64bit SMP NetBSD OS Porting

for TILE-Gx VLIW Many-Core Processor

Toru Nishimura

Sanctum Networks, Pvt. Ltd.

nisimura@sanctumnetworks.com

Abstract

Many-core processor is an attractive platform to run a general purpose OS like NetBSD. We, a team in Sanctum Networks, ported NetBSD 6.0 to 64bit VLIW many-core processor named TILE-Gx. In this paper we introduce distinctive features of TILE-Gx in some depth as the product remains less known in our engineering community. NetBSD porting was made well and smooth than anticipated. We realized that NetBSD 6.0 is a mature SMP OS which provides streamlined kernel structure and offers rich set of kernel API specifically designed for large degree SMP, beyond 32 processor, configuration. As a part of conclusion we mention about some of TILE-Gx NetBSD application area which we're willing to build.

1. Project outlook and time line

This porting project was initiated in mid June 2012 at Tokyo. The goal was to achieve full SMP capabilities on the 36 core TILE-Gx processor and understand the suitability of the TILE architecture for various application development.

- our porting target is Tilera Empower 1U computer with 36 core TILE-Gx processor.
- development environment with cross compiler was made at early July 2012.
- geographically separated two GIT repositories in push-pull synchronizing.
- Japan side hosts are in Sakura VPS and Amazon EC2.
- kernel image linking completed in late July 2012. It contained a lot of stab code guaranteed not to work.
- single core kernel was successful in running ramdisk sysinst program at 2012-10-30.
- since then, kernel stability, GbE driver and SMP has been persuaded.
- as of March 2013 porting project is going active. A number of functionalities, in particular ones for TILE-Gx unique features, are under development.

2. TILE-Gx features

TILE-Gx design was invented by an MIT professor, Dr. Anant Agarwal. It's the latest incarnation of a long time research since 1980s. TILE-Gx is the third generation product. There are two successors, TILE64 and TILEPro which are based on the same TILE architecture approach. Following to two 32bit designs, the third generation model is made 64bit processor.

TILE architecture emphasizes the scalability and low power with a unique on-chip inter-connection technology. TILE-Gx product family has 9 ~ 100 core configuration. Each of core is laid out in tiles-on-wall fashion. Core runs at 1.2GHz clock. Under work load with modest network activity 36 core TILE-Gx is said to achieve 25-30W power consumption in total.

TILE-Gx is a 64bit processor. It has 64bit integer and 64bit floating point operation, 64bit register and 64bit address space by 64bit pointer. These simple characteristics are quite familiar to any of plain old UNIX programmers since the time when MIPS R4000 was introduced at early 1990s.

2.1. Instruction set feature

TILE instruction set architecture is 3 way VLIW. Two or three instructions are in a single 64bit word which is in turn called "instruction bundle." TILE can run up to three instructions simultaneously.

TILE-Gx has sixty four 64bit register file. Table 2-1 shows register definition. 58 of them, including two hardwired zero registers, are general purpose. It contains thread pointer tp register to facilitate thread programming. It shows ABI to define the common register usage program ought to follow.

register	mnemonic	type	usage
0 - 9	**r0 -r9**	saved by caller	arguments/return values
10 - 29	**r10 - r29**	saved by caller	"temporary"
30 - 51	**r30 - r51**	saved by callee	"safe across call"
52	**r52**	saved by callee	frame pointer
53	**tp**	dedicated	thread pointer
54	**sp**	dedicated	stack pointer
55	**lr**	saved by callee	return address
56	**sn**		always zero
57 58 59 60 61 62	**idn0** **idn1** **udn0** **udn1** **udn2** **udn3**	onchip network communication	I/O Dynamic Network 0 I/O Dynamic Network 1 User Dynamic Network 0 User Dynamic Network 1 User Dynamic Network 2 User Dynamic Network 3
63	**zero**		always zero

Table 2-1 register assignment for TILE-Gx ABI

Note that TILE offers a rather large number of, 10, arguments in register for function call. The return values are placed in the same set of register for arguments. This means r0 register content will be destroyed quite often when a function exits. Better to remind it as one of TILE debugging tips.

