Tools and Techniques for Understanding Threading Behavior in Android*

Dr. Ramesh Peri
Sr. Principal Engineer & Architect of Android tools
Intel Corporation
Austin, TX 78746
Email: ramesh.v.peri@intel.com

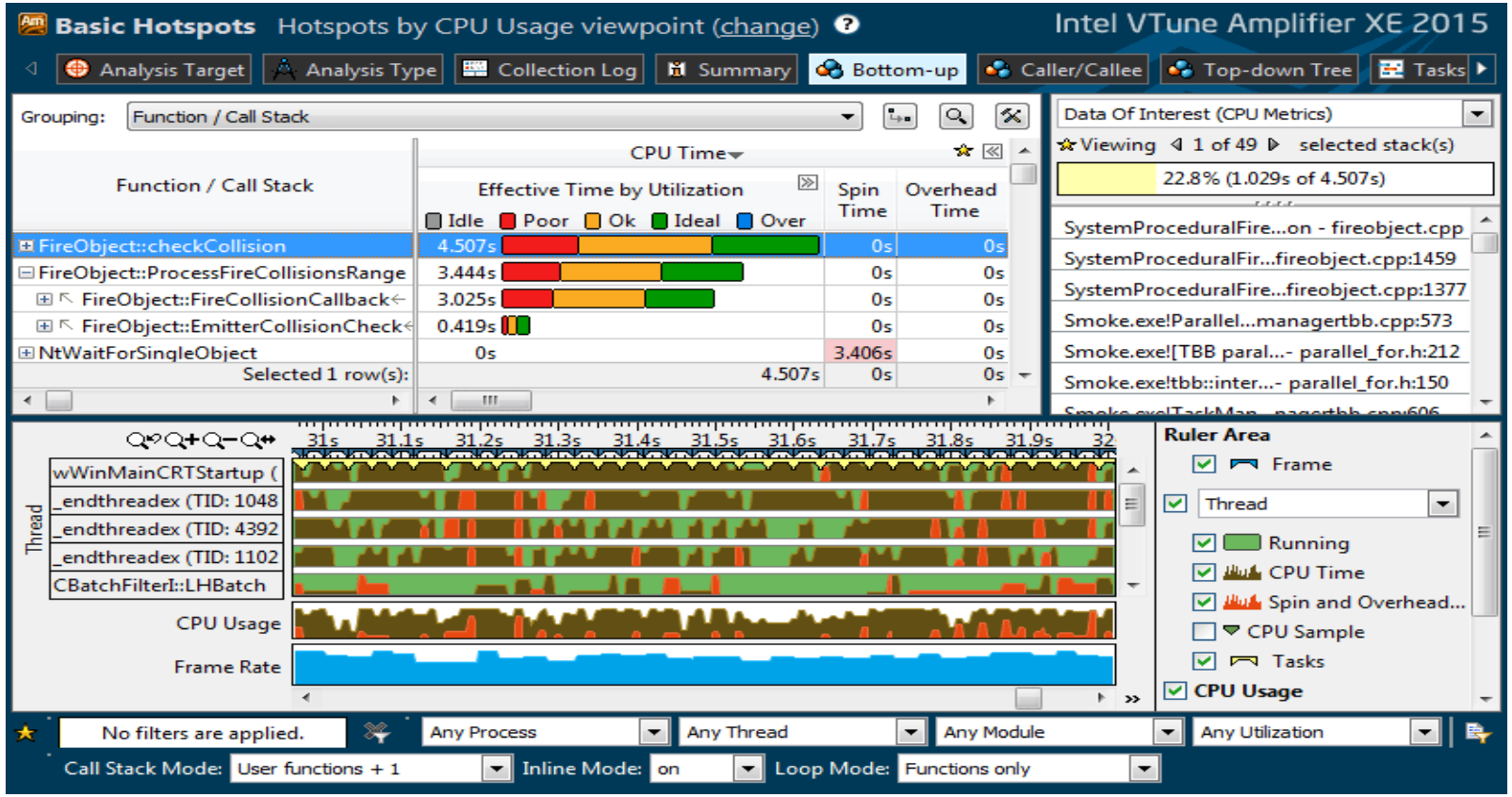
Agenda

- Goals
- Tools Roundup
  - Intel® Vtune™, Linux Perf, Nvidia* System Profiler, Google* Systrace, ARM* DS-5
- Threading Examples
  - Simple Example
  - Simple Threading
  - Communicating Threads
  - Simultaneously executing threads
  - Lazy Thread
  - False Sharing

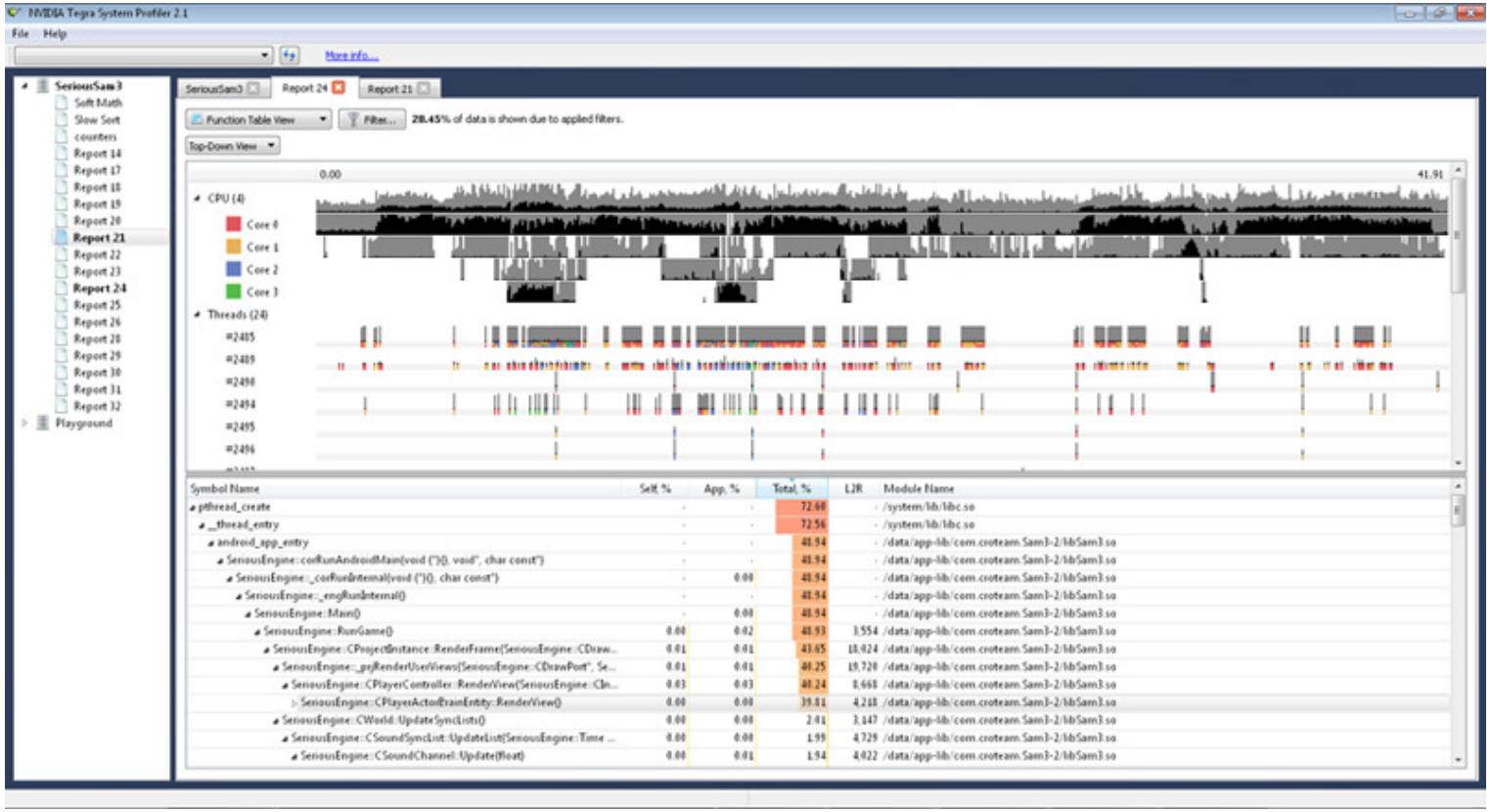
Goals

- Learn about performance analysis tools in Android
- Develop simple micro-benchmarks and run them under the control of a performance analysis tool
  - Interpret and validate the data
- Understand Threading models used in Android
  - Core to thread mapping
  - Has Impact on
    - responsiveness
    - power and
    - performance

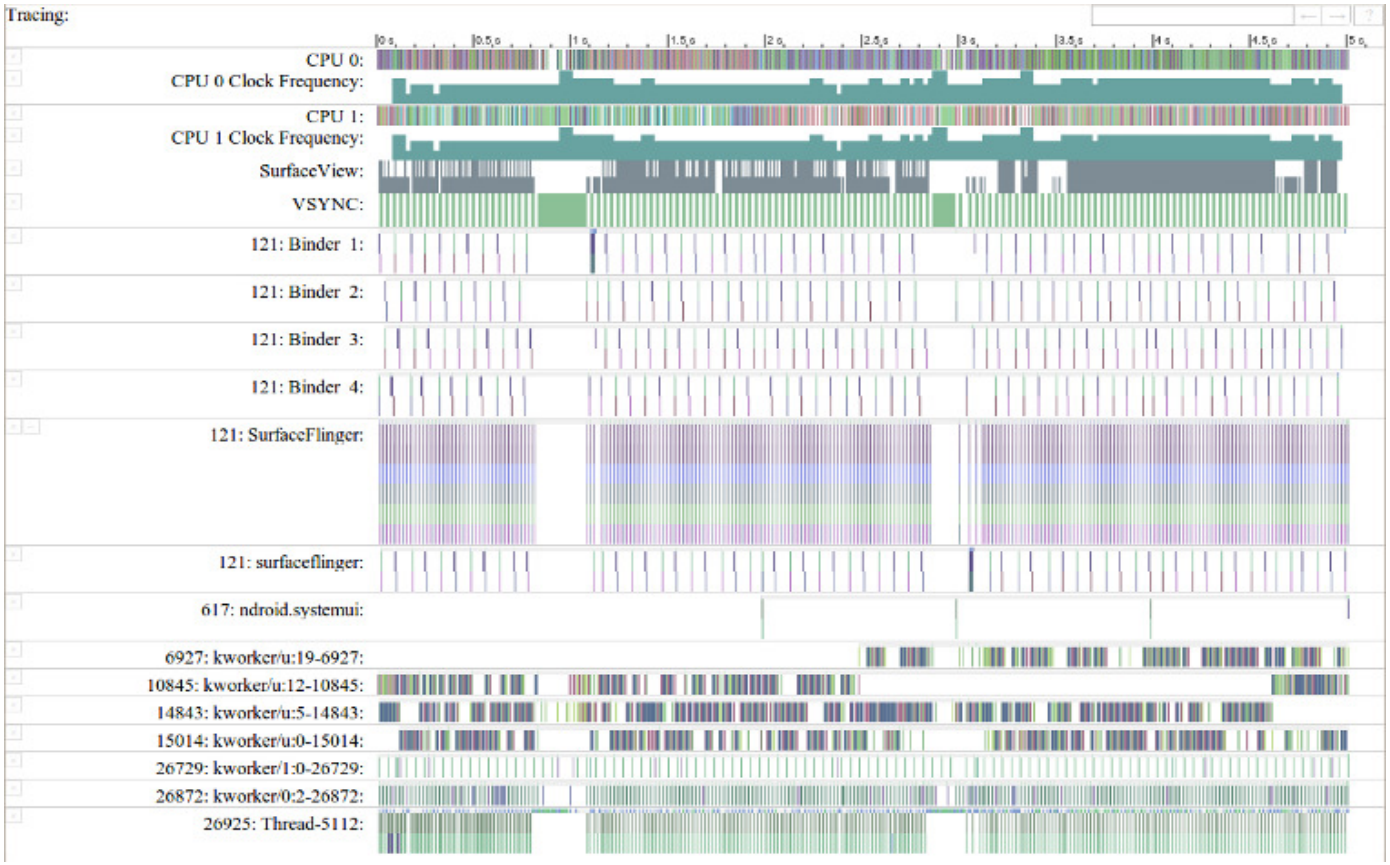
Intel® VTune™


Basic Hotspots
Hotspots by CPU Usage viewpoint (change)
Intel VTune Amplifier XE 2015
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Tasks
Grouping: Function / Call Stack
Function / Call Stack
CPU Time
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Overhead Time
FireObject::checkCollision 4.507s 0s 0s
FireObject::ProcessFireCollisionsRange 3.444s 0s 0s
FireObject::FireCollisionCallback 3.025s 0s 0s
FireObject::EmitterCollisionCheck 0.419s 0s 0s
NtWaitForSingleObject 0s 3.406s 0s
Selected 1 row(s): 4.507s 0s 0s
Data Of Interest (CPU Metrics)
Viewing 1 of 49 selected stack(s)
22.8% (1.029s of 4.507s)
SystemProceduralFire...on - fireobject.cpp
SystemProceduralFir...fireobject.cpp:1459
SystemProceduralFire...fireobject.cpp:1377
Smoke.exe!Parallel...managertbb.cpp:573
Smoke.exe![TBB paral...- parallel_for.h:212
Smoke.exe!tbb::inter...- parallel_for.h:150
31s 31.1s 31.2s 31.3s 31.4s 31.5s 31.6s 31.7s 31.8s 31.9s 32
Thread
wWinMainCRTStartup (
_endthreadex (TID: 1048
_endthreadex (TID: 4392
_endthreadex (TID: 1102
CBatchFilterI::LHBatch
CPU Usage
Frame Rate
Ruler Area
Frame
Thread
Running
CPU Time
Spin and Overhead...
CPU Sample
Tasks
CPU Usage
No filters are applied.
Any Process
Any Thread
Any Module
Any Utilization
Call Stack Mode: User functions + 1
Inline Mode: on
Loop Mode: Functions only

