Abstract

* Cell broadband engine is a heterogeneous multi-core processor which consists of a Power PC element, (PPE) and Synergistic Processor Elements (SPEs). SPEs can be used to achieve better performance. This paper proposes utilizing SPEs from user space to accelerate kernel services. Our solution allows kernel services to access to SPEs easily. We'll show evaluation of the concept, using modified compressed loop device driver, CLOOP, to utilize SPE. We'll also discuss possible other kernel services to be accelerated by SPEs.

* Machida - Sony
* Shinohara - Sony
* Tsukamoto - SCE
* Suzaki - AIST

Cell broadband engine, SPE assisted user space device driver

2007.04.19

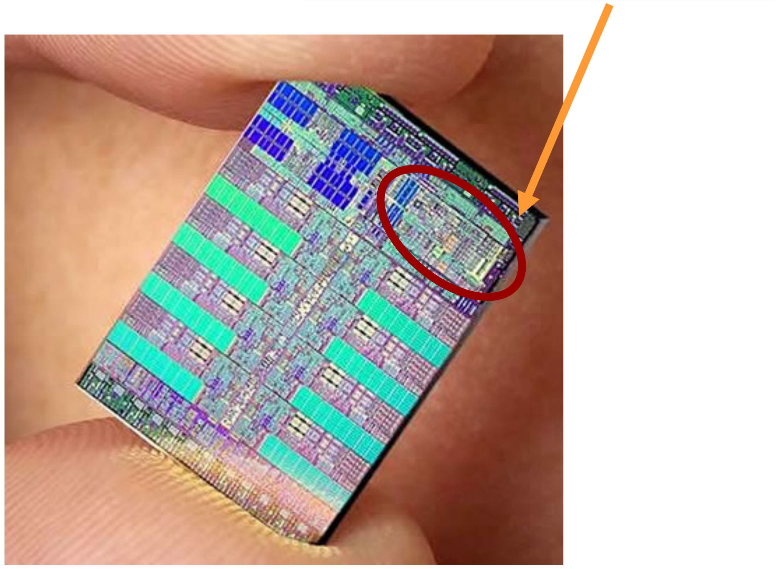
Hiroyuki Machida

What’s our intention and backgrounds?

Motivation

* Accelerate performance of commonly used functions, e.g. kernel services and device drivers, by utilizing SPEs in Cell BE.

PPE Logic is only 1/10 of die size



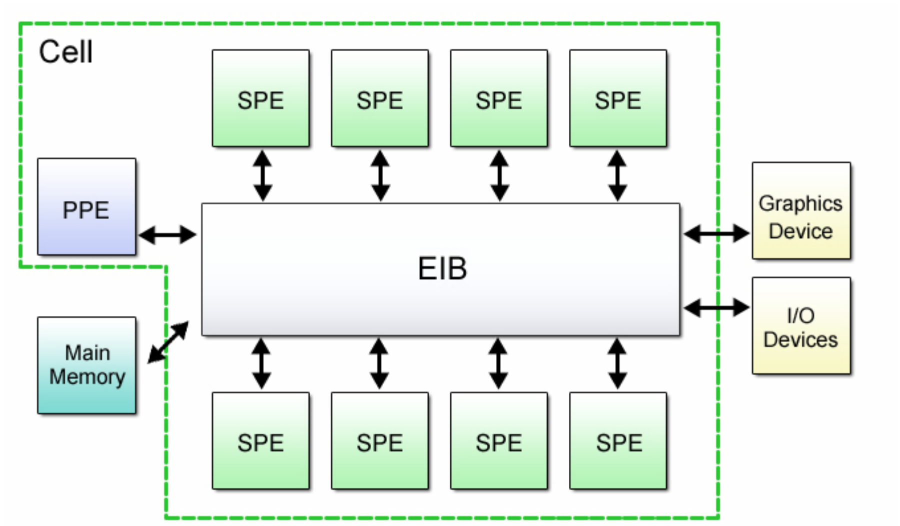
Cell employs following strategies
- "simpler structure and higher clock"
- "more room for SPEs on silicon"

