-overhead
  - Immediate
  - Includes zIIP and zAAP

- Ended up developing two tools:
  - A CICS transaction to provide a summary of MVS ASSB timers (HBZT)
  - A CICS XMNOUT exit to log transaction metrics via WTO

- The combination allowed us to:
  - drive testing fast
  - quickly assess results from all angles

- Special thanks to:
  - Larry Lawler (UNICOM)
  - Ed Jaffe (Phoenix Software)

- For info on HBZT, see me after session
CPU Measurement (HBZT)

ACTUAL mode upon entry

CPU USAGE FOR ADDRESS SPACE: ASID=003F,APPLID=CICSA

ACTUAL values at 2012/07/31 23:39:06.068080

ASSB 'Programming Interface' values (*=not normalized):

ASSBASST................. 00:00:00.000000  Additional SRB Service Time
ASSBPHTM.................. 00:00:00.385582  Preemptable-class SRB Time
ASSBPHTM_BASE............. 00:00:00.000000  ASSBPHTM at end of previous jobstep
ASSB_IFA_PHTM............. 00:00:00.000000  zAAP-only equiv of ASSBPHTM
ASSB_ZIIP_PHTM............ 00:00:00.378829  zIIP-only equiv of ASSBPHTM
ASSB_SRBB_TIME_ON_CP.... 00:00:00.288598  CP time in SRB mode
ASSB_TASK_TIME_ON_CP..... 00:00:02.473032  CP time in task mode
ASSB_TIME_IFA_ON_CP....... 00:00:00.000000  zAAP time on CP (non-enclave)
ASSB_TIME_ZIIP_ON_CP...... 00:00:00.000000  zIIP time on CP (non-enclave)
ASSB_TIME_ON_IFA......... 00:00:00.000000  zAAP time (non-enclave)
ASSB_TIME_ON_ZIIP........ 00:00:00.000000  zIIP time (non-enclave)

Other ASSB values of interest:

ASSB_ENCT................ 00:00:00.000000  Std CP time (enclave)
ASSB_IFA_ENCT............. 00:00:00.000000  zAAP time (enclave)
ASSB_ZIIP_ENCT............ 00:00:00.378829  zIIP time (enclave)

This program may be freely copied and used in object code form.  
Copyright (c) 2011 HostBridge Technology, LLC -- www.hostbridge.com  
ENTER=Update, PF1=Baseline, PF2=Toggle Mode, PF5=Update+Baseline, CLEAR=Exit
CPU Measurement (HBZT)

PF2 toggles mode

Immediate view of ASSB values
### CPU Measurement

#### CPU Usage

**CPU Usage for Address Space:** ASID=003F, APPLID=CICSA

**Delta Values from 2012/08/01 00:13:49.306914 to 2012/08/01 00:13:49.306914**

<table>
<thead>
<tr>
<th>ASSB 'Programming Interface' Values (not normalized)</th>
<th>Additional SRB Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSBASST</td>
<td>Preemptable-class SRB Time</td>
</tr>
<tr>
<td>ASSBPHTM</td>
<td>ASSBPHTM at end of previous jobstep</td>
</tr>
<tr>
<td>ASSBPHTM_BASE</td>
<td>zAAP-only equiv of ASSBPHTM</td>
</tr>
<tr>
<td>ASSB_IFA_PHTM</td>
<td>zIIP-only equiv of ASSBPHTM</td>
</tr>
<tr>
<td>ASSB_ZIIP_PHTM</td>
<td>CP time in SRB mode</td>
</tr>
<tr>
<td>ASSB_SRB_TIME_ON_CP</td>
<td>CP time in task mode</td>
</tr>
<tr>
<td>ASSB_TASK_TIME_ON_CP</td>
<td>zAAP time on CP (non-enclave)</td>
</tr>
<tr>
<td>ASSB_TIME_IFA_ON_CP</td>
<td>zIIP time on CP</td>
</tr>
<tr>
<td>ASSB_TIME_ZIIP_ON_CP</td>
<td>zAAP time (non-enclave)</td>
</tr>
<tr>
<td>ASSB_TIME_ON_ZIIP</td>
<td>zIIP time (non-enclave)</td>
</tr>
</tbody>
</table>

**Other ASSB Values of Interest:**

- ASSB_ENCT: Std CP time (enclave)
- ASSB_IFA_ENCT: zAAP time (enclave)
- ASSB_ZIIP_ENCT: zIIP time (enclave)