Six registers are reserved for inter-core communication via on-chip network named UDN and IDN. These registers work as FIFO ports much like FSL "fast-simplex-link" in MicroBlaze processor. Reading from empty register or writing on occupied register may cause the processor to stall until condition meets.

Registers are shared resource among VLIW concurrent execution flow. Each subroutine has a single entry point and a single exit. No parallel subroutines are in action. Register conflicts in parallel execution is considered program error. It brings undefined / unexpected values in register file. Register conflict avoidance is the programmer's responsibility.

TILE instruction has two different formats; X-format for 2 instruction in a bundle and Y-format for 3 instruction in a bundle. Basic arithmetics is done in 3 operand form. As register file is 64 in size, 3-operand instruction requires 18bit = 6 x 3 for register designation plus some more for opcode. 3-in-1 64bit bundle is considered rather tight encoding, however, contributes instruction density much. 2 register arithmetics have signed 8bit immediate or signed 16bit immediate value. The latter instruction belongs to 2-in-1 bundle X-format as it needs to have longer encoding. Unoccupied instruction slot in a bundle is filled with NOP which works as a flowing bubble in execution pipeline.

The most notable difference from conventional CISC / RISC instruction set is the lack of "register indirect with offset" addressing mode. TILE has no "LW R3, 0x178(R2)" style memory access. This means that local variable on stack and/or (C language) struct member must be accessed through an explicit pointer in a temporary register to refer the target address. It's a stark contrast to the case where "register indirect with offset" addressing mode can achieve load / store operation with "base register + immediate value offset" very handy for local variable and struct member. This is another tip for TILE programming to remember.

2.2. Other useful instruction

TILE instruction set has an orthogonal set of atomic math / lock instructions.

fetchadd	Atomic addition
fetchand	Atomic logical AND
fetchor	Atomic logical OR
cmpexch	compare-and-swap

Table 2-2 atomic instructions

All of them have two variations for 8byte operation and 4byte operation. These instructions are comfortably useful to implement NetBSD atomic_ops(3) routines. They are well defined set of MP-safe atomic operations and widely used in SMP NetBSD kernel construct and/or parallel programming library like pthread(3).

Cmpexch instruction is for CAS "compare-and-swap" or TAS "test-and-set" operation. It works like as Intel cmpxchg instruction. Note that it's not based on LL/SC synchronize model found in MIPS, Alpha, PowerPC and ARM64. TILE cmpexch works with accompanying "CmpValue" SPR register. Locking primitives can be implemented with it in usual manner.

TILE-Gx has rich set of DSP and SIMD instruction. It also has some fancy instructions. A set of bit field operations, CLZ (count-leading-zeros) and CRC32 polynomial math for hashing / checksum and so on.

TILE-Gx floating point math does not have dedicated register set. FP instruction uses GP registers for source / destination operands.

2.3. Address space

TILE-Gx has 42bit effective address bit out of 64bit VA virtual address pointer. Virtual address is separated in upper 2TB space and lower 2TB space. There is a large void in between. VA[63:41] is either of all-0 or all-1, that is, sign extended from VA<41> value.

TILE-Gx has no MIPS KSEG0 / KSEG1 / XKPHYS like address segment exists to distinguish cache nature. Software is in charge of address inhabitation and cache nature control with help of smart TLB usage.

2.4. Layered protection

TILE architecture provides four level protection scheme. Level is ranging from 0 least protected to 3 most protected. It allows to build layered protection domains which run protected programs in each level.

PL0	User applications
PL1	Guest OS
PL2	Tilera Hypervisor
PL3	“ virtual machine monitor” (who knows what really it is.)

Table 2-3 TILE protection level

Program runs in low order protection level is inhibited to touch resources in hight order level. Each core runs one of four protection level. Current protection level of individual core is called CPL. Control transfer is done by executing dedicated instruction; swint0, swint1, swint2, swint3. NetBSD uses swint1 instruction for system call to be issued by applications programs.

There is a large set of SPR registers. mfspr / mtspr instructions operate them. SPR number is encoded in 14bit. Most of SPR registers have their own ML “minimal protection level” value to arbitrate which level (0 ~ 3) of program can access to. MPL is the basis of layered protection for TILE runtime environment.

