

Dynamic logical processors for Hyper-Threading on HP-UX 11i v3



Introduction	3
Introduction to Intel Itanium 2 Hyper-Threading Technology.....	3
Hardware threading model.....	3
Hardware switch events	3
Sharing memory hierarchy.....	4
Logical processors	4
LCPU and Hyper-Threading.....	5
LCPU and processor sets	5
Application considerations of Hyper-Threading Technology.....	6
Considerations for regular commercial applications	7
Your mileage may vary: Prove that HT is good for the application.....	7
HT provides opportunity for greater throughput, not better response time.....	7
Applications with high processor utilizations may benefit from the use of HT	7
The ability of applications to scale can affect HT performance benefits.....	7
Applications with tighter cache management might not benefit from HT.....	7
Applications with a CPI of less than 1.5 might not benefit from HT	7
Applications with a few stalling L3 cache misses might not benefit from HT.....	8
Application with processes that just fit in the L3 cache might not benefit from HT.....	8
Platforms with lower memory latencies yield less advantage for HT use	8
High bus utilization resulting from a combination of application and platform configuration might provide less advantage for HT use	8
TPC-C type applications might see a benefit from HT	8
An application composed of homogeneous processes might see a benefit from HT	8
A general mix of application processes might see a benefit from HT.....	8
Use normal HP-UX threading and locking mechanisms	8
HP-UX can simultaneously accommodate both HT-on and HT-off requirements.....	9
Do not base conclusions of performance based only on simple benchmark behavior	9
Considerations for high-performance technical computing.....	9
Most technical applications might see no or negative benefit.....	9
Your mileage may vary: with technical computing	9
Enabling Hyper-Threading Technology	9
Enabling at the firmware level	9
System configuration information	10

LCPU attribute setting of the default PSET	10
Changing HT setting for the next boot.....	11
PSET operations	11
pset_create(2).....	11
pset_assign(2) and pset_destroy(2).....	11
pset_setattr(2).....	12
psrset(1M)	13
Interaction with Online Addition and Deletion.....	13
Time accounting	13
Core and LCPU utilization.....	15
Variable core utilization	15
Per thread and process utilization	15
Comparing throughput and accuracy in time accounting.....	15
Time accounting extension.....	16
Topological information	16
pstat_getprocessor(2).....	16
pstat_getdynamic(2)	17
mpctl(2).....	17
Example to build bottom-up topological information.....	18
pset_ctl(2)	18
Appendix A.....	20
HP-UX 11i v2 extensions for portability.....	20
mpctl(2) extension.....	20
pset_ctl(2) extension	20
pstat_getprocessor(2) and pstat_getdynamic(2) extension.....	20
Special consideration for pstat(2) extensions	20
pstat(2) data structure extension	20
For more information.....	22

Introduction

The new generation of the Intel® Itanium® 2 processor, code named Montecito, incorporates several advanced features and improvements, including multiple cores per processor and multiple hardware threads per core. This document describes the key enhancements to the HP-UX 11i v3 operating system to support the new multiple hardware thread feature called the Hyper-Threading (HT) Technology.

Introduction to Intel Itanium 2 Hyper-Threading Technology

Montecito supports chip-level multiprocessing by incorporating two processor cores on each socket (chip). Each processor core offers two-way hardware multithreading where functional units are shared between two hardware threads (two independent streams of instructions) but the processor states are duplicated. Thus, each socket has separate instances of processor state to support the execution of four different threads (that is, each socket supports two cores and each core supports two hardware threads). Because of the duplicated architectural state, each hardware thread appears as a complete processor to the operating system.

The main motivation of multiple hardware threads per processor core is to share underutilized resources between multiple hardware threads to increase the overall throughput and performance. On Montecito, the increase in the throughput is achieved by making use of the idle cycles and idle functional units caused by memory stalls.

Hardware threading model

There are two basic approaches to hardware multithreading in the industry: simultaneous multithreading (SMT) and temporal multithreading (TMT). SMT enables the hardware thread contexts to share the processor resources simultaneously. TMT enables the hardware thread contexts to share the processor resource at some fixed time intervals. The Montecito HT Technology combines both approaches in a way that the hardware threads share memory hierarchy using the SMT approach and share the cores using the TMT approach. The Montecito TMT approach is further refined with the hardware control that dynamically relinquishes the cores to the appropriate hardware thread when a high-latency event (that is, memory stalls) might cause a hardware context switch before the expiration of hardware thread quantum. This modification is known as switch-on-event multithreading (SoEMT).

Hardware switch events

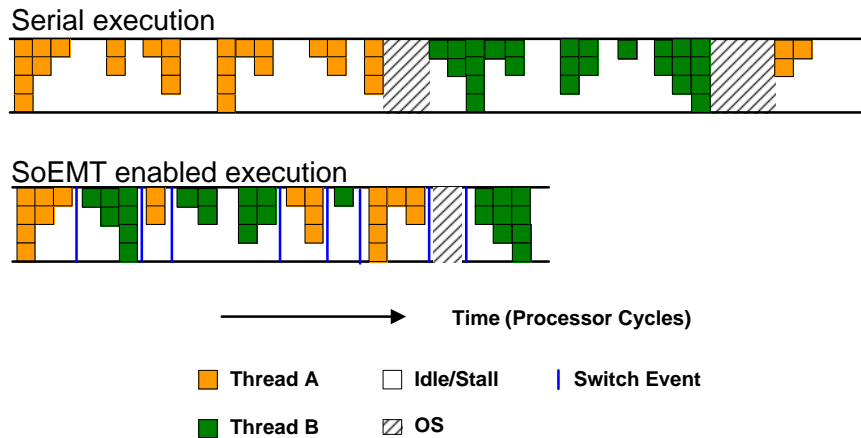
Several events can cause a thread context switch:

- L3 cache miss—An L3 cache miss by foreground thread is most likely to cause memory stall until the fabric resolves the miss. Because of the low latency to access L1 and L2 cache, the L3 cache miss event is the primary switch event.
- L3 cache miss return—An L3 cache miss return event indicates that the L3 cache miss by the background hardware thread is resolved and ready to run. This event can trigger a thread context switch depending on the urgency of hardware threads.
- Hardware timeout—When the thread quantum expires, the thread context switch occurs, which ensures fairness between the foreground and background hardware thread.
- Software switch hint—Hint@pause is an instruction that gives a context switch control to the software. If possible, the foreground thread yields execution to the background thread. The hint@pause instruction is typically called when the foreground hardware thread does not need the core execution resource.