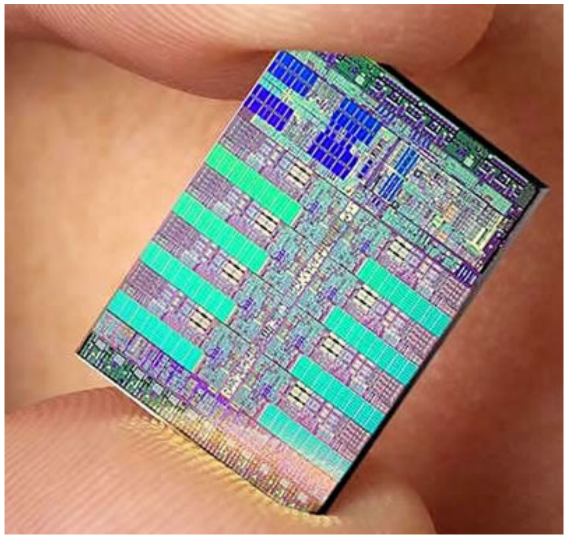
Cell consist multiple cores
-One Hyper threaded PPE
-Eight SPE

Using only PPE does not achieve same performance with same clock on G5(PPC970)

What Is Cell BE?

- 1 PPE + 8 SPE connected with EIB
  - PPE - Power Processor Element
    - PPC64+VMX insns/SMT/in-order/deep pipeline
  - SPE - Synergetic Processor Element
    - general purpose SIMD with Local Storage (LS), not cache
  - EIB - Element Interconnect Bus
    - Hi-speed coherent SMP Bus





Again, What Is Cell BE?

From "Tapping the Power of Cell", by Mike Acton

* The Cell is not a magic bullet.

* The Cell is not a radical change in high-performance design.

* The Cell is fun to program for.

- Do not hesitate to try programming Cell.
  - "Most good solutions for the Cell will be good solutions on other platforms."
  - As same as modern computer programming, "DATA IS MORE IMPORTANT THAN CODE.".
- Difference is just...
  - "Where data and code optimization are ***merely important*** on conventional architectures, it's ***critical*** on the Cell."

Again, Motivation

* Even leading Cell BE Linux Project, I was not so much familiar with Cell Programming.
* Frequently, I had to ask others about Cell BE Program Logic.

* OK, It's just time to try by myself.

* Actually, I'm personally VERY interested in Cell Programming now. ☺

Starting Investigation
Offload Kernel to SPE

Requirements

* Preserve existing Kernel-User APIs
* Changes should be minimized
* Utilize existing functions as much as possible
* Improve performance
  - Less CPU(PPE) usage
  - Faster in execution

Constraints

* SPE itself doesn’t have privilege mode on execution.
  - CPU core of SPE doesn’t have “privilege” concept.
  - However, MFC has capability to switch kernel mode and user mode address space of PPE side main memory。

* Current Kernel doesn’t support executing heterogeneous CPU core instructions.

Possible Two Solutions

Offloading kernel functions on which address space

* Kernel Mode

- Adding new infrastructures in kernel to support SPE acceleration in kernel mode.

* User Mode

- Adding helper feature inside kernel to allow off loading function in user space for SPE.

Pros v.s. Cons

- Kernel Mode
  - Pros
    - Less overhead
  - Cons
    - It's difficult to debug with this model
      - kgdb do not speak SPEs
    - New code required for controlling SPE in kernel
- User Mode
  - Pros
    - No new code required for controlling SPE in kernel
    - This allows programmers to use existing tools for debugging
  - Cons
    - Overhead switching between kernel and user space
    - Require protecting user space SPE data/prog from other regular user space application

Feasibility Study with User mode

* Try out a simple example using CLOOP, software device driver.

* Why CLOOP ?
  - It’s small
  - It has locality in memory reference (block decompression)

It won't be a security hole

* MFC access to the main memory (XDR) is bound inside corresponding user process, since MFC is tied to MMU on PPE.
  - SPE can’t reach XDR directly, just to Local Storage (LS) .
  - Data transfer between LS and XDR is taken place by MFC.

* PPE access to SPE registers and LS are virtualized in each user process unit.
  - PPE can map LS and SPE registers, however SPE registers have privillage and those mappings are under controlled by kernel.
  - Kernel prevents to map LS and SPE registers from other user process.
  - If it could, it’s a bug of kernel.