2.5. TLB and TSB

TLB plays a central role in TILE architecture. In TILE architecture TLB dost not just make VM virtual memory possible but also realizes chip-wide global cache coherency. TILE TLB entry is designed to be multi-core aware. TLB entry optionally holds the location of core in chip (in X-Y coordinate) to track and identify how TLB entry to tell VA-PA mapping is tied with a specific core.

Like as most of modern processors, TILE TLB is software managed. TILE-Gx has independent TLB stores for instruction and data; 16 entry iTLB and 32 entry dTLB. Note that TLB is a shared resource among programs which run in different protection domain. The 42bit VA space is also shared among them. Tilera Hypervisor reserves some of TLB entries for its own. Remaining is free for guest OS and application programs to use.

The TLB management strategy is modeled after SPARC processor. TILE uses “TSB” and “TTE” nomenclature for the very same purposes.

TSB “translation store buffer” is a software extension of TLB. TSB holds a super set of TLB in main memory. It works as a staging area to inject TLB entry into processor's iTLB or dTLB. HV is in charge for TLB miss handling. It always consults with TSB content in action. Guest OS can only operate TSB store. As TLB is one of highly sensitive shared resource among various programs, guest OS can not make access TLB. TSB is normally reserved inside protected guest OS memory area. TILE TSB is a unified one to hold iTLB entries and dTLB entries. The approach is different from SPARC64 which has iTSB and dTSB in parallel. TTE “translation table entry” is the software defined intermediate format of TLB entry.

On TLB miss HV takes control to run TLB refill operation. It searches first the offending TLB entry in TSB store. If the target entry is found, HV injects it to either of iTLB or dTLB and complete refill operation. If HV finds TSB has no such entry, then it posts a request for guest OS to come in and solve this “TSB miss” condition. Guest OS, in turn, responds to the TLB miss exception identifying it as genuine access error or recoverable fault condition. The rest of operation is identical to popular software managed TLB processors. If guest OS finds the exception is true TLB refill case, it adds the offending TLB entry into TSB store and returns. HV will take care the refilling. If guest OS finds the exception access error or protection violation, it performs its way to handle the cases.

TILE has ASID “address space identifier.” ASID is to improve TLB hit ratio, that is, better VA->PA translation efficiency. It's as normal as and identical to other ASID processors. TILE ASID is 8bit, offering 256 individual address spaces to be distinguished for TLB lookup operation. Some literatures incorrectly mention that ASID is an extension of VA, like saying it realizes concatenated 8 + 42 address bit. ASID is to virtualize TLB, or to make imaginary multiple TLB stores which are numbered and iterated by ASID. ASID demands a smarter VM to operate. This topic will be discussed in a later section.

As other processors do, TILE processor handles many kinds of interrupt / exception. Device asynchronously posts variety of requests and different types of exception happens while a processor is in action. TILE uses IPI “inter-processor interrupt” not only for pure inter-processor messaging, but also for I/O device interrupt notification. As TILE integrated on-chip devices are located apart of core and notification comes across on-chip network, it'd be reasonable to use IPI laminating many into a single form.

2.6. iMesh on-chip inter-connect

Processing core is laid out in a tiles-covering-wall fashion with mesh shape inter-connect to couple each other. Inter-connect has X-Y / street-avenue like layout. At each crossing is an independent switch processor to tie a computing node with the entire switch network. Tilera names it iMesh technology.

Switch processor is 16bit RISC to run low latency and high bandwidth switching function through limited number of signal connections. Besides of 4 paths for N, E, S and W directions to neighboring switches, one switch data-path is coupled with processor's L2 cache. Data stream travels through L2 first, then either of L1 iCache or dCache reaching to a processing core.
The inter-connect offers UDN "User Dynamic Network" and IDN "I/O Dynamic Network" for general purpose on-chip streaming and messaging communication. Total 6 register of TILE-Gx processor are assigned to accommodate the ease of programming.

It should be reminded that iMesh does not implement nor enforce any kind of "smart network topology." There was a number of massively-parallel multi-processor super computers built from time to time. All of them more-or-less persuaded a smarter topology for processor inter-connect to maintain low-latency and high bandwidth.