- Power saving mode—When the foreground thread enters the power saving mode, the core resource is yielded to the background thread.

The switch is completely transparent to the application and the operating system.

Figure 1. Example: Comparing non-HT and HT execution on core. This example illustrates one possible scenario in which a serial execution of two independent execution streams can run concurrently on a core, context switching between two hardware threads resulting from a switch event trigger.



Sharing memory hierarchy

The SMT aspect of the Montecito is that memory resources are shared simultaneously or concurrently between two hardware threads. The shared memory resource includes:

- L1 and L2 Translation Lookaside Buffer (TLB)
- L1, L2, and L3 instruction and data caches
- A system interface

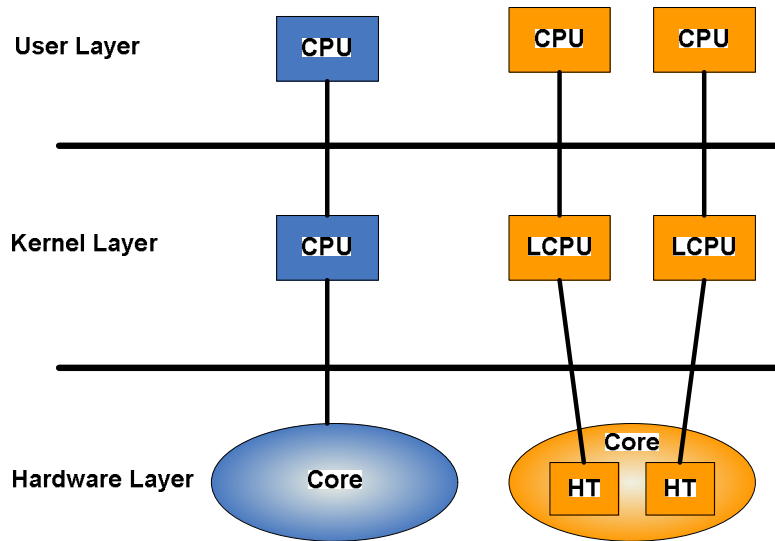
Logical processors

Although the functional units are shared between two hardware threads, there are separate processor states for each hardware thread within the core. That is, each Montecito hardware thread appears as a separate processor to applications and the operating system.

Before HT Technology, the operating system and applications could always expect full access to processors without concern for a core resource contention. However, while functional correctness and binary compatibility can be safely assumed, there are secondary effects of sharing resources between two hardware threads on a core. To address this issue, software abstractions called logical processors (LCPUs) are termed to represent the Montecito hardware threads.

A single-threaded core is referred as a CPU. Unless explicitly stated as hardware thread, a thread refers to the software execution stream. Also, HT and hardware thread are interchangeable terms.

Figure 2. Comparing a single-threaded processor core and multithreaded processor core



The abstraction of LCPU is necessary because each hardware thread is not really a full CPU. While each hardware thread appears like CPU to the user-level applications, they do not behave exactly like a CPU.

LCPU and Hyper-Threading

On HP-UX 11i v3, HT Technology can be enabled at different levels. The Montecito Hyper-Threading feature is enabled or disabled at the system boot time through the firmware setting. If the HT feature is enabled, the operating system can dynamically enable or disable LCPU feature so that kernel threads can be scheduled on each hardware thread.

When HT is on and LCPUs are enabled, both hardware threads on a core are utilized. However, if HT is on and LCPUs are disabled, only one of the hardware threads on a core is utilized—effectively transforming it into a single-threaded core. The LCPUs can be dynamically disabled using `PAL_HALT_LIGHT` functionality. Specifically, this call reduces the power and eliminates core resource consumption by stopping instruction execution, but it still maintains cache and TLB coherence in response to the external interrupt requests. Any unmasked external interrupt brings the disabled hardware thread to the normal state, in which LCPU is enabled.

Within the kernel, each LCPU represents a separate run queue, and the threads in that run queue are scheduled to run on the corresponding hardware thread.

The key benefits of this approach are that irrespective of the system configuration, all interfaces available for general use by applications that deal with CPU IDs (for example, `mpctl(2)`, `pstat(2)`, `pset_ctl(2)`, etc) expose logical CPUs as CPU objects. Therefore, there is no impact to the existing applications.

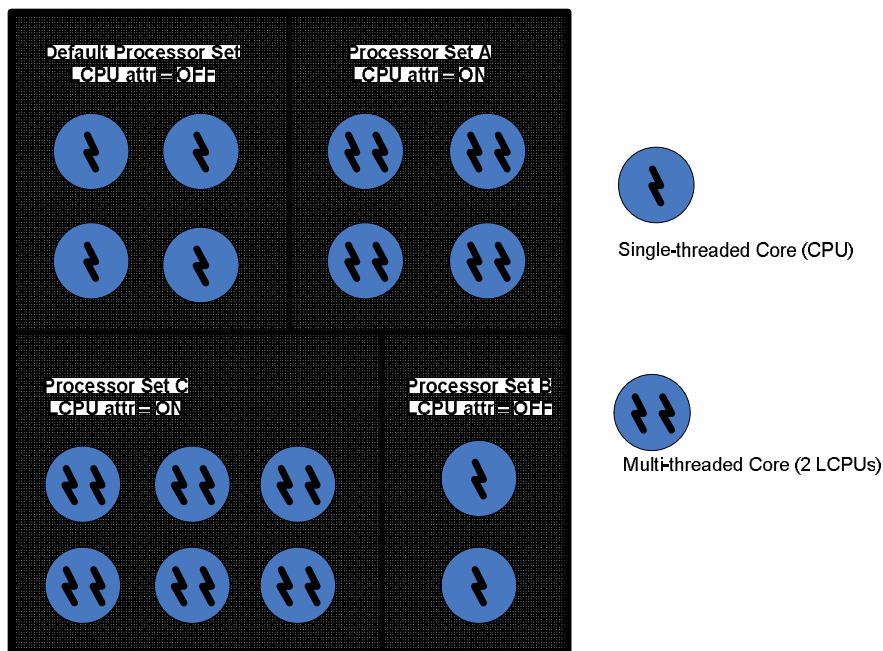
LCPUs and processor sets

While many applications perform well with LCPUs on Hyper-Threading based systems, some applications might not—some might actually perform worse. For this reason, LCPUs are integrated

with HP-UX's existing partitioning technology, specifically processor sets (PSETs), as a way to offer simultaneous availability of CPUs and LCPUs in a single operating system instance.