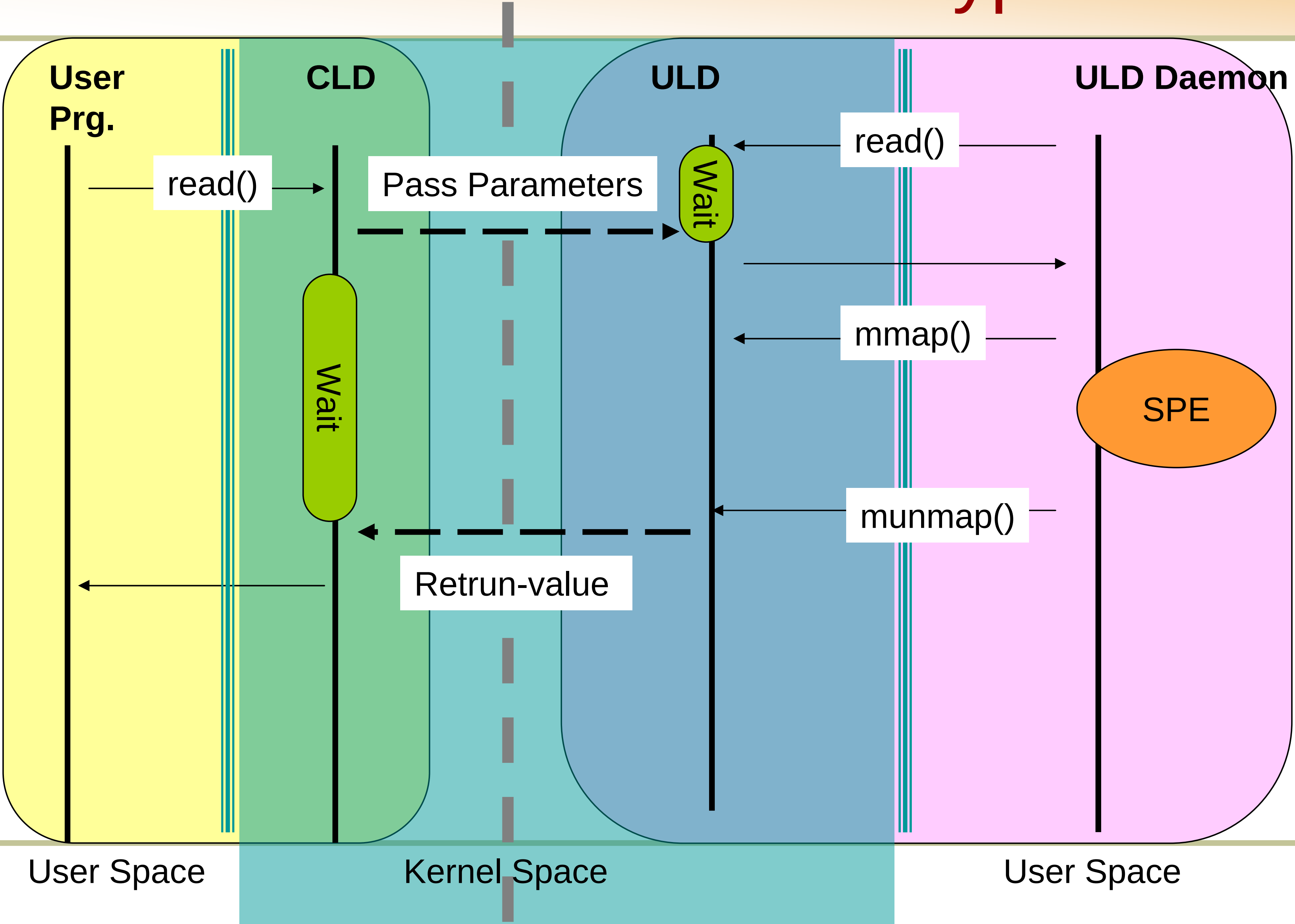
* As a conclusion, SPE running kernel functions are well isolated from the other user processes.

Prototyping –
So, Is it possible to implement?
How does it work?

Terminologies

* CLOOP	Compressed Loop block device
* UIO	Userspace I/O kernel drivers
* CLD	CLOOP Driver
* ULD	User Level Device Driver

Overview of our Prototype



Basic Design

* User space SPE offloading daemon waits a request from kernel.