Notable examples are Cray T3D and SiCortex SC5832. T3D had a 3-dimensional "torus" graph topology to make Alpha processors tightly coupled each other. SC5382 had "Kautz graph" topology to inter-connect 6-core MIPS64 processor with the help with built-in DMA engine to talk with L2 caches and I/O devices.

In TILE architecture on-chip inter-connect is software defined. Switch processor can program the network topology to adapt varying demands. In this way, TILE architecture can maintain the flexibility and the scalability in parallel. It's unlikely "topology optimized" super computers can achieve both natures in balance.

iMesh API is provided to make finer control over on-chip network. Cores can be partitioned into groups which work parallel as if they are islands. This feature is implemented by switch network programmability

With help by "topology-aware" and "cache attribute aware" TLB entries, iMesh acts a central role for cache coherency.

2.7. Cache design and feature

Each core has 32KB iCache, 32KB dCache and 256KB i/d combined L2 cache. Either of L1 cache has VIPT "Virtual Index and Physical Tag" nature.

L1 iCache	32KB, 2 way associative, 64B line size.
L1 dCache	32KB, 2 way associative, 64B line size, write-through.
L2 cache	256KB, i/d combined. 8 way associative, 64B line size, write-back.

Table 2-4 cache characteristics

Some TILE processor literatures mention to "coherent L3 cache." It's somehow imprecise. The L3 functionality is achieved by a group of L2 cache. The scheme is called "cache homing." Let us start the explanation.

TILE L1 cache is inclusive to L2. L1 holds subset of L2 contents at any moment. L2 miss happens when offending cache line data is not found in L2. Core asks about the missing cache line data to "neighboring cores" which are grouped by HV for a single OS instance. If found there, cache line data is transferred to requesters L2 cache.

Foreign L2 caches work as an extension of local L2 cache. In other words, a group of cores share their L2 cache contents each other. This scheme is named "cache homing" and Tilera calls the group of L2 cache as "coherent L3" cache. 36 core Gx processor has "9MB coherent L3" = 36x 256KB L2.

Cache lines can be populated sparsely among different L2 to improve the cache efficiency. L3 cache homing is one page attributes. It's controllable by per-page basis.

3. TILE-Gx on-chip devices

integrated multiple DDR memory controller

2 controllers in 36 /16 core models, 1 in 9 core model. TILEPro, the successor of TILE-Gx, 64 core model had four DDR2 memory controller on chip. With dual controller configuration, memory can be driven in interleaved fashion.

mPIPE packet classifier

It's a programmable intelligent packet engine. It offers "frame parse" function to run "sieve-to-forward" classification on incoming Ethernet frame stream at line speed. mPIPE is tightly integrated with GbE / 10G Ethernet network interface.

- 4x 10G ports are available in 36 core model.
- Each port can be reprogrammed to host 4x GbE network interface.
- GbE-only ports are also available in 16 / 9 core model.

mPIPE has local buffer memory to handle incoming and outgoing Ethernet frames. mPIPE can perform load balancing to distribute ingress frames to cores.

Core binds mPIPE device register set to a particular virtual address with a designated dTLB entry for control. mPIPE in turn holds an I/O TLB entry to access data which resides in target (~accelerating application or guest OS) address space so that it can understand VA->PA translation for frame data and accompanying descriptors.

mPIPE has its own 32bit instruction set. A special GCC toolchain is provided to program it.

– two or three operand instruction.
– 32x 32bit register file; 22 of them are general purpose.
– Private SPR registers with mfspr and mtspr to use.

MiCA crypto and compression engine

It's a standalone computing processor populated inside TILE-Gx. Multiple MiCA processors are on a single Gx. MiCA can copy data while encrypting and compressing operation in action. It's a streaming operation.

Core binds MiCA device register set to a particular virtual address with a designated dTLB entry for control. MiCA in turn holds an I/O TLB entry to access data which resides in target address space so that it can understand VA->PA translation for crypto / compression data.

Conventional I/O devices

There are some conventional I/O devices like PCIe, USB2.0 and I2C/SPI in our porting target computer.

PCIe controller works in either root-complex (host) or end-point (device) mode. USB2.0 is used for multiple purpose. It works as virtual console while in development and debug. It can also inject a binary image to Gx processor to run. The binary image consists of boot programs, HV image and guest OS in predefined format.