Specifically, the definition of HP-UX PSETs is expanded to include a new attribute. This attribute allows LCPUs to be dynamically enabled or disabled for all cores within a PSET on an HT capable system. HP-UX customers can tailor each system specifically to the needs of the applications being run on that system. In this model, PSETs provide "quarantine zones" for applications that do not perform well with HT by allowing the application to run on processors without LCPUs. Other applications that perform well with hyper-threads can run in PSETs with LCPUs enabled. While this is a new use of PSETs, it allows a small granularity of control for HT with the use of an existing and well tested mechanism. Furthermore, this use provides control for HT manipulation without requiring programming changes for applications. This approach is fully dynamic and allows all customer applications to coexist within one operating system instance, regardless of their tolerance of HT.

Figure 3. Partitioning a single operating system instance with multiple processor set to support mix of LCPU attribute ON and LCPU attribute OFF



Because the LCPUs on the same core share common resources, the interfaces that deal with workload migration and CPU assignment require operations at the core granularity. For example, if a PSET operation reassigns an LCPU, depending on the LCPU attribute setting, it must also reassign that sibling of the LCPU as well. Otherwise, the resulting configuration would violate the PSET contract, which specifies that applications bound to a PSET have complete control of all processor within that PSET.

This configuration contract also applies to other operations, including iCAP, vPAR, and OL*, to control system re-configuration at the core granularity.

Application considerations of Hyper-Threading Technology

While most applications would benefit from the HT Technology, some applications might experience a loss of performance.

Considerations for regular commercial applications

Generally, applications do a fine job of optimizing system resources (for example, memory resources), but most commercial applications are not especially considerate in handling cache access patterns. Consequently, applications (even the tuned ones) typically spending 60% or more of the time stalled, waiting for the resolution of a cache miss with a copy to the cache from elsewhere in the memory hierarchy. With most applications spending no more than 40% of their time actually retiring instructions, having two threads with those kinds of characteristics share a single core is reasonable and makes higher utilization and total throughput possible.

Given the large stall time for typical application processes, some applications could see some throughput benefit from HT. This benefit can range upward of 15 to 20%, but it is more likely to be less than 10%. Although large benefit might not be typical, the potential benefit of HT warrants consideration for an application's use of HT.

Your mileage may vary: Prove that HT is good for the application

The analysis of an application on a system with HT is the most effective way to determine the application's performance characteristics. Performance measurement tools such as Caliper can help do this. Determine whether HT will provide a benefit—do not automatically assume that HT will provide an application benefit without a full analysis.

HT provides opportunity for greater throughput, not better response time

From a throughput perspective, a core with HT enabled might provide up to 130% of a core with HT disabled. From a strict instruction execution rate, each HT would be running at about 65% of a core without HT. Applications with the ability to take advantage of additional threads of execution for additional throughput might see an HT benefit. Applications with a dependency solely on response times based on processing speed are likely to see worse performance with the use of HT.

Applications with high processor utilizations may benefit from the use of HT

Applications that are at a point of high utilization such that additional hardware threads would provide additional throughput are potential candidates for HT. Enabling HT is like adding processors. If the processor utilization is too low (below 75 to 80%), the application might not see additional throughput from the use of HT.

The ability of applications to scale can affect HT performance benefits

Applications that do not scale well with each additional non-threaded IPF processor core (that is, Montecito with HT disabled) usually do so because of resource sharing bottlenecks. Such applications will likely not see any performance benefit with HT enabled. Such applications could see hyper-threads on the same core wrestling for the same shared resources as two slower CPUs, thus exacerbating some scalability problems.

Applications with tighter cache management might not benefit from HT

Commercial applications that manage cache resources carefully do not spend quite as much time stalling as typical applications. Such applications would spend less time waiting for the resolution of a cache miss and increase the probability of contention between the threads sharing a core.

Applications with a CPI of less than 1.5 might not benefit from HT

If the analysis of an application shows a cycles-per-instruction (CPI) count of less than 1.5, HT might not provide any advantage for additional throughput. Such applications are already making efficient use of core resources. Their use of HT would increase the probability of contention between the threads sharing a core, likely with disappointing performance.

Applications with a few stalling L3 cache misses might not benefit from HT

If the analysis of an application shows few stalling L3 cache misses, HT might not provide any advantage for additional throughput. Such applications might not be sufficiently active with a large enough working set to cause cache misses beyond L1 and L2 and would not trigger the L3 miss events necessary to take advantage of HT. Their use of HT would increase the probability of contention between the threads sharing a core, likely with disappointing performance.

Application with processes that just fit in the L3 cache might not benefit from HT

Application processes that just fit into the L3 cache execute very quickly with HT disabled. With HT enabled, two such processes occupying neighboring hardware threads on a core would suddenly find contention for cache resources where there was none before. Any advantages of being able to fill cache stall time would be quickly overcome by the sudden cache contention that would occur.

Platforms with lower memory latencies yield less advantage for HT use

HT increases throughput by filling L3 cache miss stall time with instruction execution. Lower latency platforms provide less time to fill because stalls take less time to resolve cache misses. Consequently, lower latency platforms see less HT benefit than their higher latency counterparts.

High bus utilization resulting from a combination of application and platform configuration might provide less advantage for HT use

An application that scales well with a high CPI and a high rate of stalling L3 cache misses might not gain from HT if bus utilization is already very high without HT. While such applications can benefit from the additional computing that HT would provide, the resulting overhead from bus saturation could hurt overall throughput rather than help it.