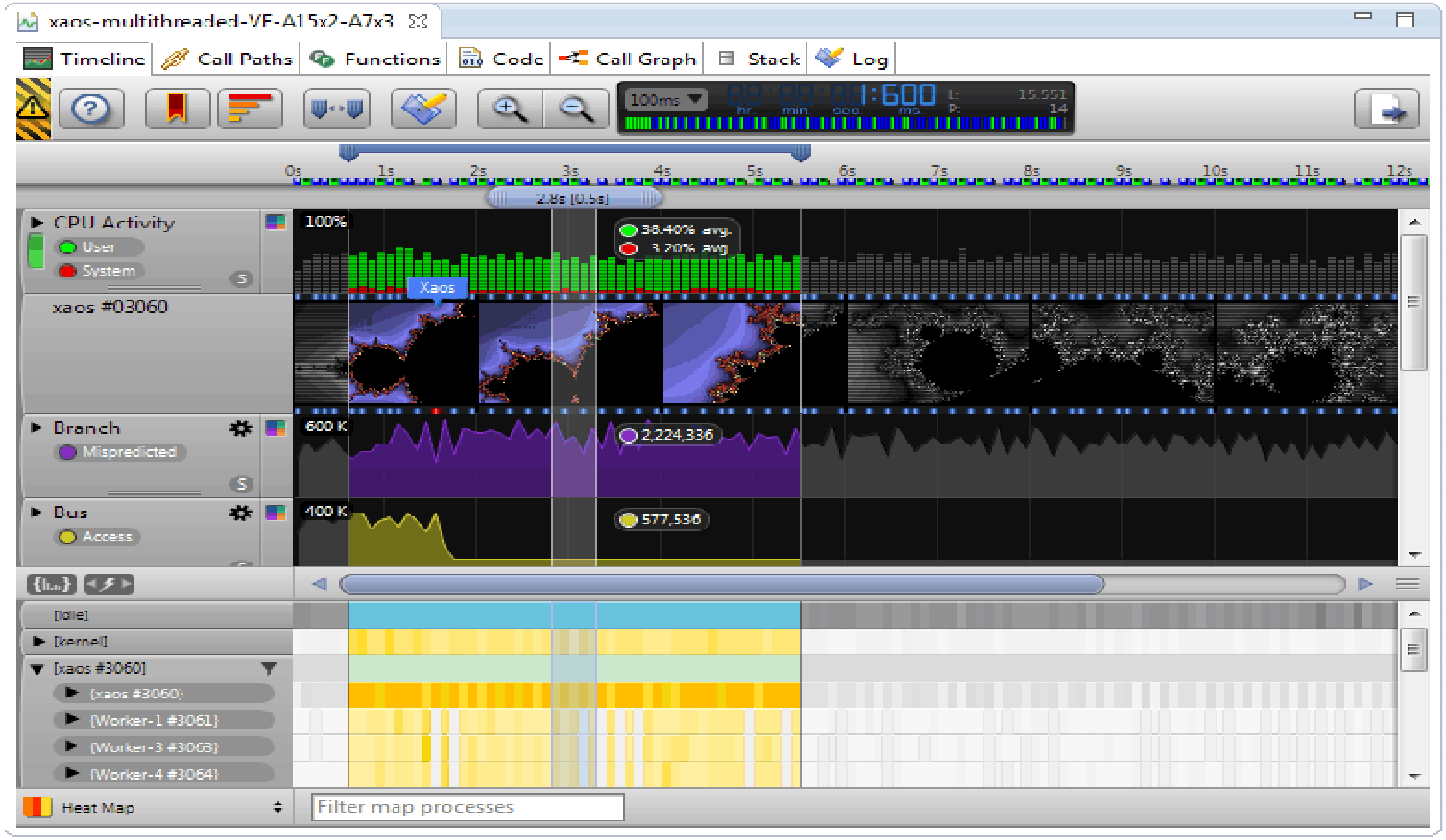
Nvidia* System Profiler



Google* Systrace



ARM* DS-5


Timeline
Call Paths
Functions
Code
Call Graph
Stack
Log
CPU Activity
User
System
xaos #03060
Branch
Mispredicted
Bus
Access
38.40% avg.
3.20% avg.
Xaos
2,224,336
577,536
[idle]
[kernel]
[xaos #3060]
[Worker-1 #3061]
[Worker-3 #3063]
[Worker-4 #3064]
Heat Map
Filter map processes

Linux Perf

Basic commands

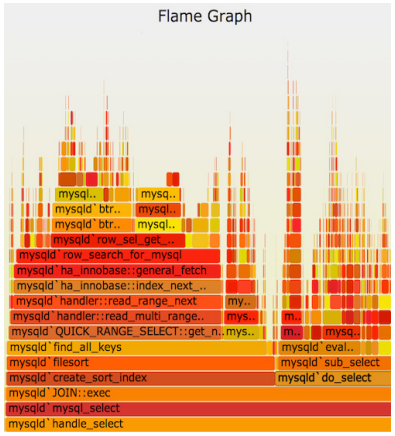
- `perf stat`: obtain event counts
- `perf record`: record events for later reporting
- `perf report`: break down events by process, function, etc.
- `perf annotate`: annotate assembly or source code with event counts
- `perf top`: see live event count
- `perf bench`: run different kernel microbenchmarks

Hotspot

```
perf report

# Events: 1K cycles
#
# Overhead          Command                  Shared Object
                    Symbol
# ........  ...............  ..............................
....................................
#
   28.15%      firefox-bin  libxul.so                        [.] 0xd10b45
    4.45%          swapper  [kernel.kallsyms]               [k] mwait_idle_with_hints
    4.26%          swapper  [kernel.kallsyms]               [k] read_hpet
    2.13%      firefox-bin  firefox-bin                      [.] 0x1e3d
    1.40%  unity-panel-ser  libglib-2.0.so.0.2800.6         [.] 0x886f1
    [...]
```

Annotated Source view

```
 Percent |   Source code & Disassembly of noploop
------------------------------------------------
         :
         :
         :
         :   Disassembly of section .text:
         :
         :   08048484 <main>:
         :   #include <string.h>
         :   #include <unistd.h>
         :   #include <sys/time.h>
         :
         :   int main(int argc, char **argv)
         :   {
    0.00 :    8048484:       55                      push   %ebp
    0.00 :    8048485:       89 e5                   mov    %esp,%ebp
[...]
    0.00 :    8048530:       eb 0b                   jmp    804853d <main+0xb9>
         :                            count++;
   14.22 :    8048532:       8b 44 24 2c             mov    0x2c(%esp),%eax
    0.00 :    8048536:       83 c0 01                add    $0x1,%eax
   14.78 :    8048539:       89 44 24 2c             mov    %eax,0x2c(%esp)
         :            memcpy(&tv_end, &tv_now, sizeof(tv_now));
         :            tv_end.tv_sec += strtol(argv[1], NULL, 10);
         :            while (tv_now.tv_sec < tv_end.tv_sec ||
         :                   tv_now.tv_usec < tv_end.tv_usec) {
         :                    count = 0;
         :                    while (count < 100000000UL)
   14.78 :    804853d:       8b 44 24 2c             mov    0x2c(%esp),%eax
   56.23 :    8048541:       3d ff e0 f5 05          cmp    $0x5f5e0ff,%eax
    0.00 :    8048546:       76 ea                   jbe    8048532 <main+0xae>
[...]
```



kcachegrind

Tools

Feature	Vtune	Linux Perf	Nivida System Profiler	Google Systrace	ARM DS-5
OTB Experience	Hard	Hard	Hard	Easy	Hard
TimeLine	Yes	No	No	Yes	Yes
Java Source	Yes	Limited	No	No	Maybe
Gridview(HotSpot)	Yes	Yes	Yes	No	Yes
h/w events	Yes	Limited	Limited	No	Yes
OS events	Limited	Yes	Limited	Yes	Yes
Filtering	Yes	No	No	No	No
Grouping	Yes	Limited	No	No	Limited
Call Stack	Yes	Yes	Yes	No	Yes
Platform View	Yes	No	No	No	Limited

Devices

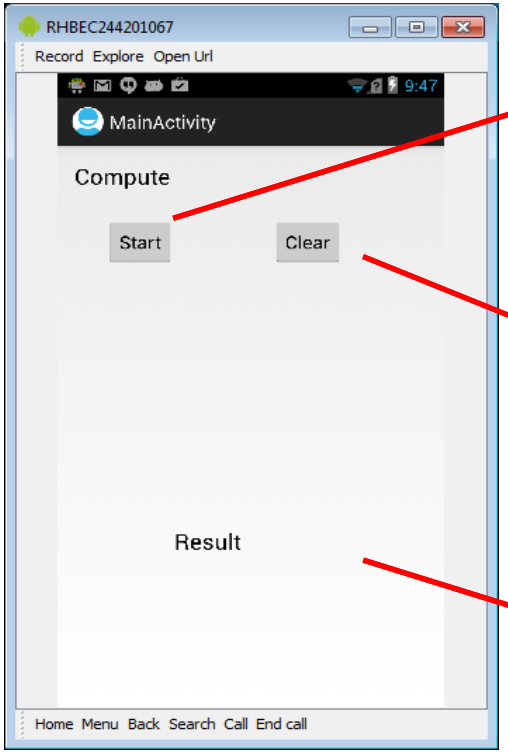
	Nexus 7	Dell Venue 8	Nvidia Shield
Processor	Snapdragon S4	CLTP/Merrifield	Tegra K1
Frequency	1.5Ghz	1.3Ghz	2.2Ghz
Memory	2	1	2
Number of Cores	4	2	4
Android Version	4.3	4.3	4.4
Storage	32	16	32
Manufacturer	Qualcomm	Intel	Nvidia
Display	1920x1200	1920x1200	1920x1200

Device Resources

- Dell Venue 8
  - http://software.intel.com/mdk
  - http://opensource.dell.com
- Nvidia Shield
  - https://developer.nvidia.com/develop4shield
- Nexus 7
  - http://play.google.com

Simple Example

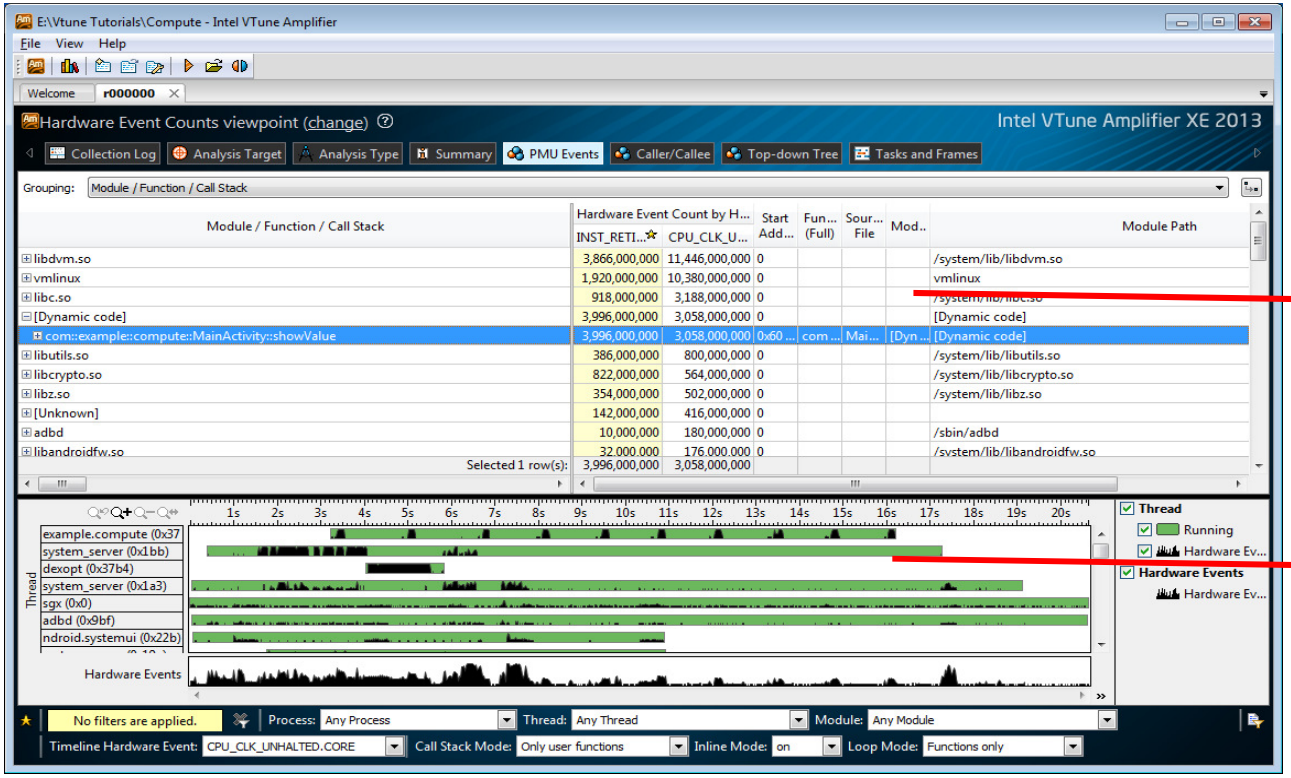
A Simple Example




```
public void showValue(View v) {
  int i, j, sum=0;

  for (i=0;i<10000;i++)
    for (j=0;j<10000;j++)
      sum+=i;

  TextView tv = (TextView)findViewById(R.id.textView2);
  tv.setText(String.valueOf(sum));
}

public void clearValue(View v) {
    TextView tv = (TextView)findViewById(R.id.textView2);
    tv.setText("");
}
```