4. Tilera Hypervisor

HV utilizes TILE protection level feature. Guest OS has heavily limited access to SPR registers. Only handle number of SPR registers allowed to used by Guest OS.

HV is populated at the 1MB area in the upper 2TB space with a hardwired TLB entries.

HV has great control over the entire TILE processor complex. HV makes cores into groups which are manged in M x N rectangle shape to form OS instance.

HV assigns I/O devices to particular instances with I/O TLB entries. Tilera calls the scheme MMIO “memory mapped IO” scheme while SPARC names it “IOMMU.”

HV allows several guest OS'es to run simultaneously. Device and core grouping is defined a HV configuration at the machine startup. Because of it, HV is yet to be improved as flexible as what Xen can do in these days.

Two serial ports are provided in Gx processor. HV can dynamically bind one of serial ports to running OS instance as it console.

BME “bare metal environment”

BME is an API to build “light weight monitor” which runs designated core(s) run special purpose “driver” for data-plane processing. In general any BME program needs accompanying fully-featured OS, like Linux, as a control plane to manage the whole software complex.

iMesh messaging facility API is used by control OS to communicate with BME programs which run on separate core(s).

Several code examples are provided by Tilera;
– one TILE core runs “encryption server” on BME while Linux as “client” which receives the results from BME. In this example data transfer is done in a share page with help of UDN messaging between two.
– A number of Linux process get private cores to run and communicate each other with UDN messaging and shared pages.

5. NetBSD/tile

This port is based on NetBSD 6.0 STABLE code set. It's a 64bit SMP kernel and 64bit userland. The kernel runs as a guest OS conjunction with Tilera Hypervisor.

NetBSD/tile uses GCC 4.6.3 ported by Tilera. We have been using it as it is. As GCC 4.5 is still in use in NetBSD 6.0 code set, we integrated GCC 4.6.3 to start.

64bit pmap was implemented from scratch. It's modeled after Alpha pmap. Although TILE-Gx offers 13 different page sizes, HV employs much humble page size selection. We chose 64KB page for NetBSD/tile as it is parallel to Tilera Linux VM implementation. The virtual address partitioning is “10 + 8 + 8 + 16.”

NetBSD/tile utilizes SMP ready NetBSD6 kernel internal as large as possible. NetBSD5 introduced much sophisticated kernel constructs and API sets which are effective and useful for scalable SMP OS. Since then gradual streamlining has been done for fore-running SMP

NetBSD ports. Now NetBSD6 is a mature platform to make a jump start for fresh SMP porting.

The following is the typical set of useful SMP API;
- atomic_ops(3)
- kcpuset(9)
- xcall(9)

The first group must be implemented in early kernel porting stage. In most cases they have to be written with assembler code to be best suited for particular processor nature. The latter two are pure software construct written in plain C code.

Parallel programming model is NetBSD pthread. NetBSD pthread is well organized to adapt various processors with minimum effort. We did not make particular modification for TILE-Gx support. It works just like as any other pthread implementations like one in Tilera Linux.

Very limited number of assembler files were written so far. One one for kernel; it's "locore.S" The file contains 4 well define major routines;
- CPU startup for primary core and secondary cores.
- Exception entry / dispatch / return
- CPU context switch
- fast software interrupt dispatch / return

Other assembler files are for libraries and a few application program like rtld(1). The following is the list of major TILE-Gx dependency in concern.
- src/common/lib/libc/arch/
- src/lib/libc/arch/
- src/libexec/rtld/arch/

5.1. Key design decisions

In this section we describe concisely about some design decisions to make a port realized.
- struct trapframe, struct switchframe and struct pcb.
- UPACE to hold kernel stack and struct pcb.
- pmap(9) to interface processor with NetBSD VM.
- Exception and interrupt handling to comply target processor design intent.
- IPI "inter-processor interrupt" which is essential to make SMP possible.

struct trapframe is a snapshot image of runtime context. One trapframe is always created at the high end address of USPACE. Actual kernel stack starts just below of it to grow downward. The reserved trapframe area is for user process context. Whenever user process gets interrupted by exception or device notification, the trapframe is to record the user context to resume later. This area is also used for system call. While in kernel mode, kernel gets interrupted by the same reasons as user mode process does. At the occasion, trapframe is created and pushed on kernel stack.