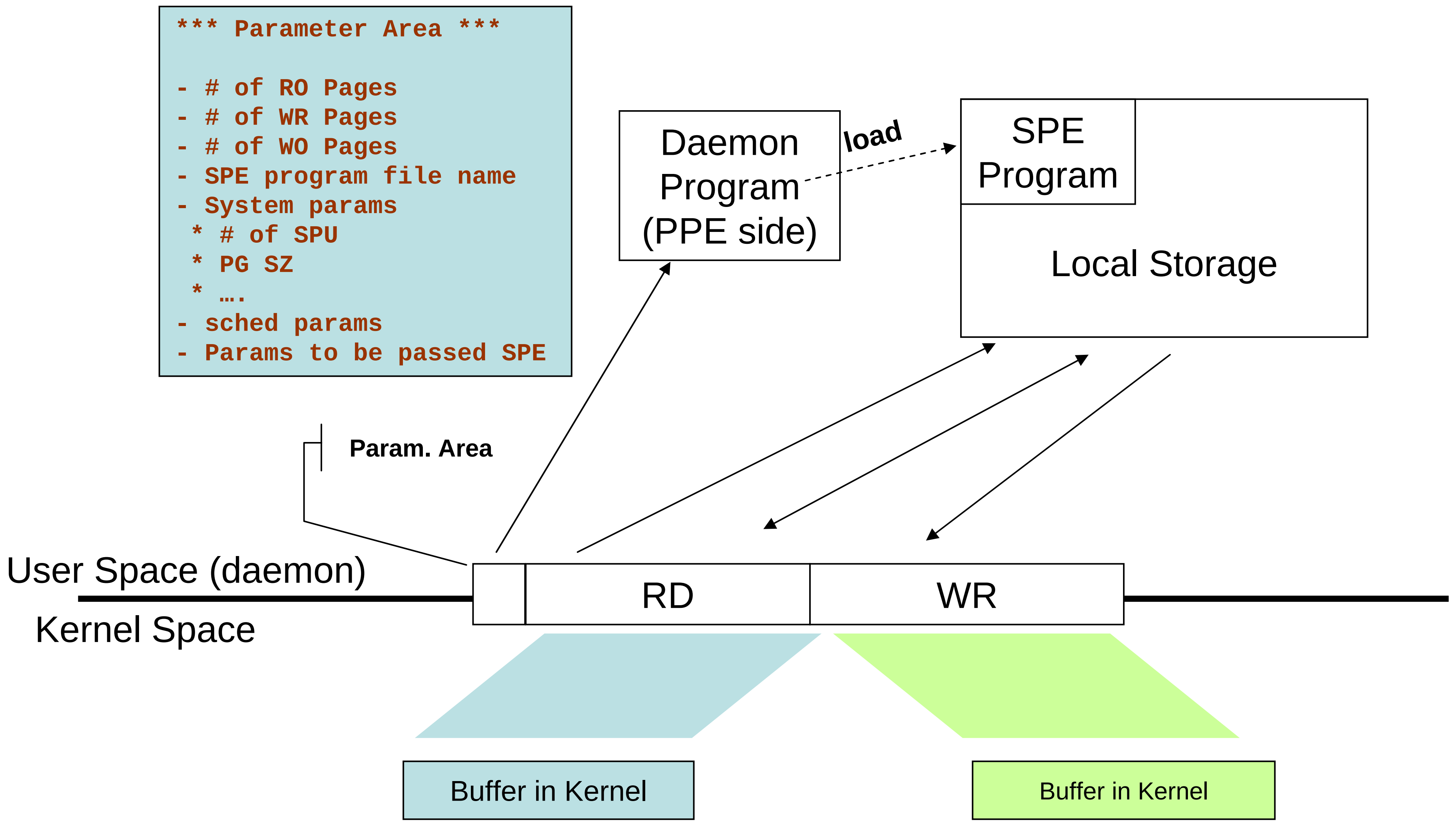
* Driver locks pages, wake up the daemon and pass the pages to the daemon.

* The passed page includes file name of SPE codes, parameters and input data/out put data.

* The daemon start SPE and has been blocked until SPE execution, according the parameters passed from kernel.
  - SPE would transfer data to/from the locked pages.