TPC-C type applications might see a benefit from HT

Transaction processing applications that exhibit behavior similar to TPC-C benchmarks can benefit from HT. TPC-C benchmarks can achieve greater than 20% performance improvement from the use of HT. Applications of similar character have the right mix of CPI and stalling L3 cache misses to take advantage of potential throughput gains from the use of HT.

An application composed of homogeneous processes might see a benefit from HT

An application composed of homogenous processes might see a general performance benefit from the use of HT. It will make good use of processor cache resources while taking advantage of cache miss stall time to increase core utilization and get more throughput performance.

However, it is more effective to pair, on a single core, an extremely cache-efficient process with an extremely cache-inefficient process on a single core. Pairing two extremely cache-efficient (or cache-inefficient) processes on the same core can diminish the throughput gains that HT can provide to an application.

A general mix of application processes might see a benefit from HT

On a highly utilized machine, mixes of general application processes that fit comfortably in the processor cache and do not require much process locking or other process coordination might see a general performance benefit from the use of HT.

Use normal HP-UX threading and locking mechanisms

There are potential interactions between hardware threads executing on the same core. As such, two hardware threads vying for locks around critical code sections must coordinate to prevent any pathological behavior that would damage application performance. Standard HP-UX library threading and locking mechanisms have been enhanced to account for the behavior of hardware threads. Consequently, application developers should use standard threading and locking interfaces and only implement their own threading and locking mechanisms with great care.

HP-UX can simultaneously accommodate both HT-on and HT-off requirements

Some applications might have processes that work best with HT enabled and other processes that work best with HT disabled. The LCPU model in HP-UX allows such applications to run HT-friendly processes in one PSET with HT enabled while running HT-unfriendly processes in another PSET with HT disabled. Such a mixture of HT-on and HT-off allows applications to maximize the performance benefits of HT.

Do not base conclusions of performance based only on simple benchmark behavior

It is relatively easy to construct an artificial code stream that present hardware threads negatively. However, such artificial benchmarks are not necessarily representative of commercial applications and should not be used as a basis to determine whether HT provides an advantage for a given application.

Considerations for high-performance technical computing applications

The previous advice for commercial applications holds true for high-performance technical computing as well. However, high-performance technical applications are tuned to optimize access patterns for the whole memory hierarch, including caches, which means much less stall time with thread switching mainly done at the end of a thread's time quantum. For such an application, a small (but non-zero) thread switching overhead can cause a performance loss with HT enabled.

Most technical applications might see no or negative benefit

Most high-performance technical applications (for example, LINPACK) are very cache aware and tuned with a low CPI. On many such applications, stalling L3 cache misses are rare enough that HT switch overhead becomes a significant factor and results in worse performance with HT enabled. Many applications also tune to the exact cache size and would see significant delays from cache contention because of the resource sharing required with HT.

Your mileage may vary: with technical computing

Just as with commercial applications, the analysis of an application on a system with HT is the most effective way to determine the application's performance characteristics. Performance measurement tools such as Caliper can help do this.

Enabling Hyper-Threading Technology

There are several levels of settings to enable the HT Technology. The firmware setting controls the physical settings of the HT feature, and the operating system tunable setting controls the LCPU feature of the operating system instance.

Enabling at the firmware level

At the Extensible Firmware Interface (EFI) shell, you can configure the system to enable or disable the HT Technology. The HT setting affects all the processors in the physical partition. Therefore, if multiple partitions are configured in the system, different settings can be applied to each physical partition.

Table 1. HT feature settings

Firmware setting description	Command
Disables the HT feature	<code>cpuconfig threads off</code>
Enables the HT feature	<code>cpuconfig threads on</code>

System configuration information

Two new `sysconf(2)` options were added to allow applications developers to query the system's HT feature capability and current state of HT feature setting.

Table 2. `sysconf(2)` options to query the HT feature

<code>sysconf(2)</code> option	Description
<code>_SC_HT_CAPABLE</code>	Determines if the system has the capability to enable HT feature
<code>_SC_HT_ENABLED</code>	Determines if the system currently has the HT feature enabled

The query can also be made from the `getconf(1)` command.

LCPU attribute setting of the default PSET

Since 11i V2, the PSET is core HP-UX functionality. When a system boots, it is initially configured with a single PSET, called the default PSET, which contains all the processors in the system. Although the user might never create additional PSETs, users are implicitly using PSET functionality.

Because some users might never use PSET functionality in their environment and would want to keep the LCPU attribute setting across a reboot, enabling and disabling the LCPU attribute is done through the tunable operations.

A new tunable, `lcpu_attr(5)`, is introduced to dynamically enable or disable the LCPU attribute of the default PSET for several reasons:

- There is no need for the user to learn about the PSET functionality. The tunable is a well established infrastructure to modify the system settings, and if the user does not intend to use the PSET functionality, setting the LCPU attribute of the default PSET implicitly affects the entire system.
- It provides a mechanism to dynamically enable or disable LCPU attribute of the default PSET without using the PSET system calls or related commands. The legacy behavior dictates that the PSET system calls do not manipulate the default PSET attributes.
- It also provides persistent LCPU attribute setting across reboot, so the user does not have to reset the default PSET LCPU attribute after reboot.

Table 3. Tunable commands for `lcpu_attr(5)`

Description	<code>kctune(1M)</code> commands
Obtains current LCPU attribute setting of the default PSET	<code>kctune lcpu_attr</code>
Enables the LCPU feature in the default PSET	<code>kctune lcpu_attr=1</code>
Disables the LCPU feature in the default PSET	<code>kctune lcpu_attr=0</code>
Sets the LCPU feature in the default PSET to default setting, which is disabled	<code>kctune lcpu_attr=Default</code>

The following table represents the different combination of the firmware HT feature settings and LCPU attributes of the default PSET settings.

Table 4. Initial HT state and the default PSET LCPU attribute

		default PSET LCPU attribute setting	
		LCPU OFF	LCPU ON
HT FW setting	HT OFF	Ü	û
	HT ON	ü	Û

The Montecito base systems are initially shipped with HT feature disabled and the default PSET’s LCPU attribute disabled. The HP-UX 11iv3 installation process enables the HT feature. Because the performance of HT feature enabled and the default PSET LCPU attribute disabled setting delivers equivalent performance of HT disabled, this setting offers the most flexible environment for users.