textview2

Performance Profile

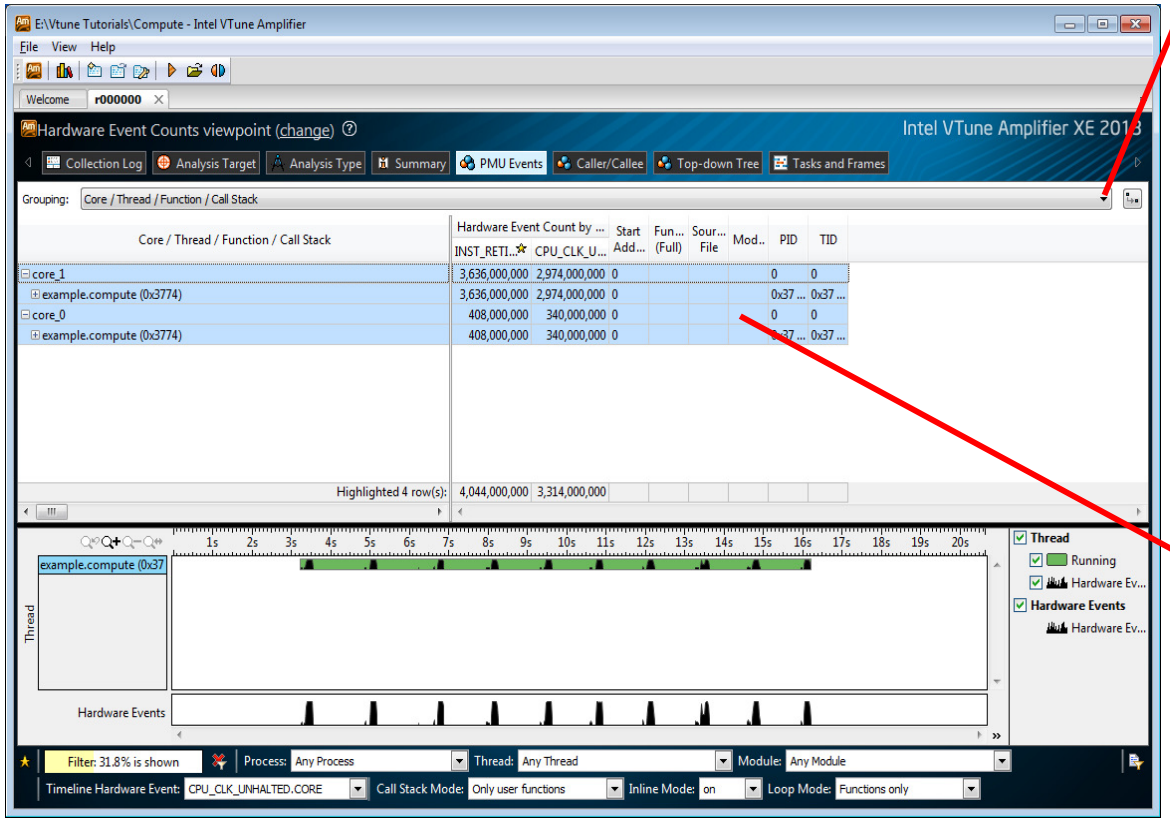


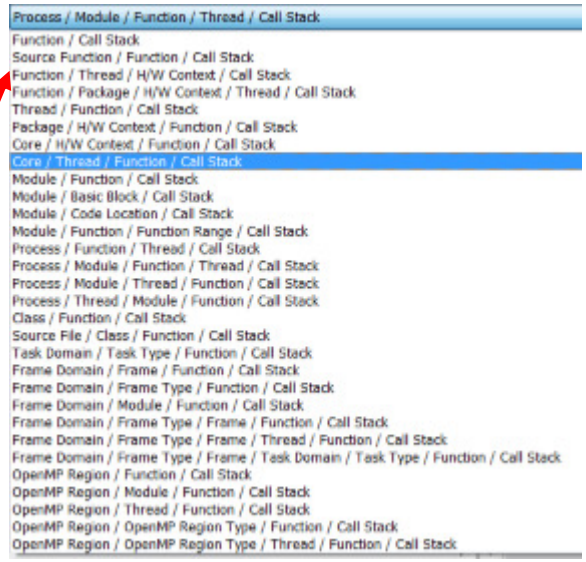
GridView

Timeline

Gives you the birds eye-view of overall execution

Data Grouping

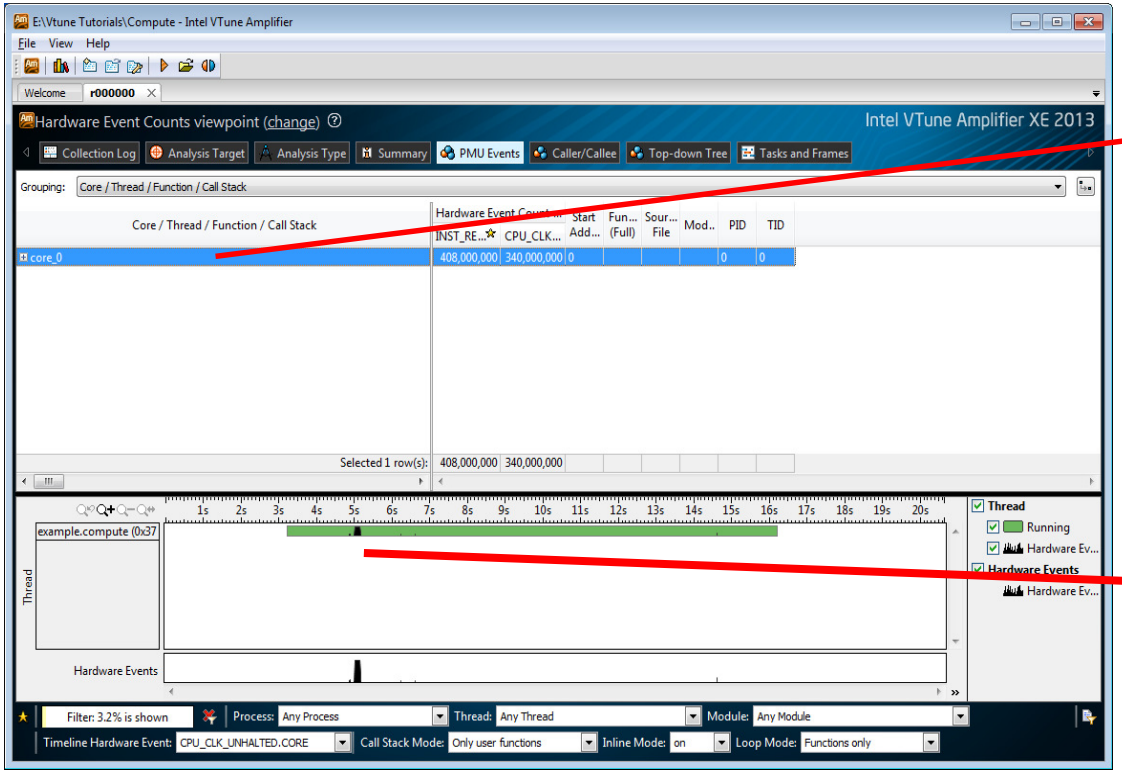




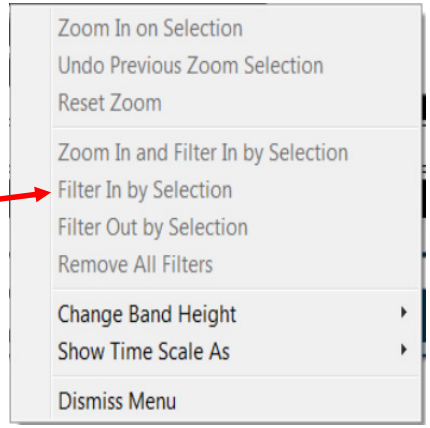
Mostly using core 1

Allows you to look at data in multiple ways

Filtering



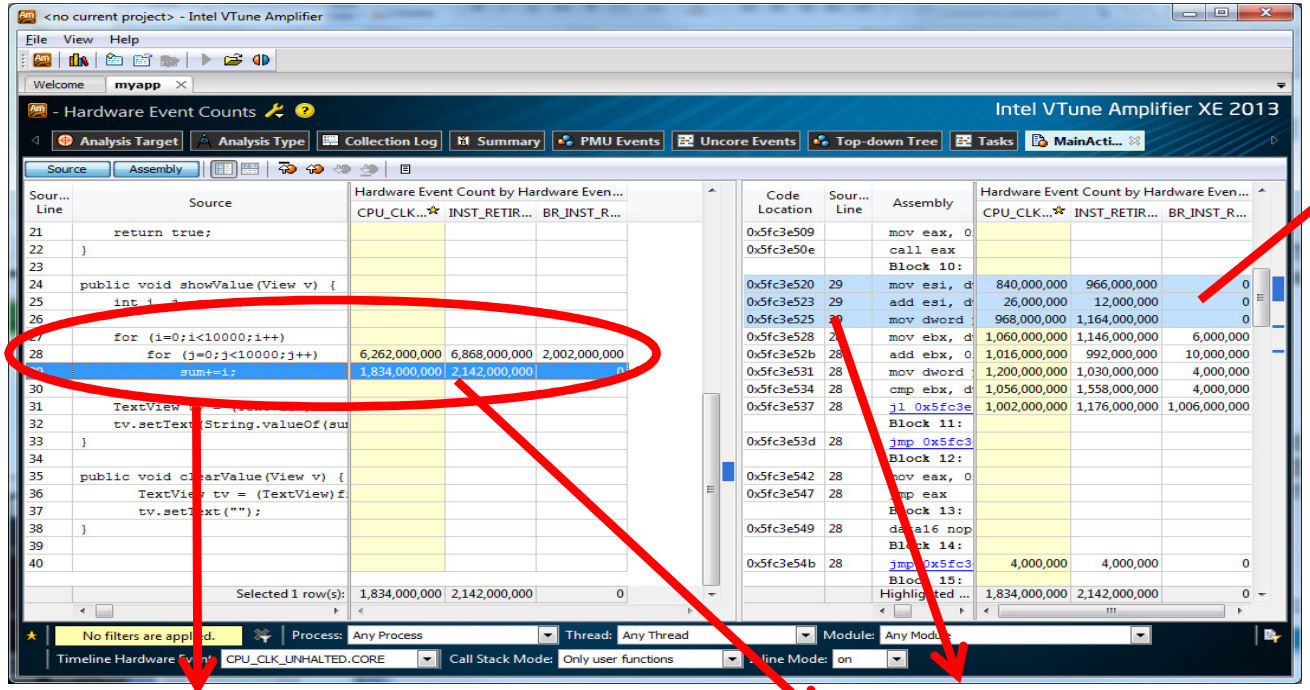
Mouse right button click



Second press used Core 0

Lets you focus on specific events in the execution

Java Source to Assembly Mapping



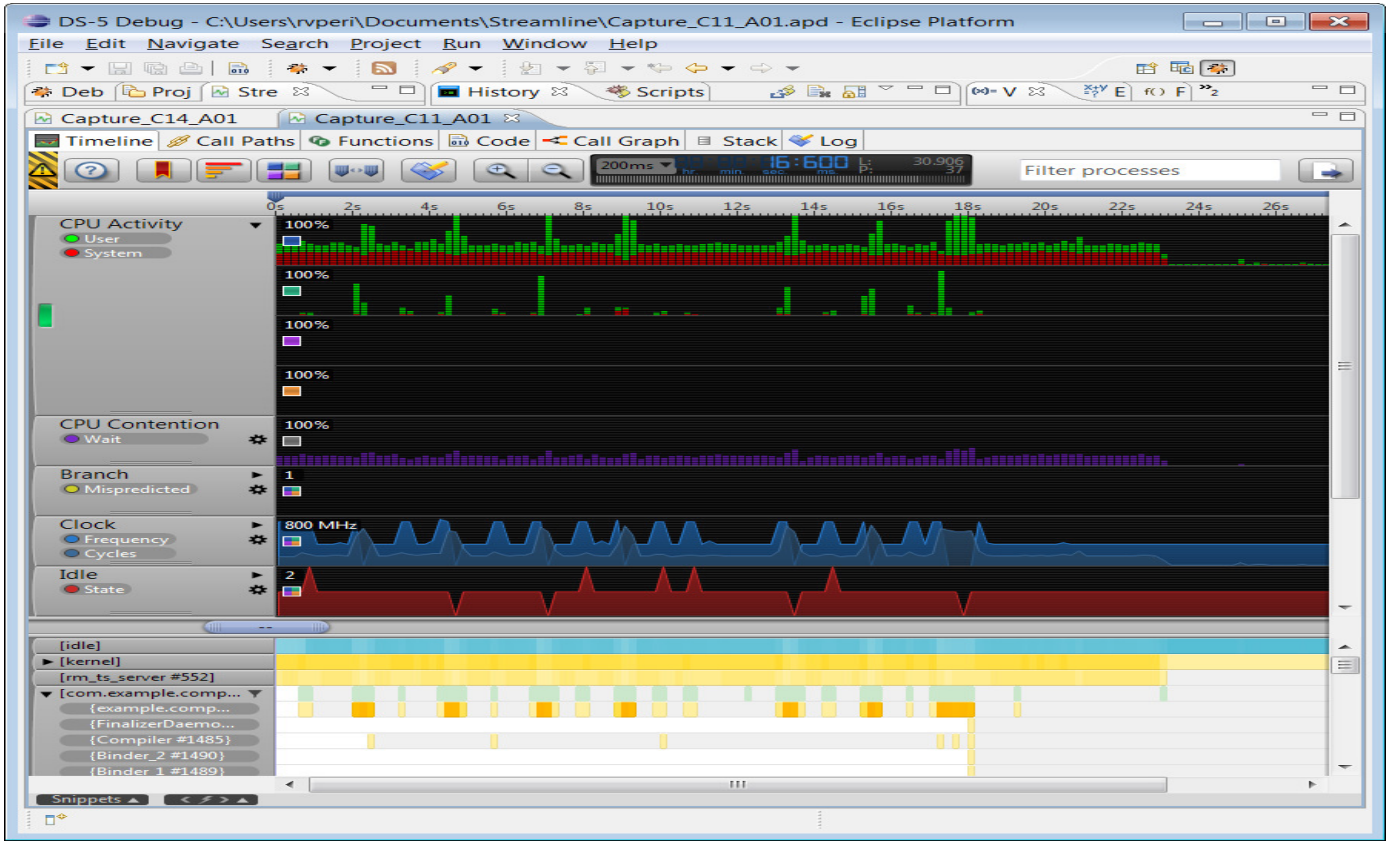
The hot loop in assembly

The hot loop

Jit asm to source java correlation

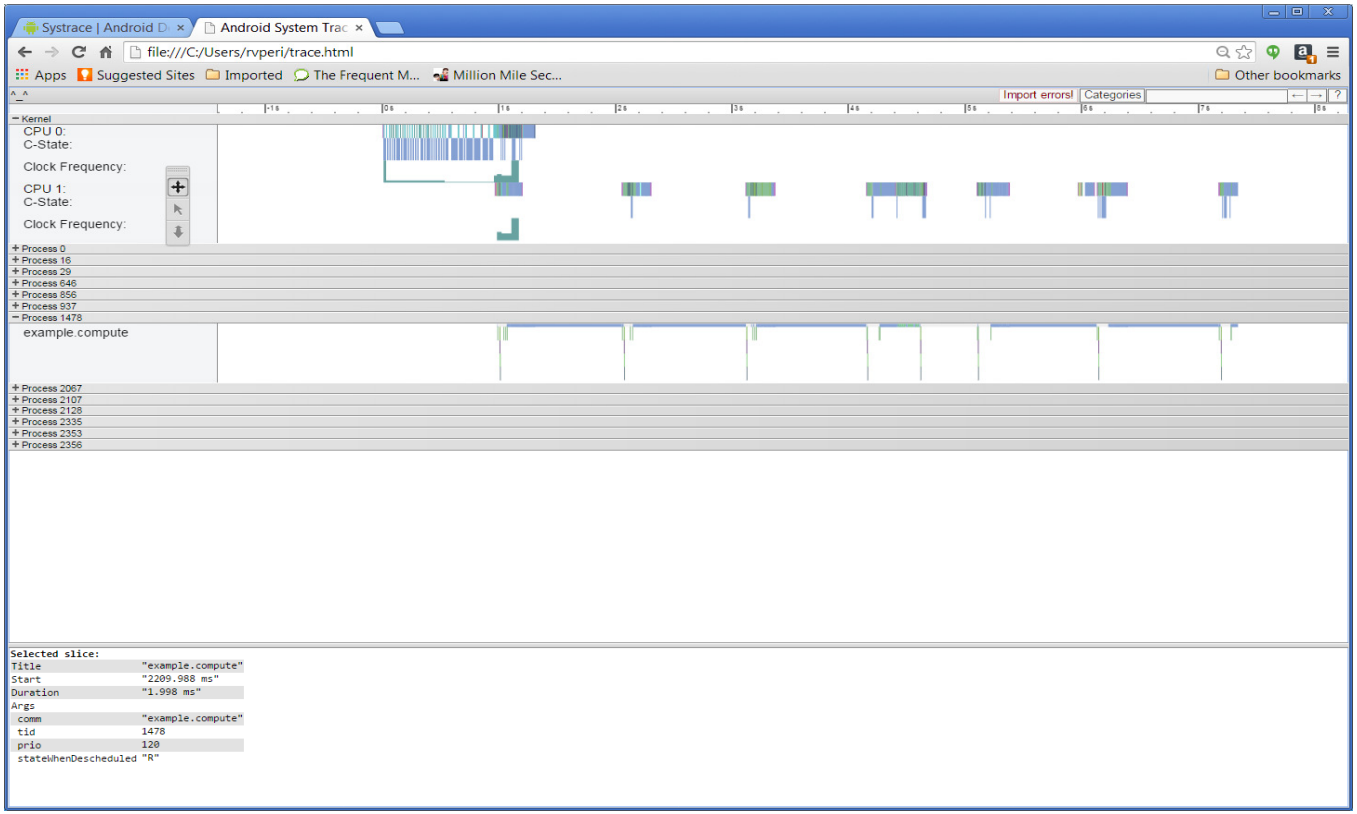
Requires support in the Dalvik/Art runtime

Arm Streamline on Nvidia Shield

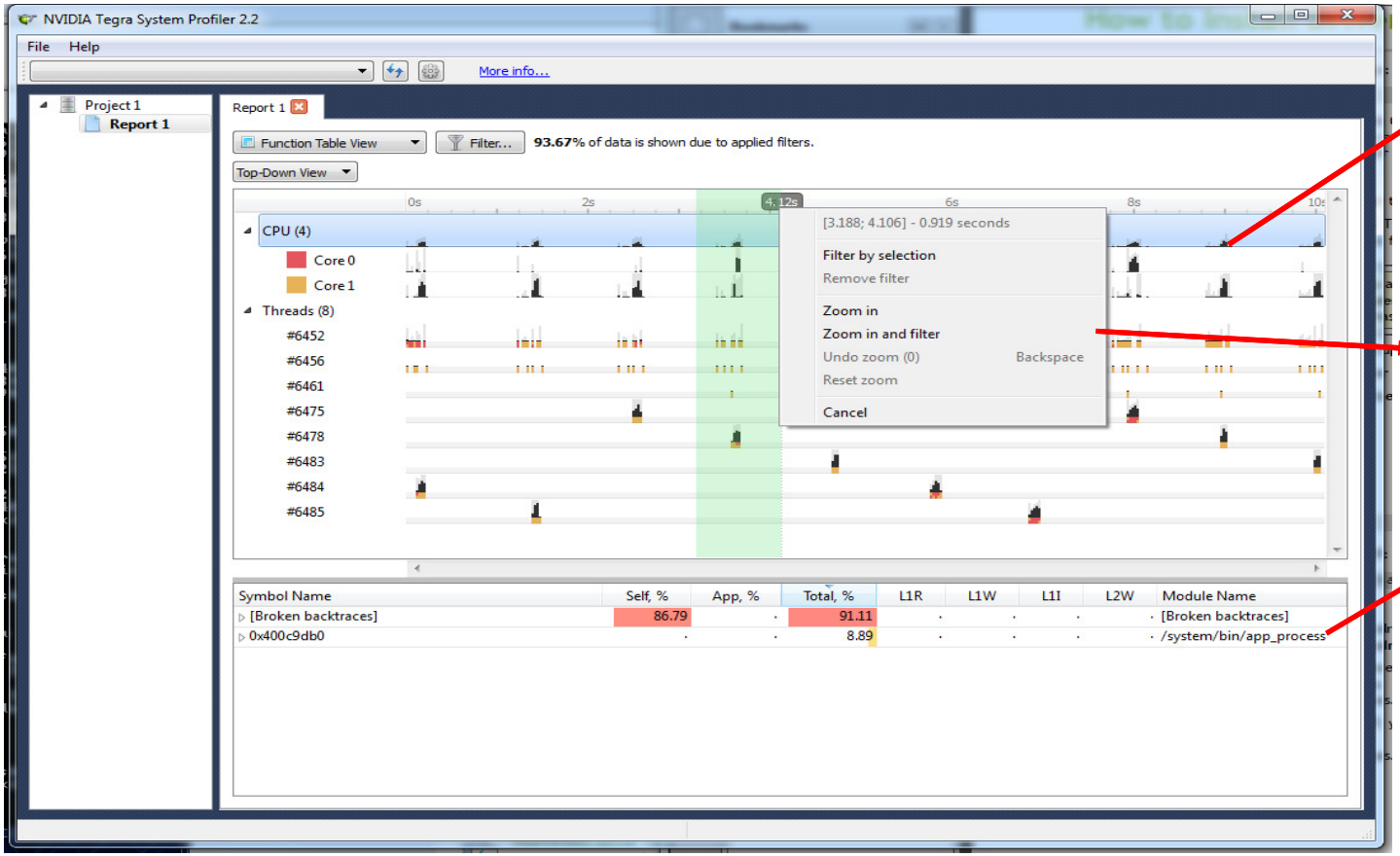


Two cores active