---

**Notes:**

- **PF1 resets baseline**
- **All delta values now zero**

---

*This program may be freely copied and used in object code form. Copyright (c) 2011 HostBridge Technology, LLC -- www.hostbridge.com*

ENTER=Update, PF1=Baseline, PF2=Toggle Mode, PF5=Update+Baseline, CLEAR=Exit
CPU Measurement

Run load test and press ENTER

Immediate view of ASSB values (deltas)
CPU Measurement

CPU USAGE FOR ADDRESS SPACE: ASID=0155, APPLID=CTORP7

ACTUAL values at 2013/02/05 01:25:19.880228

ASSB 'Programming Interface' values:
ASSBASST................. 00:00:00.886150  Additional SRB (accessing an SRB class)
ASSBPHTM................. 05:45:53.138314  Preemptable-critical task (non-enclave)
ASSBPHTM_BASE............ 00:00:00.000000  ASSBPHTM at entry (entry-enclave)
ASSB_IFA_PHTM............ 00:00:00.000000  zAAP-only equivalent of ASSBPHTM
ASSB_SRB_TIME_ON_CP..... 00:44:57.171445  CP time in SRB mode
ASSB_TASK_TIME_ON_CP.... 20:22:02.621241  CP time in task mode
ASSB_TIME_IFA_ON_CP...... 00:00:00.000000  zAAP (non-enclave)
ASSB_TIME_ON_IFA......... 00:00:00.000000  zAAP (non-enclave)
ASSB_TIME_ON_ZIIP........ 00:00:00.665266  zIIP time (non-enclave)
ASSB_TIME_ZIIP_ON_CP..... 00:00:00.086624  zIIP time on CP (non-enclave)
ASSB_ZIIP_PHTM........... 05:33:11.619545  zIIP-only equiv of ASSBPHTM

Other ASSB values of interest:
ASSB_ZIIP_ENCT........... 05:33:10.954278  zIIP
ASSB_ENCT................ 00:00:00.000000  Start

This program may be freely copied and used in object code form.
Copyright (c) 2011 HostBridge Technology, LLC -- www.hostbridge.com
ENTER=Update, PF1=Baseline, PF2=Toggle Mode, PF5=Update+Baseline, CLEAR=Exit
Where the Data Led Us

- Under volume testing, the CPU burn associated with the CICS Sockets Support was measurable and linear (confirmed customer’s theory)
- I won’t characterize it as “high” or “low” because the only thing that mattered was whether it could be lower (or not so linear)
- Thus, we began to:
  - Isolate various components and their impact
  - Consider how to provide alternative functionality (but complimentary to CICS TS)
- Low hanging fruit seemed to be CICS Socket Handler (via EZASOKET API)
Customer A - Solution 1

- Leverage EZASOKET API as established design pattern
- Replace CICS Socket Handler
- Keep CICS Socket Definition/Management
- Exploit CICS TS v4 OTE, z/OS, USS
- EZASOKET apps must be defined as THREADSAFE & OPENAPI
Solution 1 Assessment

❖ Good...
  ▪ The Alt. Socket Handler lowered GP CPU burn associated with Socket I/O
  ▪ All it required was a re-link of apps that used EZASOKET API (with alternate load module)
  ▪ Transparent to existing user-written Listeners, Sender and Receiver TXs

❖ However...
  ▪ EZASOKET API emulation seemed to be a bit of needless overhead (e.g., parameter marshaling and transformation)
  ▪ zIIP enablement opportunity wasn’t optimal due to task switching

❖ But wait...
  ▪ The design patterns for CICS-based Listeners, Receivers and Senders are fairly common
Customer A - Solution 2

- Replace Listener, Receive, Send TX with equivalent/generic alternatives
- Eliminate EZASOKET API as a design pattern
- Keep CICS Socket Definition/Management
- Exploit CICS TS v4 OTE, z/OS, USS, zIIP
Solution 2 Assessment

- Very Good…
  - GP CPU burn associated with Socket I/O went down further
  - EZASOKET API emulation eliminated (all components use native sockets)
  - Transparent to the customer’s applications
  - CICS Socket definition/management leveraged
    - EZAO still used to Configure, Start, or Stop Listeners
- zIIP enablement potential maximized
  - Minimal task switching
  - Customer application code not zIIP enabled (per IBM-ISV T&C’s)
Pathway - Old vs. New

BEFORE: CICS Sockets Support

AFTER: Alt. Sockets Support

z/OS Communications Server, IP Sockets Application Programming Interface Guide and Reference