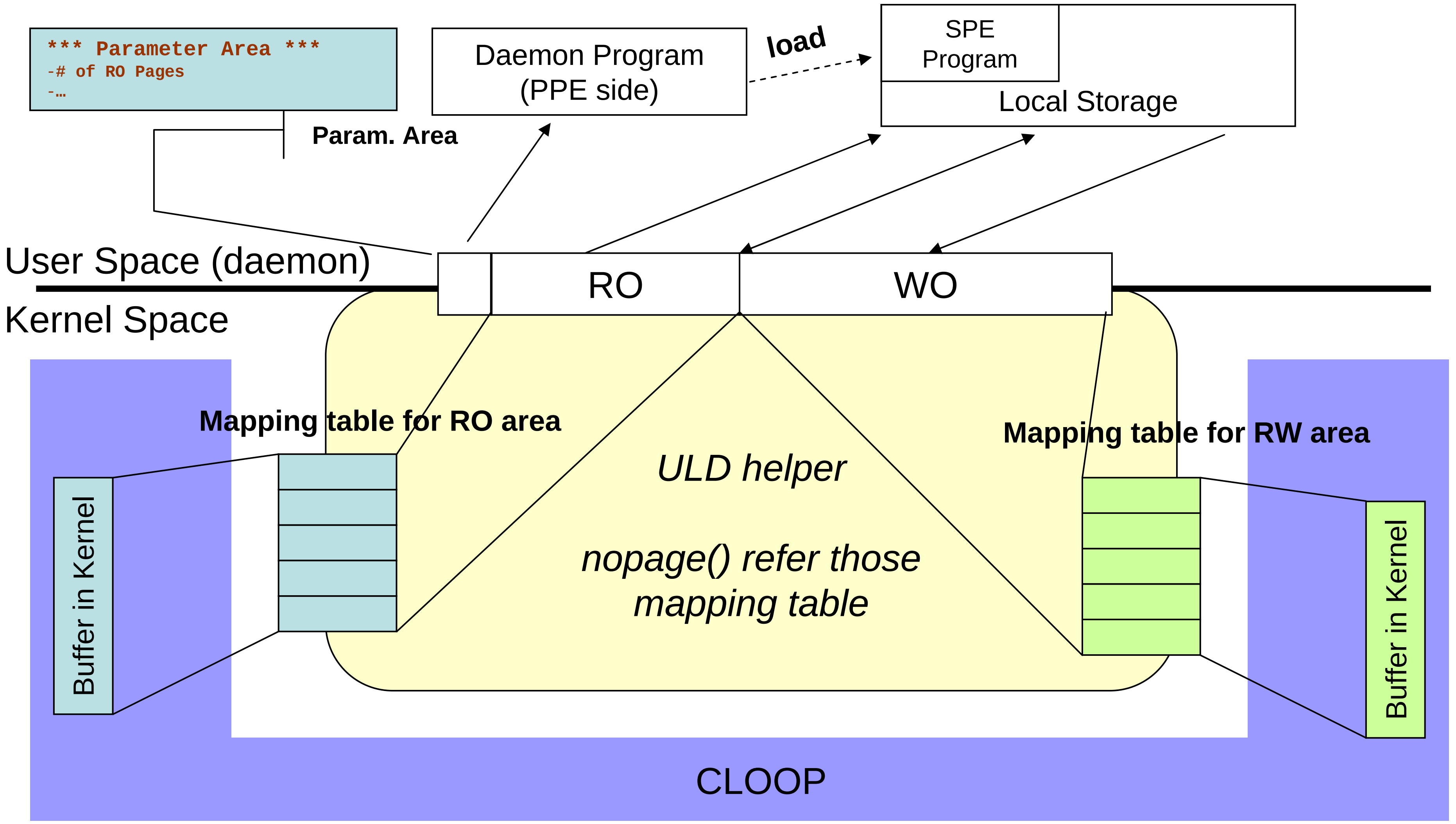
* After SPE execution finished, daemon will inform it to kernel.

Data flows in ULD daemon

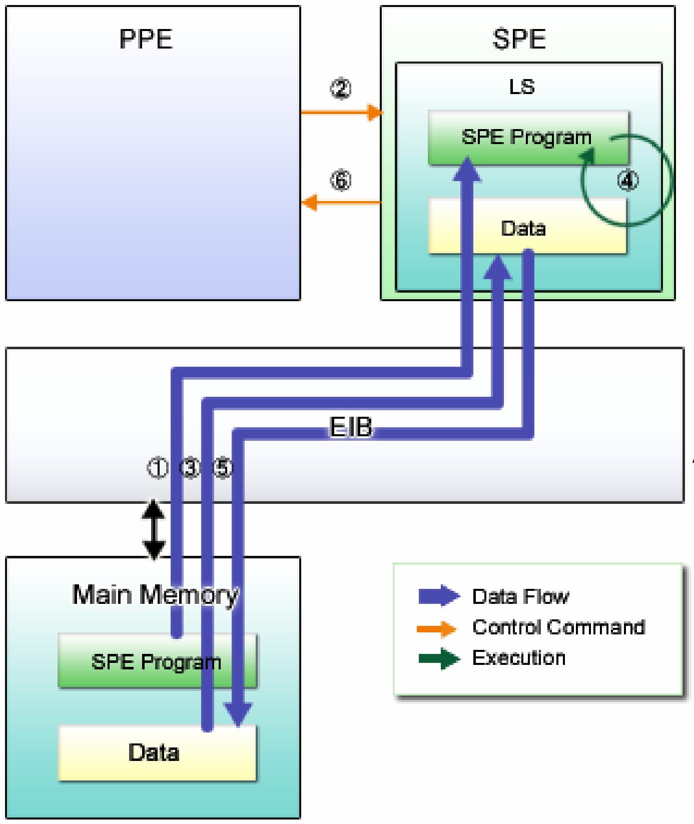


```
*** Parameter Area ***

- # of RO Pages
- # of WR Pages
- # of WO Pages
- SPE program file name
- System params
  * # of SPU
  * PG SZ
  * ….
- sched params
- Params to be passed SPE
```