TILE architecture has 64x register file. 8 out of them are not a part of process context and to be excluded. We chose 64x 64bit = 512B size anyway for struct trapframe. In vacant fields we place some extra contexts for process to retain. They are exception return address, status register value at the time when exception happened, offending exception type and a value of a certain SPR, "CmpValue" indeed, for cmpexch instruction.

struct switchframe is for CPU context switch. NetBSD defines two context switch routines. cpu_switchto(9) and lwp_return(9) are the routine to perform context switch. TILE architecture has a large set of caller-saved register. Our switchframe is 25x 8B = 200B in size.

struct pcb is one of longest surviver among UNIX kernel primitives. It got smaller than used to be since the way how to run context switch made smarter. Our struct pcb is as small as just to hold struct switchframe and a bit extra.

USPACE size is 64KB as aligned with NetBSD/tile page size.

5.2. ASID management

ASID management is modeled after the way used for NetBSD/alpha and NetBSD/mips. In this section we explain it in larger degree.

Kernel has a variable for "ASID generation number" to make sure a unique ASID assigned for running process in processor. It's a central idea. Our ASID management algorithm works in this way.
- pmap_activate(9), one of NetBSD kernel API, switches processor's current ASID value whenever a new process is ready to take control.
- Switching current ASID is a light weight operation for OS as it eliminates the necessity of TLB flush at every context switch. ASID-less processors need to perform the whole scale TLB invalidation to discards all entries at every context switch. As TLB works as a cache for VM address translation, TLB flush hearts severely TLB hit ratio spoiling VM performance. ASID-aware processors just need to switch current ASID value. Changing processor current ASID can be considered to switch imaginary TLB store which exists for each ASID value.
- Every new born process has no ASID assigned. pmap_activate() chooses new one which is never allocated so far and assign it with the process. pmap_activate() also records the current ASID generation number in the process's pmap store.
- ASID is a small number to count only up to 255. If

pmap_activate() finds the 8bit gets exhausted, then it bumps ASID generation number in a kernel variable by 1 and chooses a new ASID wrapped to the least available number (normally 1 as ASID 0 is reserved for NetBSD kernel pmap). On this occasion, kernel makes full scale TLB invalidation to discard all TLB entries.
– Whenever pmap_activate() is about to switch current ASID, it checks ASID generation number in kernel variable matches the process's generation number recorded at ASID creation. If they differ, it means the process's ASID is no longer valid. pmap_activate() selects and assigns a fresh ASID for the process to run recording current ASID generation number too.

Given any moment every running process has its own unique ASID. The generation number scheme reduces the necessity of full scale TLB invalidation in great degree. TLB flush only happens when ASID range gets run out and ASID generation number is to be bumped.

5.3. TLB shootdown

TLB shootdown is the essential operation in any SMP kernel. Like as processor cache, TLB is a local resource to processor core. The way to invalidate local cache or local TLB is provided by a certain mechanism. In general invalidating remote TLB is as hard to archive as invalidating remote cache.

In SMP system, TLB invalidate operation must be propagated to multiple cores which have been running a particular process. Process's pmap must maintain a "processor set" to track which cores have run it. Here goes the explanation of remote TLB shootdown by ASID bump;

When pmap(9) detects the necessity to invalidate one or more TLB entry of particular process, kernel needs to run invalidate operation both for;
– the "local" core which happens to run the kernel on behalf of pmap() at the very moment, and
– all of "remote" cores which the process's pmap() are aware of.

The latter operation is named "TLB shootdown." It's implemented with IPI. It triggers a remote core action by inter-core message. TLB shootdown logic can be built in with help of xcall(9) "cross call" kernel API.

A smart ASID management can achieve remote TLB invalidation with a small cost.
– mark ASID in offending process's pmap() store "unassigned."
– broadcast a xcall(9) message to remote cores triggering IPI.
– When one of cores is about to run the process in the next scheduling, pmap_activate() will choose and assign a fresh ASID the offending process. The stale TLB entry with abandoned ASID gets invalidated at once.