Copyright © 2013 HostBridge Technology
## Test Results

**Standard Socket Infrastructure (EZA-based)**

<table>
<thead>
<tr>
<th></th>
<th>Send TX (GP)</th>
<th>Recv TX (GP)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140714</td>
<td>332702</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>138355</td>
<td>317988</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>141509</td>
<td>336017</td>
<td></td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>140193</strong></td>
<td><strong>328902</strong></td>
<td><strong>469095</strong></td>
</tr>
</tbody>
</table>

**Alt. Socket Infrastructure (ziip=n)**

<table>
<thead>
<tr>
<th></th>
<th>Alt Send TX (GP)</th>
<th>Alt Recv TX (GP)</th>
<th>% reduction - Old vs. New w/o zIIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128676</td>
<td>285711</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>125736</td>
<td>271014</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>119938</td>
<td>240784</td>
<td></td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>124783</strong></td>
<td><strong>265836</strong></td>
<td><strong>390620 -17%</strong></td>
</tr>
</tbody>
</table>

**Alt. Socket Infrastructure (ziip=y)**

<table>
<thead>
<tr>
<th></th>
<th>Alt Send TX (GP)</th>
<th>Alt Recv TX (GPU)</th>
<th>% reduction - Old vs. New w/o zIIP</th>
<th>% reduction - Old vs. New w/ zIIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94956</td>
<td>48131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>94766</td>
<td>48759</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>94049</td>
<td>47391</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>94522</td>
<td>47390</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>94573</strong></td>
<td><strong>161630</strong></td>
<td><strong>256203 -34%</strong></td>
<td><strong>-45%</strong></td>
</tr>
</tbody>
</table>

All times in microseconds
# Test Results (w/ Concurrency)

## Standard Socket Infrastructure (EZA-based)

<table>
<thead>
<tr>
<th>Send TX (GP)</th>
<th>Recv TX (GP)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>609880</td>
<td>1226658</td>
</tr>
<tr>
<td>2</td>
<td>614881</td>
<td>1234086</td>
</tr>
<tr>
<td>3</td>
<td>617669</td>
<td>1259704</td>
</tr>
<tr>
<td>Avg</td>
<td>614143</td>
<td>1240149</td>
</tr>
</tbody>
</table>

## Alt. Socket Infrastructure (ziip=n)

<table>
<thead>
<tr>
<th>Alt Send TX (GP)</th>
<th>Alt Recv TX (GP)</th>
<th>% reduction - Old vs. New w/o zIIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>491684</td>
<td>782429</td>
<td></td>
</tr>
<tr>
<td>496651</td>
<td>780384</td>
<td></td>
</tr>
<tr>
<td>502901</td>
<td>804619</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>789144</td>
<td>1286223 -31%</td>
</tr>
</tbody>
</table>

## Alt. Socket Infrastructure (ziip=y)

<table>
<thead>
<tr>
<th>Alt Send TX (GP)</th>
<th>Alt Recv TX (zIIP)</th>
<th>Alt Recv TX (GP)</th>
<th>Alt Recv TX (zIIP)</th>
<th>% reduction - New w/o zIIP to New w/ zIIP</th>
<th>% reduction - Old vs. New w/ zIIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>417841</td>
<td>198962</td>
<td>657107</td>
<td>424739</td>
<td>19%</td>
<td>44%</td>
</tr>
<tr>
<td>417388</td>
<td>194910</td>
<td>613641</td>
<td>401113</td>
<td></td>
<td></td>
</tr>
<tr>
<td>409281</td>
<td>194758</td>
<td>618252</td>
<td>399555</td>
<td></td>
<td></td>
</tr>
<tr>
<td>410077</td>
<td>193542</td>
<td>600015</td>
<td>397736</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>413647</td>
<td>622254</td>
<td>1035901 -19%</td>
<td></td>
<td>-44%</td>
</tr>
</tbody>
</table>

The TCP/IP stack seems to get more efficient the harder you load it.