If the system boots with HT feature disabled, the LCPU attribute of the default PSET defaults to the disabled setting, which will persist across a reboot.

Changing HT setting for the next boot

To avoid unnecessary steps of setting the HT feature at the EFI prompt, the `setboot(1m)` command has been enhanced to enable or disable HT feature on the Montecito-based platforms. The configuration change is for the next and consecutive boots. This feature is equivalent to configuring the HT feature setting from the EFI shell.

Table 5. Initial HT state and the default PSET LCPU attribute

setboot(1M) command description	Command
Enables HT feature on next reboot	<code>setboot -m on</code>
Disables HT feature on next reboot	<code>setboot -m off</code>

PSET operations

The interface definition of the existing PSET related system calls has not changed, but because the system configuration is at the core granularity level, several subtle behaviors are observed when using the PSET related system calls. See “Processor Sets – A Technical White Paper” for further information regarding PSETs.

pset_create(2)

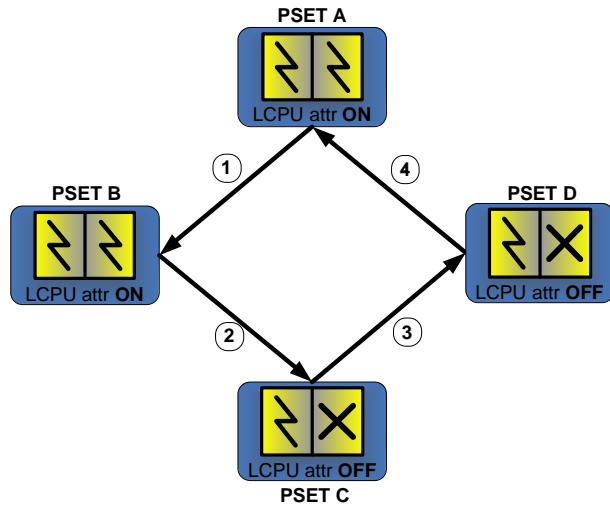
When a new processor set is created using the `pset_create(2)` system call, the LCPU attribute of the default PSET is inherited. So, at the time of creation if the default PSET’s LCPU attribute is:

- OFF—A newly created PSET’s LCPU attribute is set to OFF.
- ON—A newly created PSET’s LCPU attribute is set to ON.

pset_assign(2) and pset_destroy(2)

The assignment of core can be changed by two PSET related calls: `pset_assign(2)` and `pset_destroy(2)`. The `pset_assign(2)` system call migrates a core from the source PSET to the target PSET. However, `pset_destroy(2)` migrates a core from the source PSET to the default PSET. Based on the LCPU attribute setting of the source and target PSET, the topological view of the CPUs and LCPUs can change unexpectedly.

Figure 4. PSET LCPU attribute and CPU and LCPU configuration transition diagram



- Note:**
- Enabled LCPU
 - Core with 2 HW threads (2LCPU's)
 - Disabled LCPU
 - PSET
- ① core move from a pset with LCPU attr **ON** to a pset with LCPU attr **ON**
 - ② core move from a pset with LCPU attr **ON** to a pset with LCPU attr **OFF**
 - ③ core move from a pset with LCPU attr **OFF** to a pset with LCPU attr **OFF**
 - ④ core move from a pset with LCPU attr **OFF** to a pset with LCPU attr **ON**

For example, reassigning a core from PSET A, which has its LCPU attribute ON, to PSET D, which has its LCPU attribute OFF, would result in only one of two LCPU's appearing in the PSET D after the reassignment of the core completes.

In another example, the default PSET's LCPU attribute is set to ON, and a PSET with LCPU attribute OFF is destroyed. The cores in the destroyed PSET are reassigned to the default PSET with all LCPU's enabled.

If the target and the source PSET LCPU attributes are the same, the reassigned core's LCPU's does not appear to change.

pset_setattr(2)

A new attribute, `PSET_ATTR_LCPU`, is introduced to the PSET functionality to dynamically enable or disable LCPU feature at the PSET boundary. The LCPU attribute of the non-default PSET (or user-created PSET) can be toggled using the `pset_setattr(2)` system call.

A caller with appropriate permission can configure the PSET to enable or disable the LCPU feature. The LCPU attribute is supported only on the platforms that have the HT feature enabled.

Table 6. LCPU_ATTR_LCPU attribute values

PSET_ATTR_LCPU values	Description
PSET_ATTRVAL_ON	This attribute enables the LCPUs in the PSET. This value is the default for the systems with the HT feature enabled.
PSET_ATTRVAL_OFF	This attribute disables the LCPUs in the PSET. This value is the default value for the systems without the HT feature capability and HT feature disabled.
PSET_ATTRVAL_DEFAULT	Depending on the HT feature setting, the default value can vary.

The PSET_ATTR_LCPU attribute for the default PSET can be set using the `setttune(2)` command programmatically.

psrset(1M)

The PSET command, `psrset(1M)`, is enhanced to support the new LCPU attribute. It toggles the LCPU attribute of the non-default PSET.

Table 7. Initial HT state and the default PSET LCPU attribute

setboot(1M) command description	Command
Enables the LCPU attribute of the target PSET	<code>psrset -t <pset id> LCPU=ON</code>
Disables the LCPU attribute of the target PSET	<code>psrset -t <pset id> LCPU=OFF</code>

To toggle the LCPU attribute of the default PSET, use the `kctune(1M)` command.

Interaction with Online Addition and Deletion

Several Online Addition and Deletion (OL*) operations are impacted by the systems with Hyper-Threading: HP Instant Capacity (iCAP), HP Virtual Partitions (vPAR), and CELL/CPU OL*. Because enabling and disabling the LCPU attribute effectively adds or deletes one of the hardware threads in cores belonging to a particular PSET, these operations are serially coordinated with PSET operations.

Additionally, OL* operates at the core level. For example, two hardware threads on the same core cannot belong to different vPARs.

CPU Online Addition (OLA) maintains the legacy behavior of initially assigning to the default PSET. Depending on the LCPU attribute of the default PSET, newly added cores can be single-threaded or multi-threaded. CELL OLA is considered as a set of CPU OLAs.