5.4. Useful SMP facilities in NetBSD6

SMP NetBSD kernel provides the way to manage CPU in finer gain. There are less known set of useful commands. Let us mention about them in brief.

cpuctl(8) ... try "/usr/sbin/cpuctl list" on your modern Intel computers. It shows the list of CPU state which tells online / offline.

prset(8) ... try "/usr/sbin/prset -p" on your modern Intel computers. It can create arbitrary number of "processor set" which is bound with any process. CPU affinity is made possible by processor set binding. Would be possible to bind a processor set with a kthread (kernel thread) which runs specific kernel subsystem like GbE and/or disk drivers.

schedctl(8) ... try "/usr/sbin/schedctl -p 1" on your modern Intel computers.
– It assigns one of predefine scheduling policy to a process. It replaces nice(1) and renice(8) priority control commands.
– Three difference scheduling policies provided by NetBSD so far.
– Time-sharing which follows the tradition UNIX semantic used for long time.
– First-in, First-out
– Round-robin

5.5. Future development

This project is active. Here we try to make a summary about missing functionalities and future development in some arbitrary order.

Soon to use TILE-Gx native FP instructions. Currently the entire NetBSD including userland is made with "–DSOFTFLOAT" compile option.

Drivers for some conventional PCIe devices like SATA and/or 100M Ethernet NIC. Currently whole system code image is injected with USB debugging facility to run NFS diskless configuration.

iMesh communication API for NetBSD. It remain under research. For now there is no provision to utilize iMesh programming.

MiCA integration with a proper API. NetBSD kernel has pcu(9) "per-CPU-unit" framework. It's for the encapsulation of CPU's hardware context to save / restore. It handsomely covers the cases beyond the general purpose register. The typical usage of pcu() is to manipulate FPU register set. We're considering whether pcu() can integrate multiple MiCA units to NetBSD kernel in sane manner.

NetBSD/xen allows dynamic attach / detach maneuvre while kernel is up-running. It allows core to attach / detach dynamically and allows block device attach / detach dynamically. We assume it'd be some difficult to implement similar functionality in TILE, however, it'd worth persuading the way to make them possible.

We're aware of Tilera HV has no provision to startup & tear down "targeted" core while up-running. HV source code is disclosed as a part of Tilera MDE development package. It's said that HV can be extended for customer's own needs.

LLVM transition from GCC4 is recognized mandatory as it would exploit the potential of TILE VLIW nature.

6. NetBSD TILE-Gx applications

We focus on compute-intensity markets. We're considering to engaged in SDN, VLDB search engine and desktop HPC.

SDN "Software Defined Network"

It's the third wave of virtualization technology; server virtualization, storage virtualization and then network virtualization. Industry trends predict that routers and firewall will vanish soon while they are morphing into big smart switches.

Bangalore team is now exploiting super fast frame forwarding algorithms. They are generalized for "search-and-lookup" computational complexity reduction problem. Problem statements are now being defined. The implementation of algorithms must be robust enough to handle incoming frame stream as fast as arriving in wire-speed rate. They must also be robust enough combination explosions of matching rules.

VLDB search engine

In these days Very Large scale DB are directly connected with Internet. It's working in real time manner. The typical case is SNS like FaceBook. "mem-caching" is now a common tactic to implement super fast search engine. We recognize many-core processor and GPGPU are now gathering industry attention as they would be good vehicle in engineering sense for VLDB search engine.

Desktop HPC "High Performance Computing"

It's a kind of human being's forever desire to own super computer at hand. TILE-Gx can be a handy basis of many-core 64bit general purpose computer. It's said that the next generation of Gx can be extended by wiring multiple processor with InterLaken inter-connect. Today a pair of Tesla GPGPU 16x lane PCIe cards can archive Tflops grade computing power. Then, how about making the twenty first century incarnation of desktop personal computer, let's say, whose outlook are just like as SGI Indigo or NextCube?

7. Conclusion

Poring NetBSD 6.0 to TILE-Gx is found easier than anticipated since NetBSD 6.0 provides SMP ready kernel constructs and API sets to use. The number of lines written in assembler was very small as the essential part of porting burden are well defined. VLIW nature of the processor is recognized not a hurdle.

Acknowledgment

Sanctum Networks wishes to express its gratitude to all members involved in this project, especially the members from Japan who contributed critically in the early stages.