Your Mileage May Vary

All times in microseconds

TEST: concurrent instances=5; total requests=12500

% reduction - Old vs. New w/o zIIP

% reduction - New w/o zIIP to New w/ zIIP

% reduction - Old vs. New with zIIP
Customer B - Initial Conditions

(Infrastructure outside System z similar to customer A)

- A single socket is used for both sending and receiving
- CICS Socket Listener connection requests
- Handler TX validates and categorizes requests
- Worker TX is long-lived
- Work requests serviced by LINKed-to programs
Replace CICS Socket Listener
Leverage EZASOKET API, but change implementation
Keep CICS Socket Definition/Management
Exploit CICS TS v4 OTE, z/OS, USS
Programs defined as THREADSAFE & OPENAPI
Nice GP CPU reduction, but no practical opportunity to exploit zIIP
Customer B - Solution 2

- Builds on Solution 1
- Replace Worker with generic alternative
- Eliminate EZASOKET API in Worker TX
- Allows zIIP exploitation by Worker
Each configuration was tested using a common benchmark from a distributed system:
- Open socket to Worker TX (via Listener/Handler)
- Send 5,000 requests (causing the same number of LINKs and responses)
- Close socket

Test constructed to isolate the actual GP CPU costs/savings for socket-related processing per request

Test not constructed to determine an average percent reduction in GP CPU per request
## Test Results and Calculations

<table>
<thead>
<tr>
<th></th>
<th>GP CPU Sec</th>
<th>zIIP CPU Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Config</strong></td>
<td>27.951</td>
<td>0.000</td>
</tr>
<tr>
<td><strong>Solution 1</strong></td>
<td>26.943</td>
<td>0.000</td>
</tr>
<tr>
<td><strong>Solution 2</strong></td>
<td>23.436</td>
<td>0.427</td>
</tr>
</tbody>
</table>

### Averages from multiple tests of each configuration

- **27.951** Initial Config runs entirely on the GP (all socket I/O and app logic)
- **(26.943)** Solution 1 runs entirely on the GP, but measures the effect of replacing the EZASOKET API
- **1.008** Difference = Estimated GP CPU savings to handle socket I/O for 5,000 requests via Customer Worker TX

- **27.951** Initial Config runs entirely on the GP (all socket I/O and app logic)
- **(23.436)** Solution 2 runs socket I/O on the zIIP, and app logic on GP
- **4.515** Difference = Estimated GP CPU savings to handle socket I/O for 5,000 requests via Alternate Worker TX (and on zIIP)
What mattered most to the customer was processing new workload efficiently during their peak 4 hour period

Assume:

- 5 million TX in max 4 hr period
- 20% processed via Alt. Worker TX

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak 4 hour transaction volume</td>
<td>5,000,000</td>
</tr>
<tr>
<td>% of TX processed via Alt. Worker</td>
<td>20%</td>
</tr>
<tr>
<td>TX processed via Alt. Worker</td>
<td>1,000,000</td>
</tr>
<tr>
<td>% of TX processed via Std. Worker</td>
<td>80%</td>
</tr>
<tr>
<td>TX processed via Std. Worker</td>
<td>4,000,000</td>
</tr>
<tr>
<td>Est. GP CPU Reduction for Alt. Worker (seconds)</td>
<td>903</td>
</tr>
<tr>
<td>Est. GP CPU Reduction for Std. Worker (seconds)</td>
<td>807</td>
</tr>
<tr>
<td>Total Est. GP CPU Seconds Reduced</td>
<td>1,710</td>
</tr>
<tr>
<td>Total Est. GP CPU Minutes Reduced during Peak Period</td>
<td>28.49</td>
</tr>
</tbody>
</table>
Customer B – Optimal Solution

- Handler modified by customer to categorize requests and START appropriate Worker (trivial change)
- Alternate Worker TX handles high-volume requests
- Eliminates EZASOKET API
- Exploit zIIP for socket I/O

**Alternate Socket Listener**

- Accept
- I/O
- Give

**Customer Handler TX** (With request recognition)

- Take
- I/O
- Give

**Alt. EZASOKET API**

**zOS/USS Socket Services**

**TCP/IP**

**zIIP Enabled**

- Alternate Worker TX (Equivalent Functionality)
  - Take
  - I/O
  - Close

- zOS/USS Socket Services

- TCP/IP

**Customer Worker TX** (Unchanged)

- Take
- I/O
- Close

**Alt. EZASOKET API**

**zOS/USS Socket Services**

**TCP/IP**

- Customer Worker TX used for request types not yet supported by Alternate Worker TX
- Uses Alt. EZASOKET API to achieve some GP CPU savings
Summary

- CICS Socket Support has been a workhorse for a long time -- it’s earned it’s keep!
- CICS TS Open Transaction Environment continues to evolve and permit new opportunities for customers and ISV’s -- thank you Hursley Lab
- An example is the Alt. Socket Support described in this presentation
- This approach is applicable to any customer who relies heavily on CICS Socket Support
  - zIIP support can only be provided by a licensed ISV
- You can substantially reduce GP CPU usage associated with CICS socket applications
- Oh… and the customers were very pleased