Systrace on Nvidia Shield for 6 key presses



Nvidia System Profiler on Nvidia Shield



TimeLine

Zoom and Filter

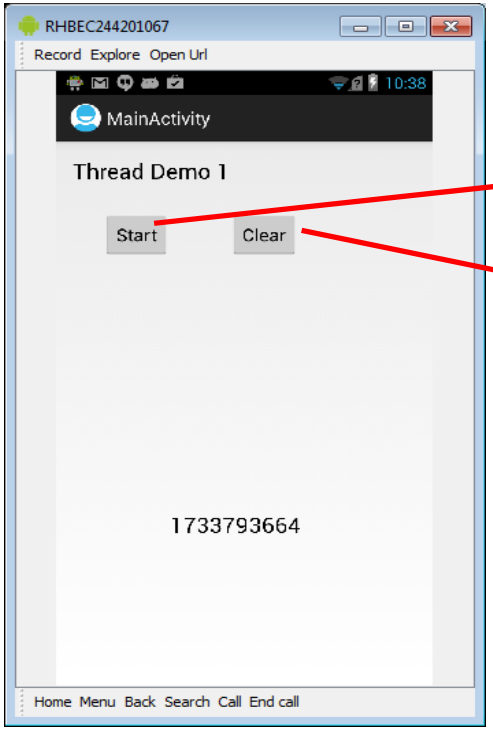
Grid view with callstacks

Observation

- All key presses are handled by one thread
- The thread is mapped to different cores at different times
- Core 1 is handling most of the work in the application for venue8
- Core 0 and core 1 are active on Nvidia shield
  - Core 1 is doing the compute and core 0 seems to be active with the OS and another process

Simple Threading

Simple Threading

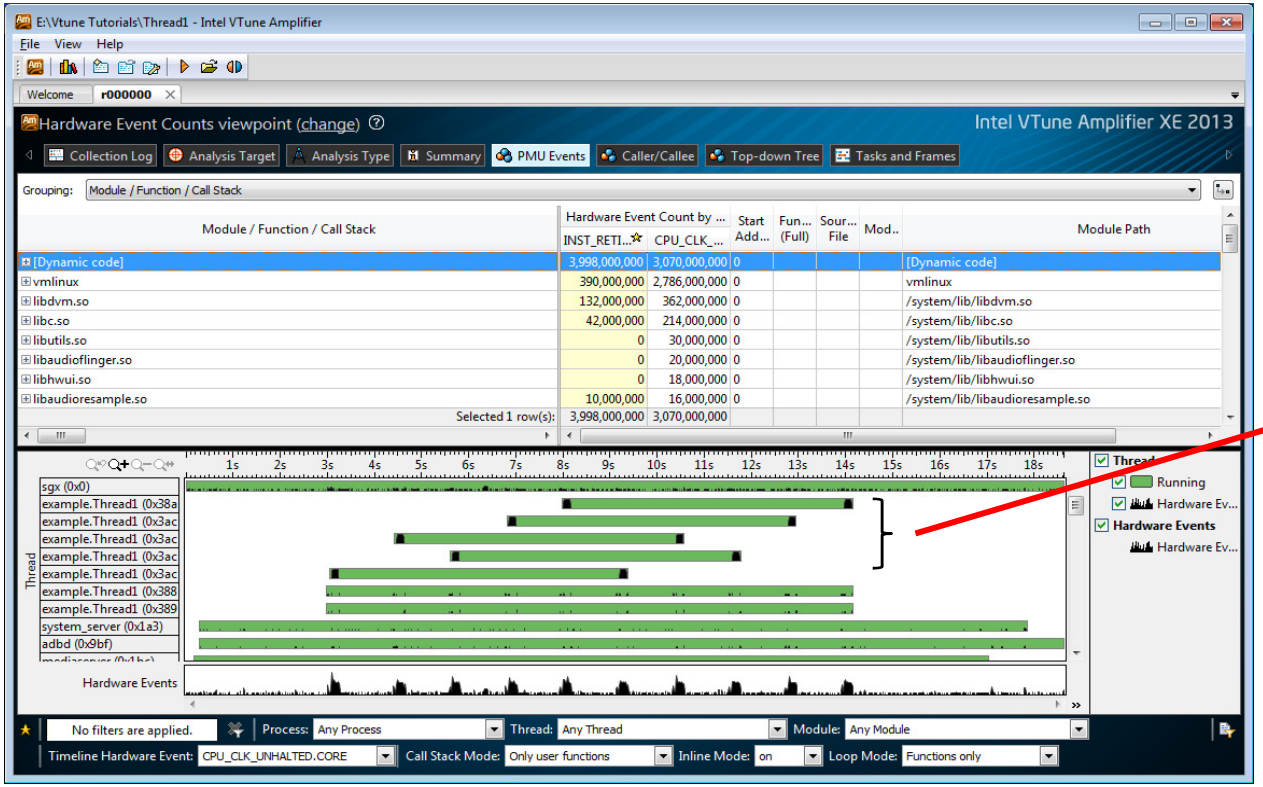



```
public void showValue(View v) {
        new LongOperation().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
}

public void clearValue(View v) {
        TextView tv = (TextView)findViewById(R.id.textView2);
        tv.setText("CLEARED");
}
```