CPU Online Deletion (OLD) also maintains the legacy behavior, where any core can be deleted, except the core with the Monarch. Also, CPU OLD can fail when some PSET attribute prohibits the removal of the last processor for core reassignment or CPU OLD.

For further details, see the respective manuals and white papers on iCAP, vPAR, and CELL/CPU OL*.

Time accounting

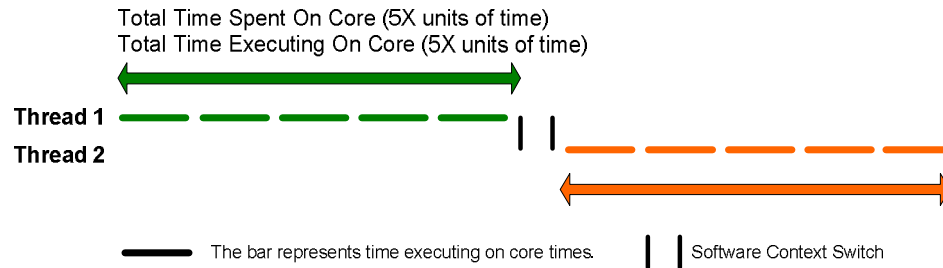
Before the introduction of the HT technology, a single-threaded execution stream had exclusive access to the core resource that included the machine cycles while retiring instructions or stalled on system resources. Therefore, the core utilization was measured by keeping track of Interval Timer Counter (ITC) at varying operating system states, such as user, sys, idle, and so on.

There are two basic views on the meaning of the measurement by the operating system at various states:

- Time-Executed-on-Core (TEoC)—This method consists of measuring the actual cycles utilized by the instruction stream between two arbitrary points in time (typically operating system state transitions points) while having exclusive access to the core.
- Time-Spent-on-Core (TSoC)—This method consists of measuring the time, in unit of cycles, spent by the instruction stream between two arbitrary points in time (typically operating system state transition points).

Both measurements consist of cycles while retiring instructions and being stalled on system resources. On the single-threaded core, the measurement of both views yields similar results. Whether the processor core is actively retiring instructions or is stalled on system resources, the execution stream does not yield the core resource and have exclusive access to the core until the operating system forces a context switch to another execution stream. Therefore, the core utilization measurement by reading ITC was accurate and consistent.

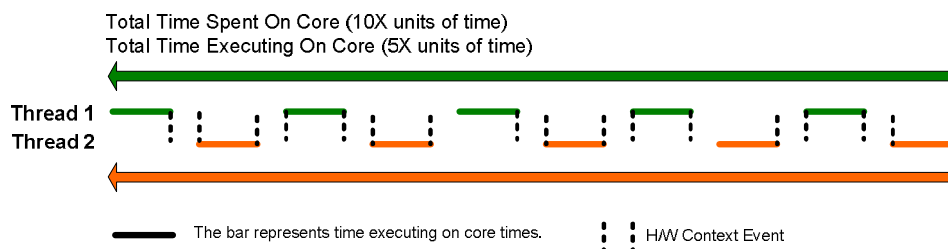
Figure 5. Time accounting on single-thread core (CPU)



As illustrated on the Figure 7, until the operating system switches to another instruction context, it guarantees that the current instruction stream monopolizes the core resource. Therefore, there is a 1:1 ratio of TEoC and TSoC. Hence, it does not matter which view is used to measure the core utilization.

However, with the HT Technology on the Montecito system, because the core is a shared resource between two hardware threads, the current method of reading ITC does not provide accurate core resource utilization by each HT. The Interval ITC is a shared register. The ITC continues to advance, irrespective of which hardware thread is currently executing on the core.

Figure 6. Time accounting on Hyper-Threading (two LCPUs)



Because the hardware context switch is completely transparent to the operating system, measuring the core resource utilization of a hardware thread by reading ITC provides accurate measurement of TSoC, but it does not provide accurate measurement of TEoC.

As illustrated in Figure 8, between two arbitrary points in time, the measurement of TSoC and TEoC are different. They are no longer 1:1 ratio because the instruction stream no longer has exclusive access to the core resource.

On HP-UX 11.31 v3, HP-UX report TEoC as the default view of measuring the core utilization. When HT is enabled, the TSoC approach also accounts for core utilization of the sibling hardware thread on the same core, which would violate the design to make the hyper-thread appear as a separate processor to applications and the operating system.

However, because the information from ITC is the only source of measurement, the operating system adjusts to reflect TEoC. This adjustment is known as the 50/50 approximation scheme. The hardware thread scheduler fairly distributes the core resource when similar or dissimilar workloads are scheduled on a sibling hyper-thread on the same core. Therefore, the 50/50 approximation scheme is feasible.

When a sibling hyper-thread is idling, it does so by spinning tightly issuing hint@pause instruction. The 50/50 approximation scheme adapts to this situation and adjusts accordingly. When a hyper-thread is doing useful work and its sibling is idling, 90 to 99% of core cycles is utilized by the non-idling hyper-thread; thus, it nearly matches the performance of single threaded core.

Several experiments, using micro benchmarks and real-world applications to determine the viability of 50/50 approximation scheme, result in 85 to 90%, or better, accuracy.

The future Itanium 2 chips might support per HT Resource Utilization Counter (RUC). The 50/50 approximation scheme is a stopgap design until RUC is available. When RUC is available, it will replace the 50/50 approximation scheme.

Core and LCPU utilization

It is important to distinguish between core utilization and LCPU utilization. Typically, when LCPUs achieve full utilization, the cores are fully utilized. However, full core utilization is possible without full LCPU utilization when a LCPU is doing a useful work and its sibling LCPU is idling. An idling LCPU aggressively issues hint@pause instruction to yield its core resource to the sibling LCPU, which is doing a useful work. It would appear that one of the sibling LCPUs is idling, but the core resource is nearly all used by the LCPU doing the useful work. Low LCPU utilization typically reflects low core utilization.

Variable core utilization

The time accounting and core resource utilization by the LCPUs are no longer linear. The core utilization varies greatly depending on the workload behavior, and the time accounting reflects this behavior even for future architectural enhancement, such as the use of RUC.

