My name is Krishnendu Sadhukhan, I work in the Solaris x86 Kernel group, and I'm going to talk about hardware performance counters that the AMD Shanghai/Barcelona platforms bring about to us and what we have done to enable those counters on the Solaris front.

So basically the AMD Shanghai or Barcelona platform brings about four 48-bit performance counters and our Solaris software actually enables different events using those performance counter registers, and we also have four different event select registers that can control those performance counters. So we have four performance counters and we have four event select registers that control those performance counters. And we have a whole list of events that we can use to get system data. For example, we can use those counters to know how many times we access memory, how many times we access L1 cache or L2 caches, or how many times L1 caches or L2 caches are flushed, and things like that. Basically there is a PCB back end which is used by the PCB libcpc library, and we have a couple commands in user land that access that library in order to get through the PCB to the hardware level. We have two commands, cpustat and cputrack that users can use in order to get performance data out of the system on Shanghai and Barcelona platforms. I can show you how to use those commands and we can look at some of the example outputs.

Here I'm showing the cpustat command. Use the cpustat –h command to get a list of events that the Shanghai or Barcelona platform supports, so here we see the list of events that the user can use to get system data. Or alternatively the user can also use the cputrack command to get the same list of events that they can use to get system data.

Now I'm going to show you how we can use cpustat to know how many times the instruction cache has been evicted. I'm using the performance counter number 2 here, as you can see "pic2," and I'm running the command now. So now we see here, we're using pic2 here, performance counter number 2, and we're getting the data, for example, cpu4, cpu5, this many times the cache has been evicted. And it continues to do it because I didn't specify any number, so it continues to get system data.

Now I'm going to show you how you can use cpustat to get system data for a particular application process. Here on the left window I'm running a particular process, a.out, which is memory intensive, which accesses memory every time in a loop. And on the right, I am binding that particular process – I got the PID of the process through pgrep – and there I have bound that particular process to processor number 4 here so that I can know which processor is getting more memory accesses, doing more memory actions. So now, here I have bound the process to a particular processor, and now I'm going to run the cpustat command to know how many times we're accessing memory for that particular processor. Now as you can see we got the first output here. If you look at processor number 4 here, we get a very
huge number, which is 387,000 compared to others. In each output we see that for processor number 4, we get a very huge value. This is because this application is running here and we have bound that process to that particular processor. Now I’m going to show you, if I kill that process here in the left window so that process is now out. Now if we look at the next output we see that for processor number 4, the output value is much less. If you look at 4 here, it’s only 64 compared to others where we had 36,000 or more. So this is how we can use cpustat to get the performance counter values from the AMD Shanghai and Barcelona processors.

Now I’m going to show you how to use the other user land command, which is cputrack, to track a particular application. The one that I showed you before was for any application on the system just to get the general data. Now this particular command, which is cputrack, can be used for a particular application. For example, here on the left I’m running the same old application a.out, which is memory intensive. And on the right, I’m using the cputrack command here to track that particular application. The PID is that one for a.out which I got through pgrep. So now I am executing cputrack to know how many times that particular application is accessing memory. Let me run that command. Here I am using the performance counter number 0 by default. Now here we see the different numbers that we’re getting. Those numbers indicate how many times that particular a.out is accessing memory. This gives us an idea of how much the system is busy and how much the application is using the system.

The Shanghai and Barcelona platforms bring us a whole new set of performance counters which were not available before and by virtue of OpenSolaris you can now use those hardware counters to track your systems in a much better way than you could in the past.