Thread Code

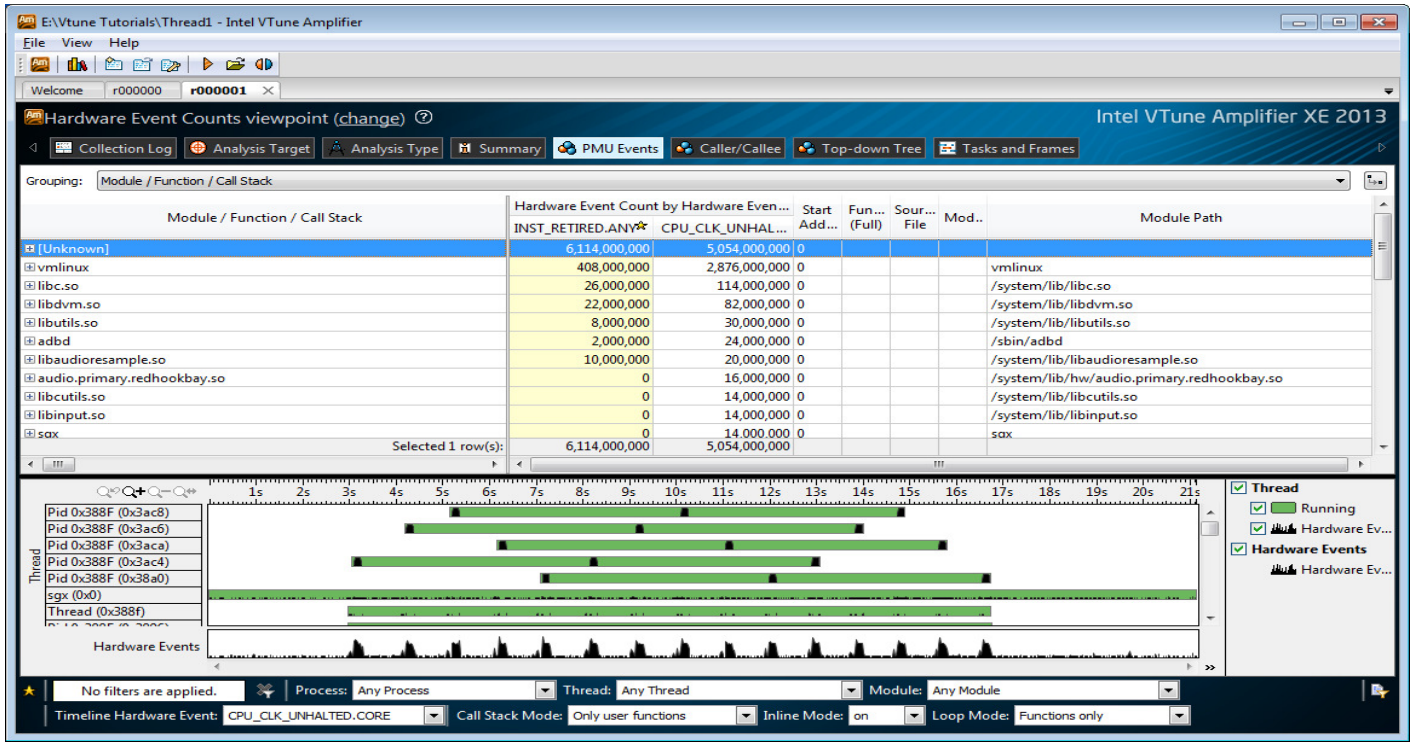
```
private class LongOperation extends AsyncTask<String, Void, String> {
      @Override
      protected String doInBackground(String... params) {
         int sum=0;
         for (int i=0;i<10000;i++)
            for (int j=0;j<10000;j++)
               sum+=i;
         return (String.valueOf(sum));
      }

      @Override
      protected void onPostExecute(String result) {
         TextView txt = (TextView) findViewById(R.id.textView2);
         txt.setText(result);
      }
}
```

The Threading Picture for 10 key presses



5 threads created and each one handling 2 key presses

The Threading Picture for 15 key presses



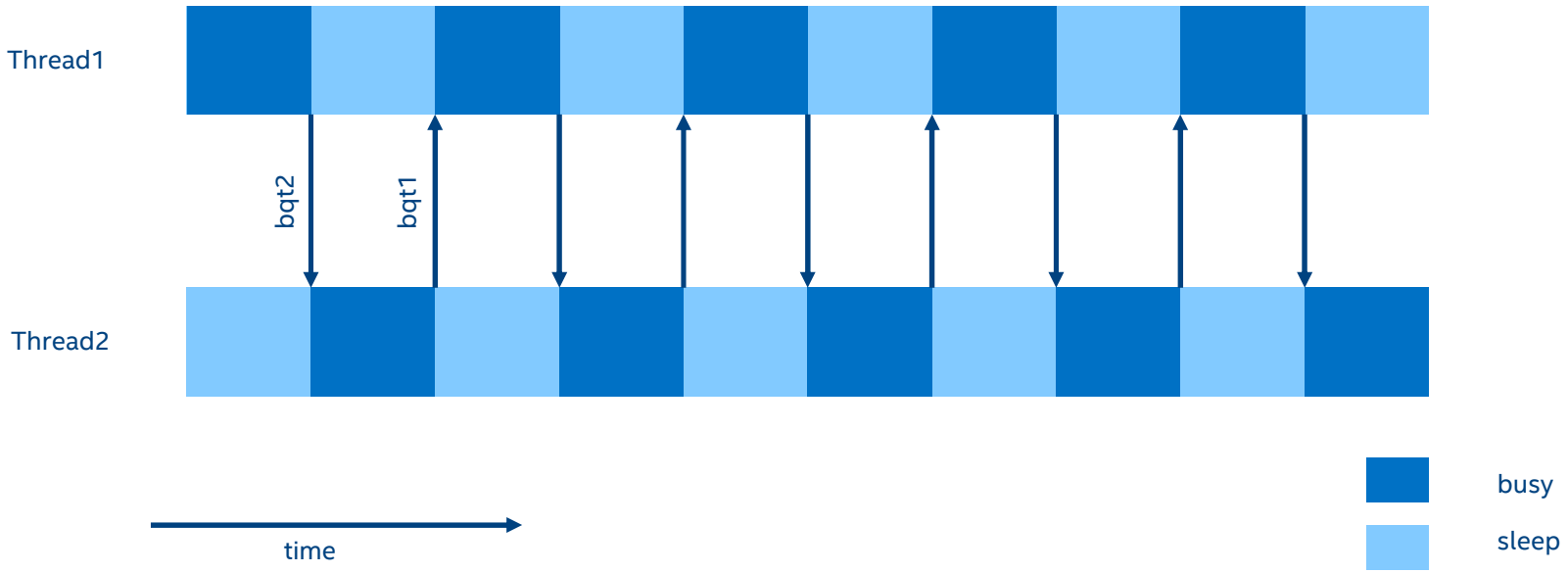
ARM DS-5 on Nvidia Shield - for 10 key presses

Observation

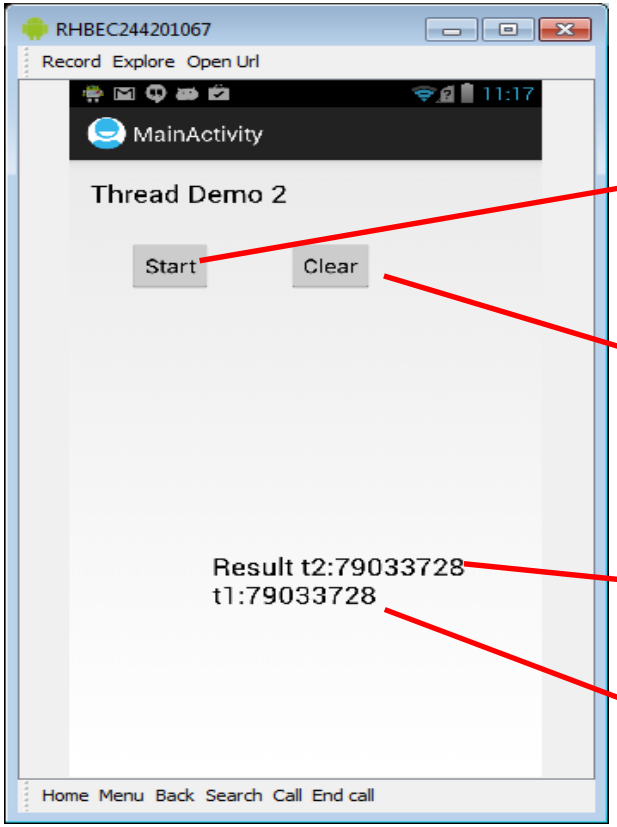
- There are 5 worker threads
- Each one gets a piece of work in a round robin fashion
- Did the thread really terminate right after the last key press ?
  - No – there were no samples from the thread
- Shield has two cores active while venue has only one core active
  - Due to the OS scheduler

Communicating Threads

Communication Pattern



Communicating Threads




```
public void showValue(View v) {
   bqt1 = new LinkedBlockingQueue<String>(2);
   bqt2 = new LinkedBlockingQueue<String>(2);

   new Thread1().executeOnExecutor(AsyncTask. THREAD_POOL_EXECUTOR);
   new Thread2().executeOnExecutor(AsyncTask. THREAD_POOL_EXECUTOR);
}

public void clearValue(View v) {
    TextView tv = (TextView)findViewById(R.id.textView2);
     tv.setText("");
}
```

Output from thread2

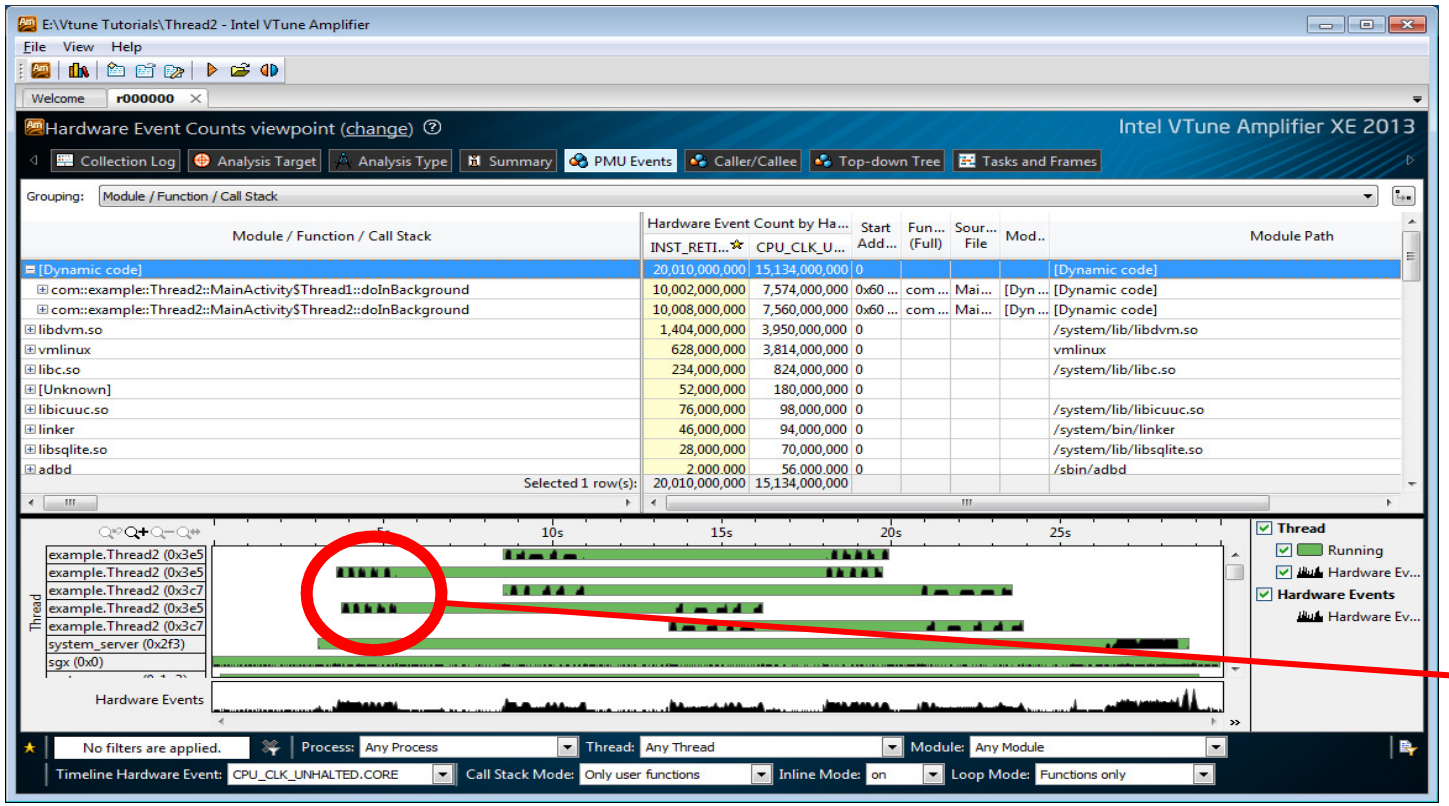
Output from thread1

Thread Code

```
private class Thread1 extends AsyncTask<String, Void, String> {
      @Override
      protected String doInBackground(String... params) {
           int sum=0;
           for (int times=0;times<5;times++)
           {
                  for (int i=0;i<10000;i++)
                     for (int j=0;j<10000;j++)
                         sum+=i;
                  try
                  {  bqt2.put("1"); }
                  catch (InterruptedException intEx)
                  { System.out.println("Interrupted! "); }
                  try
                 {   bqt1.take(); }
                 catch (InterruptedException intEx)
                 { System.out.println("Interrupted!"); }
           }
           return (String.valueOf(sum));
       }

      @Override
      protected void onPostExecute(String result) {
         TextView txt = (TextView) findViewById(R.id.textView2);
         txt.setText(txt.getText() + " t1:" + result);
      }
```