Per thread and process utilization

Thread and process utilization measurement adopts TEoC to reflect the actual core resource utilization. The virtual and profile timers also rely on the TEoC.

Comparing throughput and accuracy in time accounting

The Montecito with HT Technology is designed to increase the throughput by maximizing underutilized core resource. However, under certain circumstances, there is greater importance to maintain accurate time accounting rather than increased throughput. The 50/50 approximate

utilization scheme on the HT environment provides reasonable level of accuracy, but if absolute resource utilization measurement is a requirement, HP recommends disabling HT or disabling the LCPU attribute.

Time accounting extension

The following fields are added to the `pstat_getprocessor(2)` interface. Using existing fields, which report TEOC, and new fields, which report TSoC, resource utilization is obtained.

Table 8. `pstat_getprocessor` extension for time accounting

Fields	Field description
<code>psp_raw_usercycles</code>	64-bit cycles counter for user mode execution. This non-adjusted counter represents the Time-Spent-On-Core.
<code>psp_raw_systemcycles</code>	64-bit cycles counter for system mode execution. This non-adjusted counter represents the Time-Spent-On-Core.
<code>psp_raw_interruptcycles</code>	64-bit cycles counter for interrupt mode execution. This non-adjusted counter represents the Time-Spent-On-Core.
<code>psp_raw_idlecycles</code>	64-bit cycles counter for idle mode execution. This non-adjusted counter represents the Time-Spent-On-Core.
<code>psp_logical_cpu_idlecycles</code>	64-bit cycles counter to measure the wall-clock time spent in idle by each LCPU. This is a running total counter that LCPU would have executed in the idle cycles, but because of <code>hint@pause</code> instruction from idling LCPU, it is possible to donate most of its cycles to the sibling LCPU, which has a meaningful workload. Using this counter, it is possible to calculate both utilization of core and LCPUs on the same core.

Topological information

The existing interfaces to obtain the system and PSET topological information are extended to provide complete information about the hierarchical system configuration.

`pstat_getprocessor(2)`

The additional fields in the `pst_processor` data structure provide the hierarchical representation of LCPU, CPU, core, socket, and Locality Domain (LDOM). Because the `pstat_getprocessor(2)` command returns the information at CPU or LCPU granularity, depending whether the specified processor is in the PSET with LCPU attribute ON or OFF, you might want to

One possible method of providing information whether the specified processor is CPU or LCPU is:

```

struct pst_processor psp;
pset_attrval_t attr;

ret = pstat_getprocessor(&psp, sizeof(struct pst_processor), 1, 4);
if (ret == -1) {
    perror("pstat_getprocessor() failed\n");
} else {
    ret = pset_getattr(psp.pst_pset_id, PSET_ATTR_LCPU, &attr);
    if (ret != -1) {
        if (attr == PSET_ATTRVAL_ON) {
            printf("spu 4 is LCPU\n");
        } else {
            printf("spu 4 is CPU \n");
        }
    }
}

```

```

    } else {
        perror("pset_getattr(PSET_ATTR_LCPU failed\n");
    }
}

```

Table 9. pstat_processor(2) fields and descriptions

Fields	Field description
psp_socket_id	Socket ID of a CPU or LCPU.
psp_core_id	Core ID of a CPU or LCPU.
psp_logical_thread_id	Logical thread ID of a CPU or LCPU.
psp_sibling_cnt	On Hyper-Threading enabled machine, returns the number of sibling hardware threads on the same core. The count does not include self, so on the Hyper-Threading disabled system, returns 0.
psp_sibling[PSP_MAX_SIBLINGS]	On Hyper-Threading enabled machine, returns an array of LCPU IDs for sibling hardware threads on the same core. The invalid entries in the array returns "-1"

pstat_getdynamic(2)

New system-wide counters are added to represent the total number of active cores and the maximum number of cores supported by the system.

Table 10. pstat_getdynamic(2) fields and descriptions

Fields	Field description
psd_active_cores	Number of currently active cores in the system
psd_max_cores	Maximum number of cores supportable on the system

mpctl(2)

The topological information section of the `mpctl(2)` command is extended to provide the hierarchical information of CPU, LCPU, core, and LDOM.

Table 11. mpctl(2) system call options to find processor core topological information

Fields	Field description
MPC_GETNUMCORES_SYS	Returns the number of enabled/active processor cores in the system
MPC_GETFIRSTCORE_SYS	Returns the ID of the first enabled processor core in the system
MPC_GETNEXTCORE_SYS	Returns the ID of the next enabled processor core in the system after the specified processor core ID
MPC_GETCURRENTCORE	Returns the ID of the processor core of the calling thread
MPC_SPUTOCORE	Returns the ID of the processor core containing the specified CPU or LCPU ID
MPC_GETNUMCORES	Returns the number of processor cores in the PSET of the calling thread
MPC_GETFIRSTCORE	Returns the ID of the first processor core in the PSET of the calling thread
MPC_GETNEXTCORE	Returns the ID of the processor core in the PSET of the calling thread after the specified CPU or LCPU ID

Fields	Field description
MPC_LDOMCORES_SYS	Returns the number of enabled/active processor cores in the specified locality domain
MPC_LDOMCORES	Returns the number of enabled/active processor cores assigned to the PSET in the locality domain

Similar to `pstat_getprocessor(2)`, the default granularity of the processor-level topological information is CPU or LCPU. Different methods are available to generate the topological information, but these two examples are most commonly used to determine the configuration of a system.