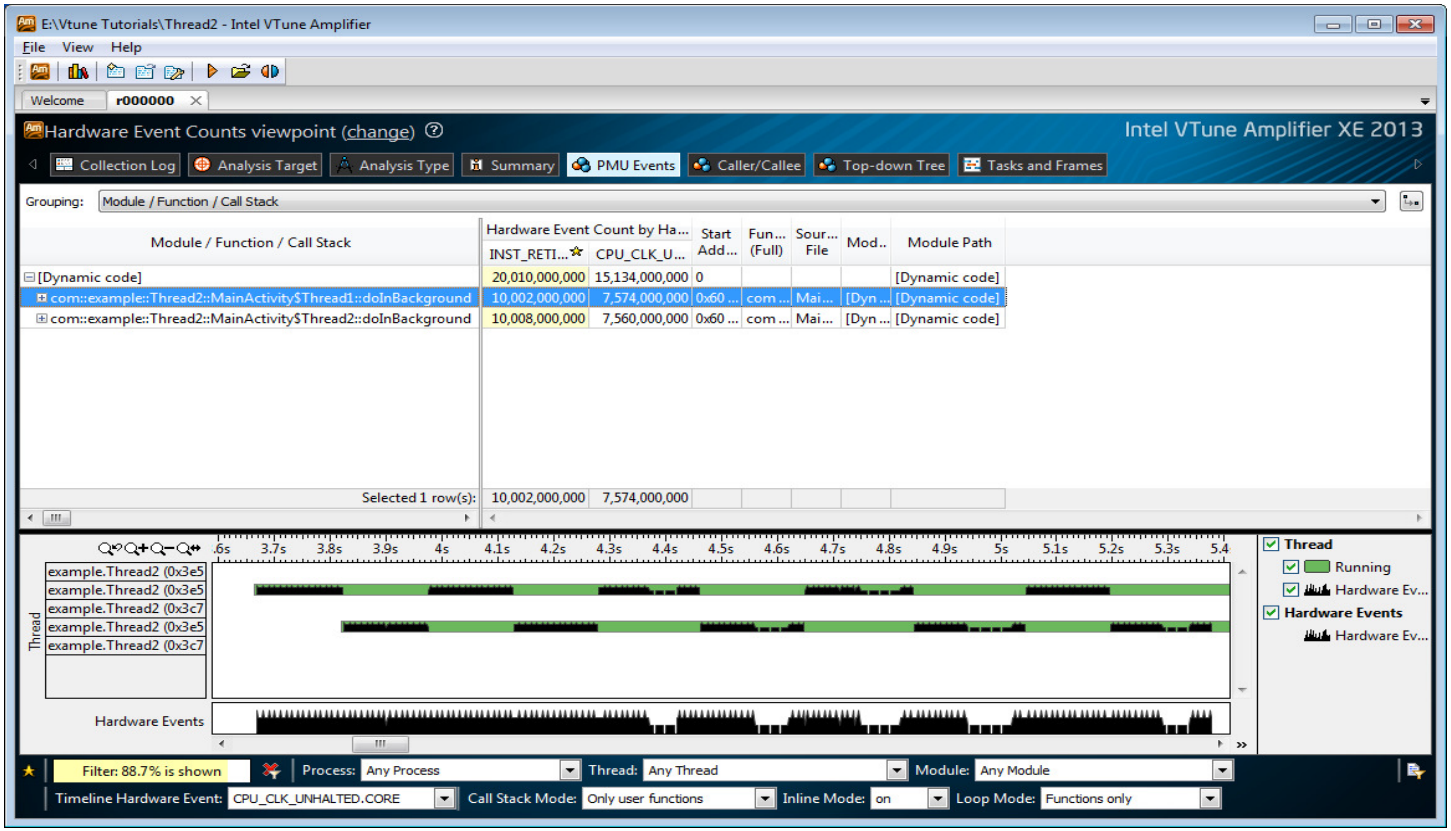
```
private class Thread2 extends AsyncTask<String, Void, String> {
      @Override
      protected String doInBackground(String... params) {
         int sum=0;
         for (int times=0;times<5;times++)
         {
             try
             { bqt2.take(); }
             catch (InterruptedException intEx)
             { System.out.println("Interrupted! "); }
             for (int i=0;i<10000;i++)
                for (int j=0;j<10000;j++)
                    sum+=i;
              try
              { bqt1.put("1"); }
              catch (InterruptedException intEx)
              { System.out.println("Interrupted! "); }
         }
         return (String.valueOf(sum));
      }

      @Override
      protected void onPostExecute(String result) {
         TextView txt = (TextView) findViewById(R.id.textView2);
         txt.setText(txt.getText() + " t2:" + result);
      }
```

The Threading Picture for 5 key presses



Alternating Thread1 & Thread2

Zoomed in view

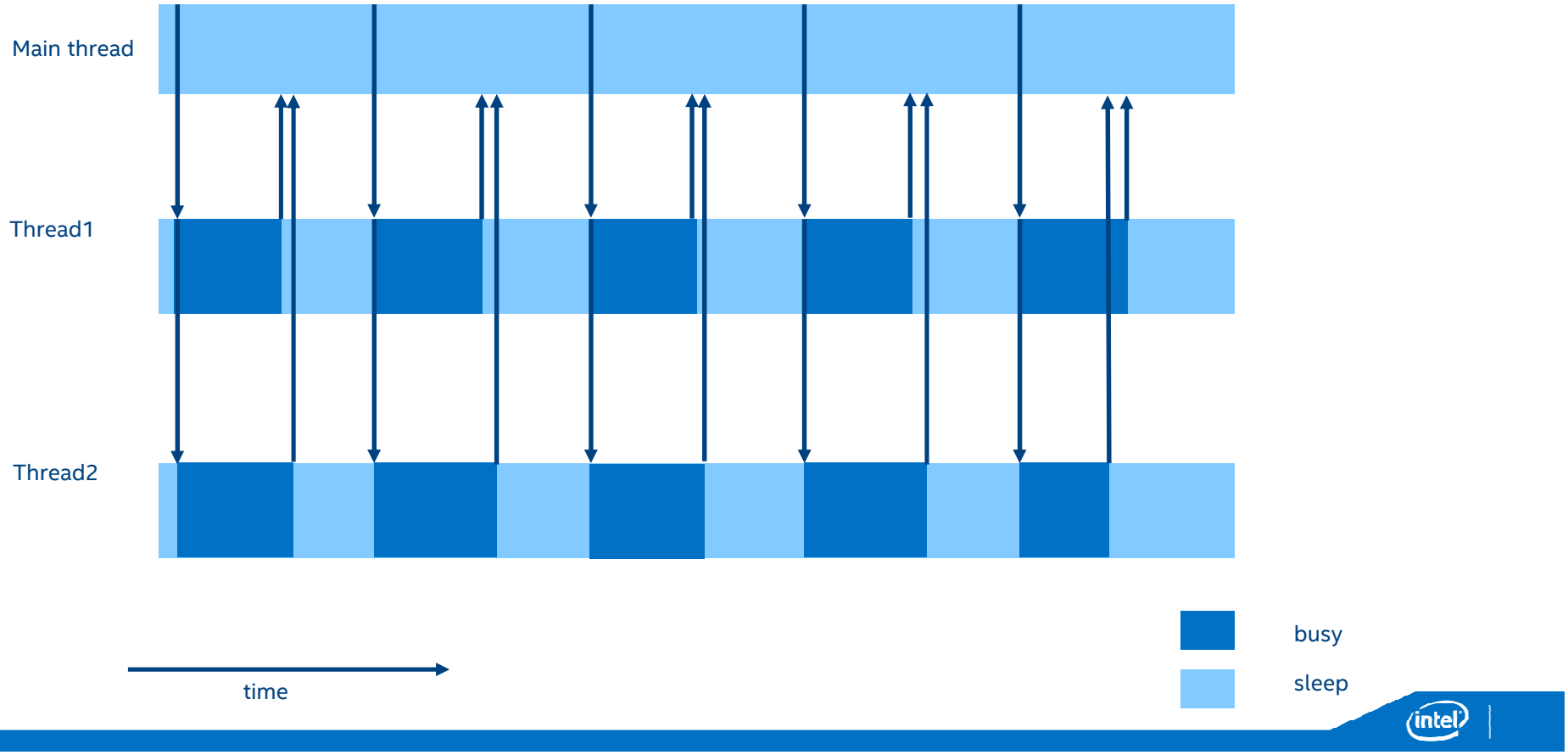


Observation

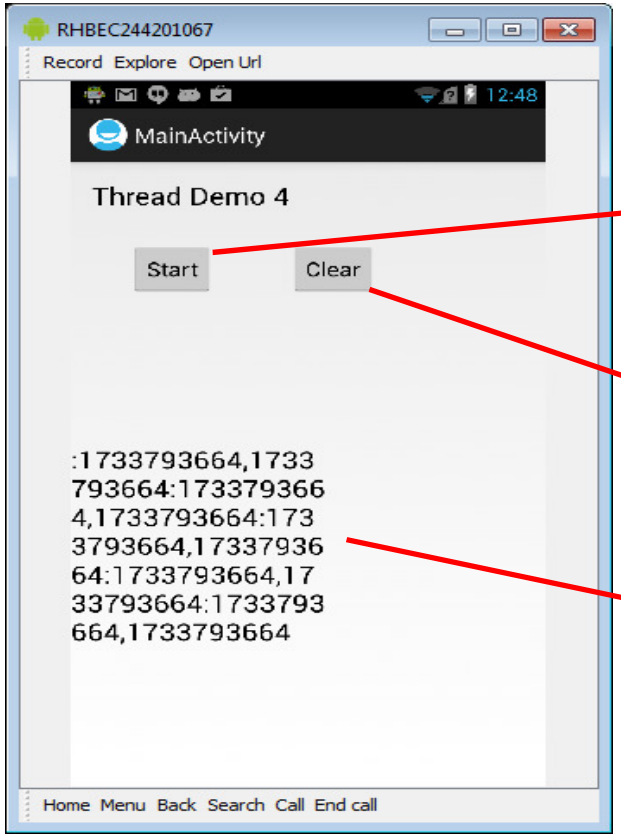
- There are 5 worker threads
- Two worker threads are selected from the pool to do the work for each key press

Simultaneously Executing Threads

Communication Pattern



Communicating Threads



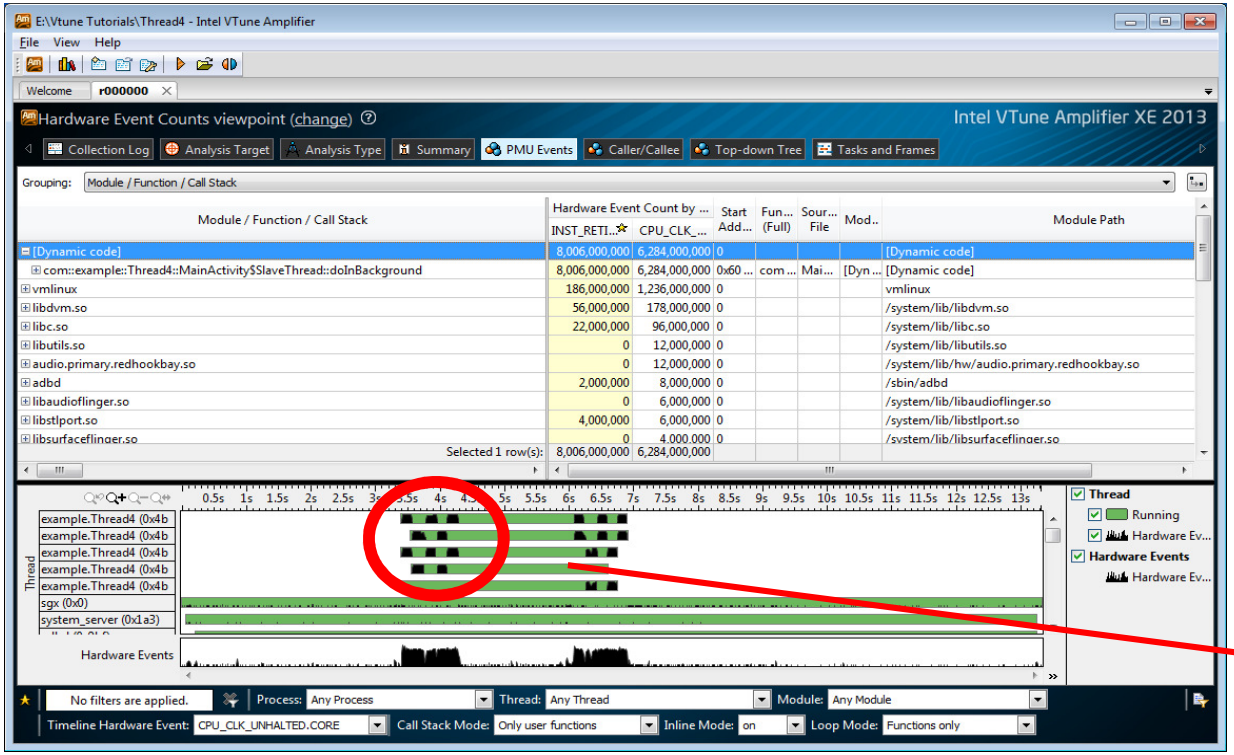

```
public void showValue(View v) {
      new MasterThread().execute();
}

public void clearValue(View v) {
   TextView tv = (TextView)findViewById(R.id.textView2);
    tv.setText("");
}
```

5tuples - <Output from thread1, output from thread2>

Thread Code

```
private class MasterThread extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... params) {
        String result="";
        for (int i=0;i<5;i++)
        {
            AsyncTask<String,Void,String> t1 = new SlaveThread().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
            AsyncTask<String,Void,String> t2 = new SlaveThread().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
            String res1 = null;
            try { res1 = t1.get(); }
            catch (InterruptedException e) { e.printStackTrace(); }
            catch (ExecutionException e) { e.printStackTrace(); }
            String res2 = null;
             try { res2 = t2.get(); }
            catch (InterruptedException e) { e.printStackTrace(); }
            catch (ExecutionException e) { e.printStackTrace(); }
            result = result + ":" + res1 + "," + res2;
        }
        return (result);
    }

    @Override
    protected void onPostExecute(String result) {
       TextView txt = (TextView) findViewById(R.id.textView2);
       txt.setText(result);
    }
 }
```

The Threading Picture for two key presses



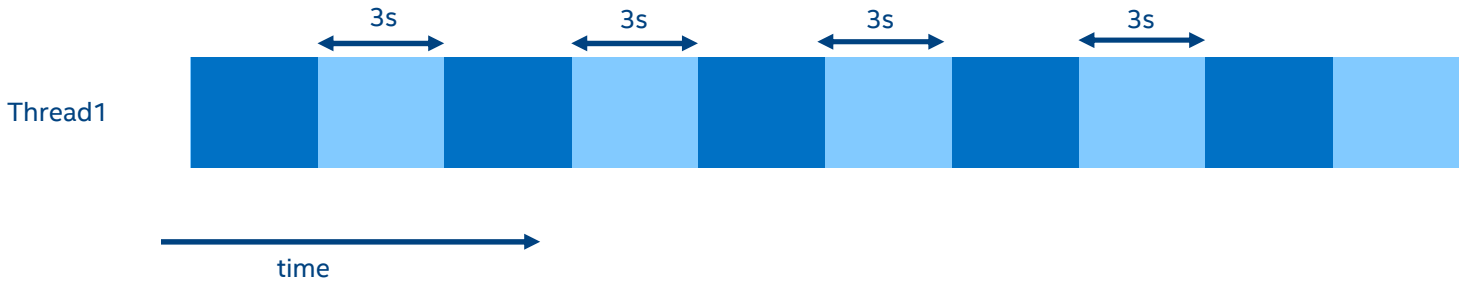
Thread1 &
Thread2 executing
At same time

Observation

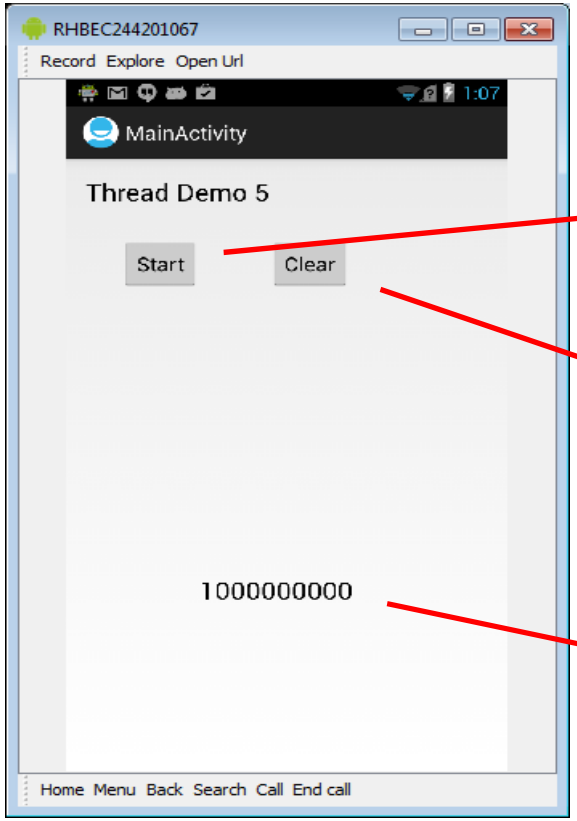
- When threads execute at same time then cores are being effectively utilized
  - Better performance
  - We want this to happen in a multi-core system
- Free running threads without any synchronization should preferably be scheduled on different cores
  - Some times OS scheduler may not do so

Lazy Threads

Communication Pattern



Lazy Thread

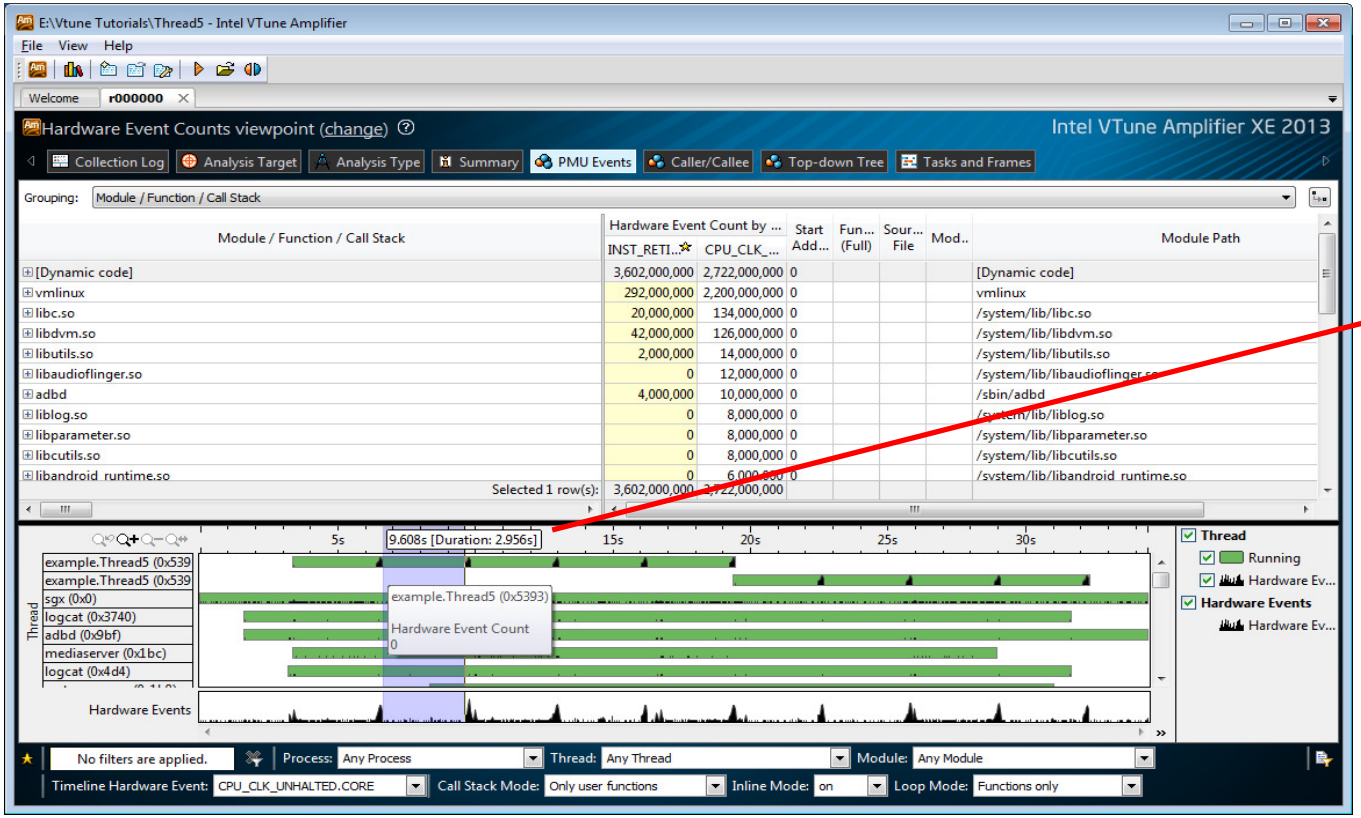



```
public void showValue(View v) {
      new Thread().execute("");
}

public void clearValue(View v) {
   TextView tv = (TextView)findViewById(R.id.textView2);
    tv.setText("");
}
```

Thread Code

```
private class Thread extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... params) {
    int sum=0;
      for (int i=0;i<5;i++)
      {
        try {
          synchronized (this) {
            wait(3000); // wait for 3sec
          }
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      for (int j=0;j<10000;j++)
        for (int k=0;k<10000;k++)
          sum+=i;
      }
      return (String.valueOf(sum));
    }

    @Override
    protected void onPostExecute(String result) {
      TextView txt = (TextView) findViewById(R.id.textView2);
      txt.setText(result);
    }
  }
```

Performance view



Thread slept
For 3 sec

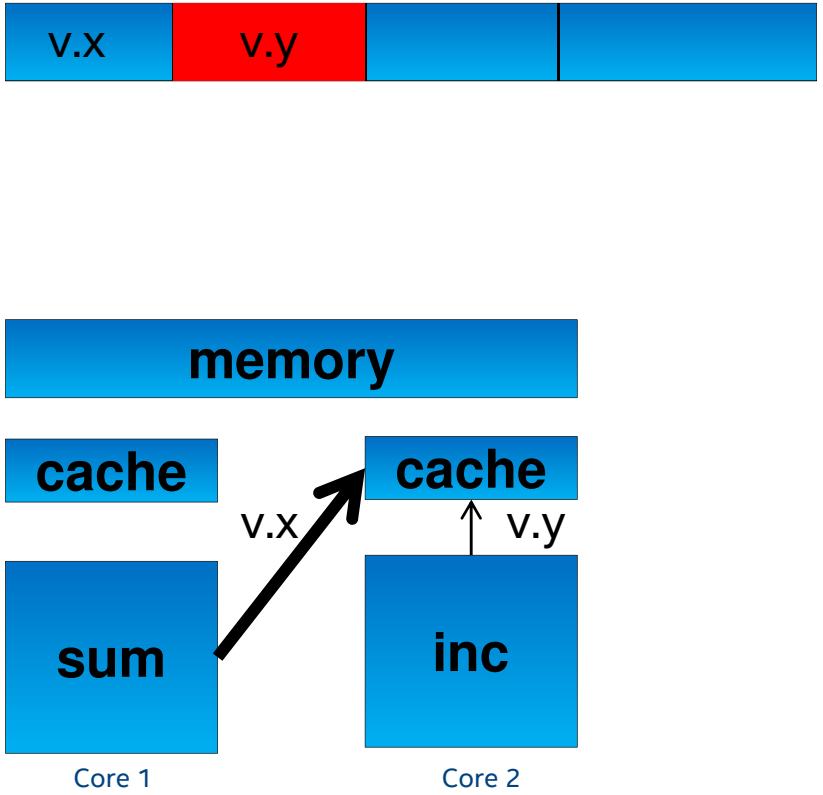
Observation

- When threads are sleeping they are doing for the right reason
- Sleeping threads are good for power/energy consumption
- Frequent sleep/wakeups are bad for power/energy consumption

False Sharing

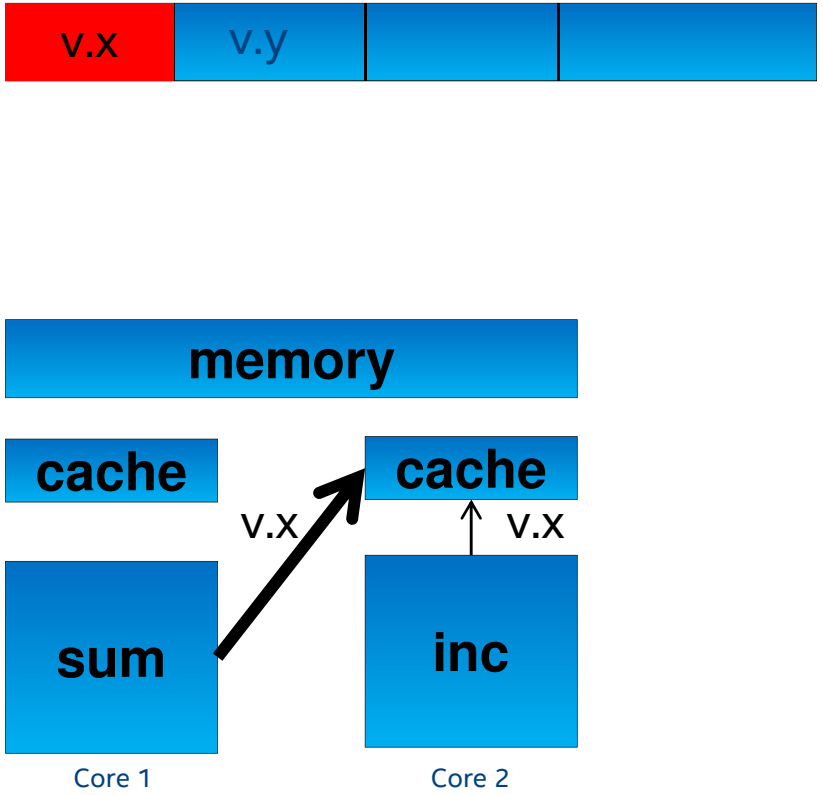
What is False Sharing ?

```
struct  {
   int x;
   int y;
} v;

/* sum & inc run in parallel */

int sum(void)
{
   int i, s = 0;
   int i;
   for (i = 0; i < 1000000; ++i)
      s+=v.x;
   return s;
}

void inc(void)
{
   int i;
   for (i = 0; i < 10000000; ++i)
      v.y++;
}
```