Example to build bottom-up topological information

The sample program was executed on a single cell system with eight cores and 16 hyper-threads. Two PSETs (the default PSET and a user-created PSET) are configured with four cores assigned to each PSET.

```

spu_t spu;
spu_t core;
ldom_t ldom;
pset_attrval_t attr;
psetid_t pset;
int ret;

spu = mpctl(MPC_GETFIRSTSPU_SYS, 0, 0);
while (spu != (spu_t)-1) {
    pset = pset_ctl(PSET_SPUTOPSET, 0, spu);
    ret = pset_getattr(pset, PSET_ATTR_LCPU, &attr);
    core = mpctl(MPC_SPUTOCORE, spu, 0);
    ldom = mpctl(MPC_SPUTOLDOM, spu, 0);

    printf("%s %d in core %d assigned to pset %d ",
           "(lcpu_attr %s) in ldom %d\n",
           (attr == PSET_ATTRVAL_ON) ? "LCPU" : "CPU",
           spu, core, pset,
           (attr == PSET_ATTRVAL_ON) ? "ON" : "OFF", ldom);

    spu = mpctl(MPC_GETNEXTSPU_SYS, spu, 0);
}

```

The sample output of the code snippet:

```

LCPU 0 in core 0 assigned to pset 0 (lcpu_attr ON) in ldom 0
LCPU 1 in core 0 assigned to pset 0 (lcpu_attr ON) in ldom 0
LCPU 2 in core 2 assigned to pset 0 (lcpu_attr ON) in ldom 0
LCPU 3 in core 2 assigned to pset 0 (lcpu_attr ON) in ldom 0
LCPU 4 in core 4 assigned to pset 0 (lcpu_attr ON) in ldom 0
LCPU 5 in core 4 assigned to pset 0 (lcpu_attr ON) in ldom 0
LCPU 6 in core 6 assigned to pset 0 (lcpu_attr ON) in ldom 0
LCPU 7 in core 6 assigned to pset 0 (lcpu_attr ON) in ldom 0
CPU 8 in core 8 assigned to pset 2 (lcpu_attr OFF) in ldom 0
CPU 10 in core 10 assigned to pset 2 (lcpu_attr OFF) in ldom 0
CPU 12 in core 12 assigned to pset 2 (lcpu_attr OFF) in ldom 0
CPU 14 in core 14 assigned to pset 2 (lcpu_attr OFF) in ldom 0

```

pset_ctl(2)

The topological information section of the `pset_ctl(2)` command provides the hierarchical information about CPU, LCPU, core, and LDOM of the specified PSET. The information provided by the `pset_ctl(2)` and `mpctl(2)` commands is similar for the specified PSET.

Table 12. pset_ctl(2) system call options to find processor core topological information

Fields	Field description
PSET_GETNUMCORES	Returns the number of cores in the PSET of the calling thread
PSET_GETFIRSTCORE	Returns the ID of the first processor core in the specified PSET
PSET_GETNEXTCORE	Returns the ID of the next processor core in the specified PSET after the specified CPU or LCPU ID
PSET_LDOMCORES	Returns the number of enabled/active processor core assigned to the specified PSET in the locality domain

Appendix A

HP-UX 11i v2 extensions for portability

A subset of HP-UX 11i v3 topological APIs is also available for HP-UX 11i v2 to enable building Hyper-Threading (HT) aware application on HP-UX 11i v2. However, enabling or testing of HT technology on HP-UX 11i v2 is discouraged by Hewlett-Packard because HT has limited support in the HP-UX 11i v2 operating environment. All testing of HT applications should be done on HP-UX 11i v3 (and later) operating environments.

mpctl(2) extension

The kernel patch ID PHKL_35767 provides the complete set of processor core based topological information on HP-UX 11i v2 using the mpctl(2) system call. For the definition and description of fields in the mpctl(2), refer to the previous section "Topological Information."

pset_ctl(2) extension

pset_ctl(2) system call is not extended on HP-UX 11i v2 because the processor core based topological information is redundantly provided by mpctl(2) system call.

pstat_getprocessor(2) and pstat_getdynamic(2) extension

The kernel patch ID PHKL_34912 provides the complete set of the processor core and hardware thread based topological information on HP-UX 11i v2 pstat(2) system call. For the definition and description of fields in the pstat_getprocessor(2) and pstat_getdynamic(2), refer to the previous section "Topological Information."

Special consideration for pstat(2) extensions

The pstat(2) topological information extension required adding new members to the data structures used to pass information between the application and kernel. These members are protected by a compile-time variable `_TOPO_EXTENSION_SUPPORTED`. Applications compiled on HP-UX 11i v2 must use this `#define` at compile time to enable pstat(2) extensions within the application.

Calls to the extended pstat(2) functions made by HP-UX 11i v2 applications compiled with `_TOPO_EXTENSION_SUPPORTED` will return `EINVAL` on HP-UX 11i v2 systems where the kernel patch has not been installed. Applications compiled without this `#define` will work correctly on HP-UX 11i v2 systems with or without the kernel patch.

This approach is to preserve binary compatibility with applications built without the patch on HP-UX 11i v2. By enabling the compile-time variable, the application developers are explicitly agreeing that the kernel patch is required to run correctly.

On HP-UX 11i v3, the compile-time variable `_TOPO_EXTENSION_SUPPORTED` does not exist; therefore, there is no need to enable this `#define` for HP-UX 11i v3 applications at compile time.

pstat(2) data structure extension

```
struct __pst_dynamic {  
  
    ... existing fields  
  
#if defined(_TOPO_EXTENSION_SUPPORT) || defined(_KERNEL)  
    _T_LONG_T    psd_active_cores; /* Number of active cores */  
    _T_LONG_T    psd_max_cores;   /* Number of cores on system */  
#endif /* _TOPO_EXTENSION_SUPPORT */  
};
```

```

struct __pst_processor {
    ... existing fields

#ifdef defined(_TOPO_EXTENSION_SUPPORT) || defined(_KERNEL)
    _T_LONG_T psp_socket_id;          /* socket id */
    int32_t    psp_core_id;          /* core id */
    _T_LONG_T psp_logical_thread_id; /* logical thread id on MT-HW
*/
    _T_LONG_T psp_sibling_cnt;       /* on MT-HW, returns number of
                                     sibling HW threads on the
                                     same core */
    _T_LONG_T psp_sibling[PSP_MAX_SIBLINGS]; /* on MT-HW, returns spu_t
of
                                     sibling HW threads on the
                                     same core */
#endif /* _TOPO_EXTENSION_SUPPORT */
};

```

For more information

See “HP-UX Processor Sets – A Technical White Paper” at <http://docs.hp.com> and “Dual-Core Update to the Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization” at <http://www.intel.com/design/itanium2/manuals/308065.htm>.

© 2007 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

4AA0-7695ENW, Rev 1, March 2007