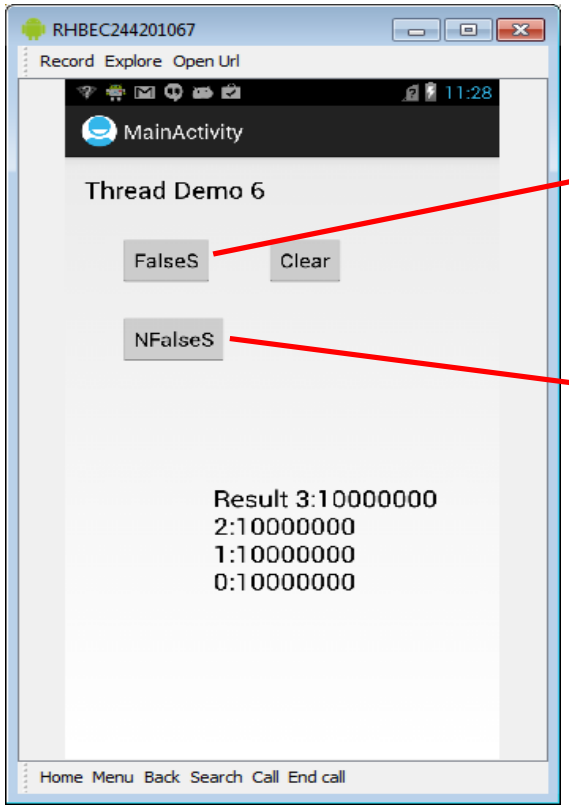


Is there “True” Sharing ?

```
struct  {
   int x;
   int y;
} v;

/* sum & inc run in parallel */

int sum(void)
{
   int i, s = 0;
   int i;
   for (i = 0; i < 1000000; ++i)
      s+=v.x;
   return s;
}

void inc(void)
{
   int i;
   for (i = 0; i < 10000000; ++i)
      v.x++;
}
```



False Sharing App




```
public void showValue1(View v) {
  for (int i=0;i<256;i++)
   a[i]=0;
  for (int i=0;i<4;i++)
      new Thread().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, String.valueOf(i));
 }

 // No False Sharing
 public void showValue2(View v) {
   for (int i=0;i<256;i++)
   a[i]=0;
   for (int i=0;i<4;i++)
      new Thread().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, String.valueOf(i*64));
 }
```

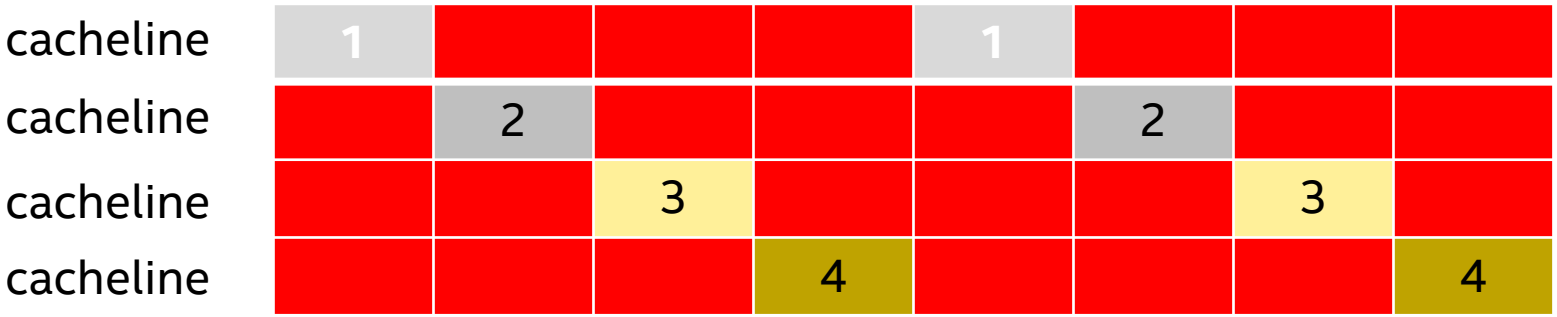
Thread Body

```
private class Thread extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... params) {
      int tid=Integer.parseInt(params[0]);
      int lim = tid+32;
       for (int j=0;j<1000;j++)
          for (int k=0;k<10000;k++)
             for (int i=tid;i<lim;i+=4)
                a[i]=a[i]+1;
       return (params[0]);
    }

    @Override
    protected void onPostExecute(String result) {
       TextView txt = (TextView) findViewById(R.id.textView2);
       txt.setText(txt.getText() + " " + result + ":" + a[Integer.parseInt(result)]);
    }
 }
```

Memory Access Pattern of Threads

False Sharing

No False Sharing



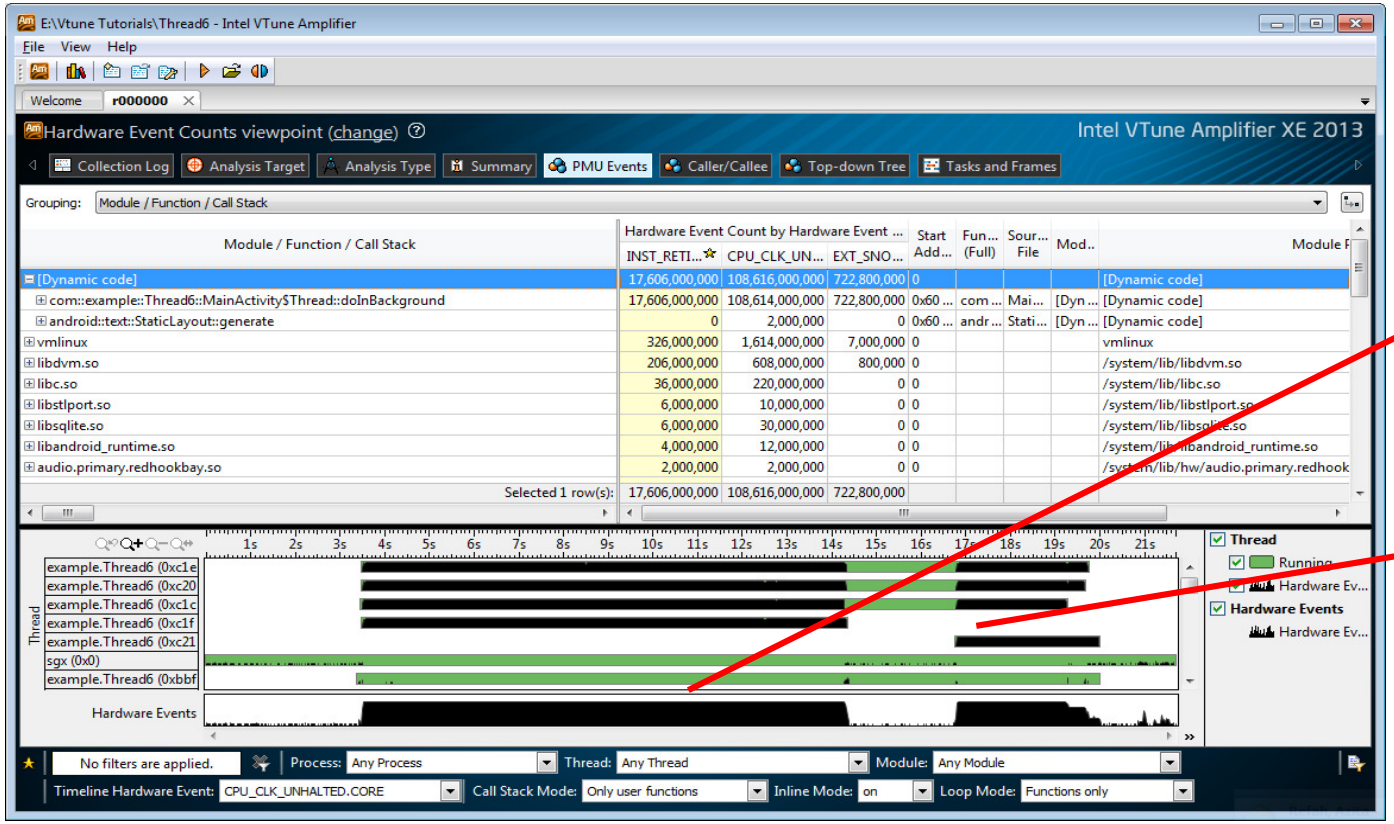
In both cases same amount of work is done

Sample App

Click falseS

Click NfalseS

Profile of the run

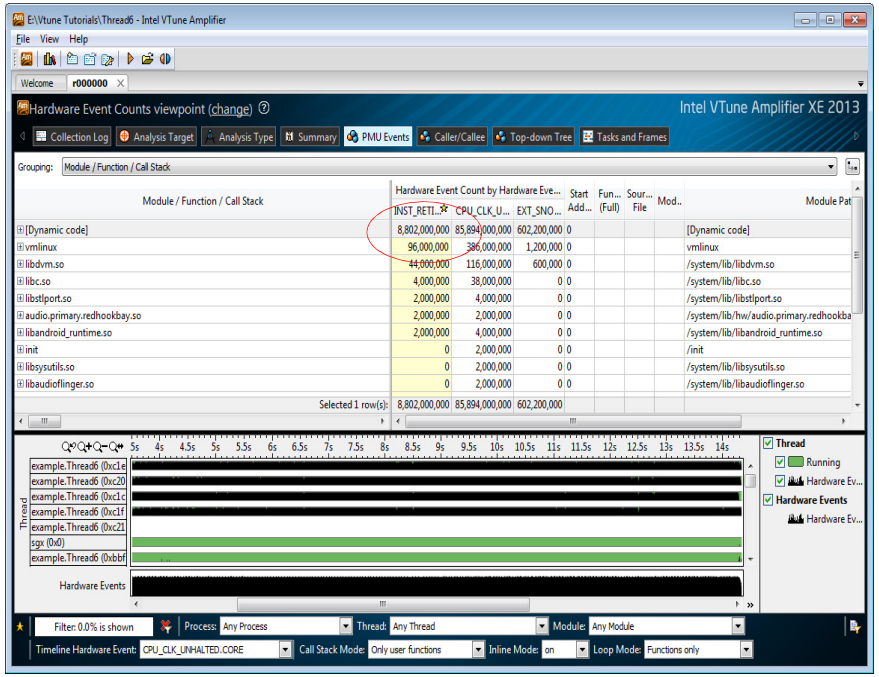


False Sharing

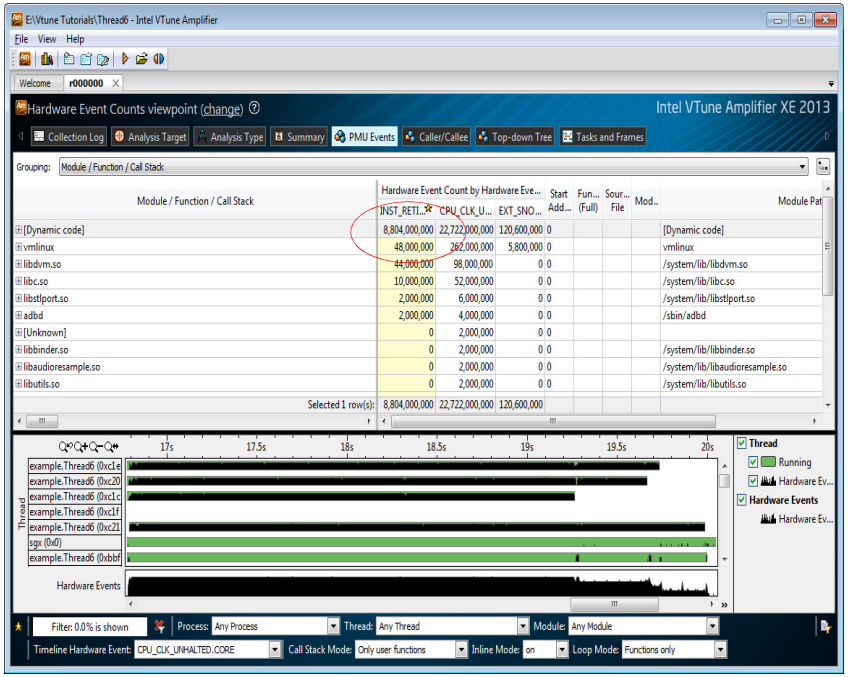
No False sharing

Detailed view

False Sharing



No False Sharing



Same number of instructions executed

Observation

- Know the architecture of your platform
- Pay attention to the memory access patterns inside your threads
  - Make sure caches are effectively utilized
  - Data structures can be rearranged to avoid false sharing

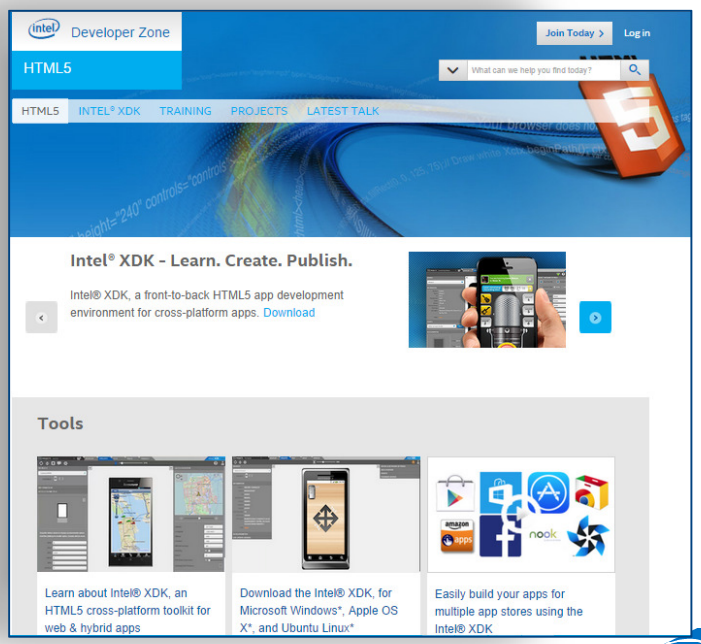
Conclusion

- A good performance tool can show what exactly is going on in your platform
- Sample programs give you a good idea about your platform
- Different platforms behave differently
  - Make sure you run your examples and observe the differences between them
- Android has complicated threading model
  - Make sure you understand it properly

Intel® Developer Zone

Tools. Knowledge. Community.

- Free tools and code samples
- Technical articles, forums and tutorials
- Connect with Intel and industry experts
- Get development support
- Build relationships



software.intel.com

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Please take a moment to fill out the class feedback form via the app. Paper feedback forms are also available in the back of the room.

eventmobi.com/adcboston