mmap() and ULD helper



Typical SPE program flow



- PPE Program side
  - spe_create_context()
    - Create SPE context
  - spe_image_open()
    - Open elf file of SPE program
  - spe_program_load()
    - [1] Load SPE program to LS
  - spe_context_run()
    - [2] Let SPE start the program

- SPE Program side.
  - [3] Transfer data to be process into LS from PPE side main memory, though by MFC.
  - [4] Processi data in LS
  - [5] Transfer processed data from LS to PPE side main memory though MFC.
  - [6] Signal PPE program that SPE program has sopped.

ULD Daemon code fragments

```
*   User space daemon, which actually do decompress.

*   spe_create_context()
*   spe_program_load();
*   pthread_create();
      -   spe_context_run()

*   fd = open();                      // ULD helper device
*   while (1)
      -   read(fd, &in, sizeof(int));   // wait a request
      -   vaddr = mmap(fd, …);          // map Req Params and buffer
      -   // making page faults to mmap()ed area
      -   // setup parameters on mapped area.
      -   spe_in_mbox_write();          // start SPE
      -   spe_out_mbox_read();          // wait answer from SPE
      -   munmap()                      // notify termination to ULD helper
```

CLD side code fragments

* Modify compressed_loop.c
  - Replace calling **`decompress()`** with
    * // acquire a request packet
    * // fill parameters and setup mapping tables
    * // send the request to ULD helper device
    * // wait for completion
    * // got a return value
    * // free mapping tables
    * // free the request packet

ULD helper, based on UIO

- What's UIO (Userspace IO devices) ?
  - See - http://www.kroah.com/log/linux/uio.html
    - drivers/uio/uio.c
    - include/linux/uio_driver.h
    - Documents/DocBook/uio-howto.tmpl

- Kernel side helper driver, *uio_spe.ko,* has been easily implemented, using UIO.
  - *uio_spe.ko* is inheriting *uio.ko*
    - Start with "gregkh-01-driver-2.6.21-rc1-git2.patch"
    - Added just few methods and attributes.
    - ***debug UIO….***

DeCompression

- * Evaluation with miniLZO
  - Alternative of GnuUnZip - Decompress() in CLOOP
  - http://www.oberhumer.com/opensource/lzo/
- * Why miniLZO?
  - Small foot print
  - *Algorithm Looks Simpler*
- * My changes to the original miniLZO.202
  - Change buffer size to 64KB, so that LS can hold entirely.
  - Some introduction of SIMD-lize.
    - * Byte copy -> inline SIMD memcpy/memove
    - * Find non-zero byte. -> inline modified strlen

Very Quick Evaluation

Evaluation Environment – PS3

* Kernel - 2.6.21-rc6 PS3 git tree
  - git://git.kernel.org/pub/scm/linux/kernel/git/geoff/ps3-linux.git
* PPC64 FC5
  - gcc-4.1.1-1.fc5
* PS3 Linux Distributor Starter's Kit v1.1
  - libspe2-2.0.1-be0647.3.20061130.1.ps3pf
  - spu-newlib-1.14.0.200612070000-1.ps3pf
* gcc/binutils from IBM Cell SDK 2.0
  - ppu/spu-gcc-3.3-72
  - ppu/spu-binutils-3.3-72
* SPE GCC Flags for Optimization

  -
    ```
    -O3 -mbranch-hints -funroll-all-loops
     -fomit-frame-pointer -ftree-vectorize
     -finline-functions  -ftree-vect-loop-version
     –ftree-loop-optimize
     -fdata-sections -ffunction-sections  -Wl,-gc-
    sections
    ```
* PPE GCC Flags for Optimization
  - `-O3`

Simple test case

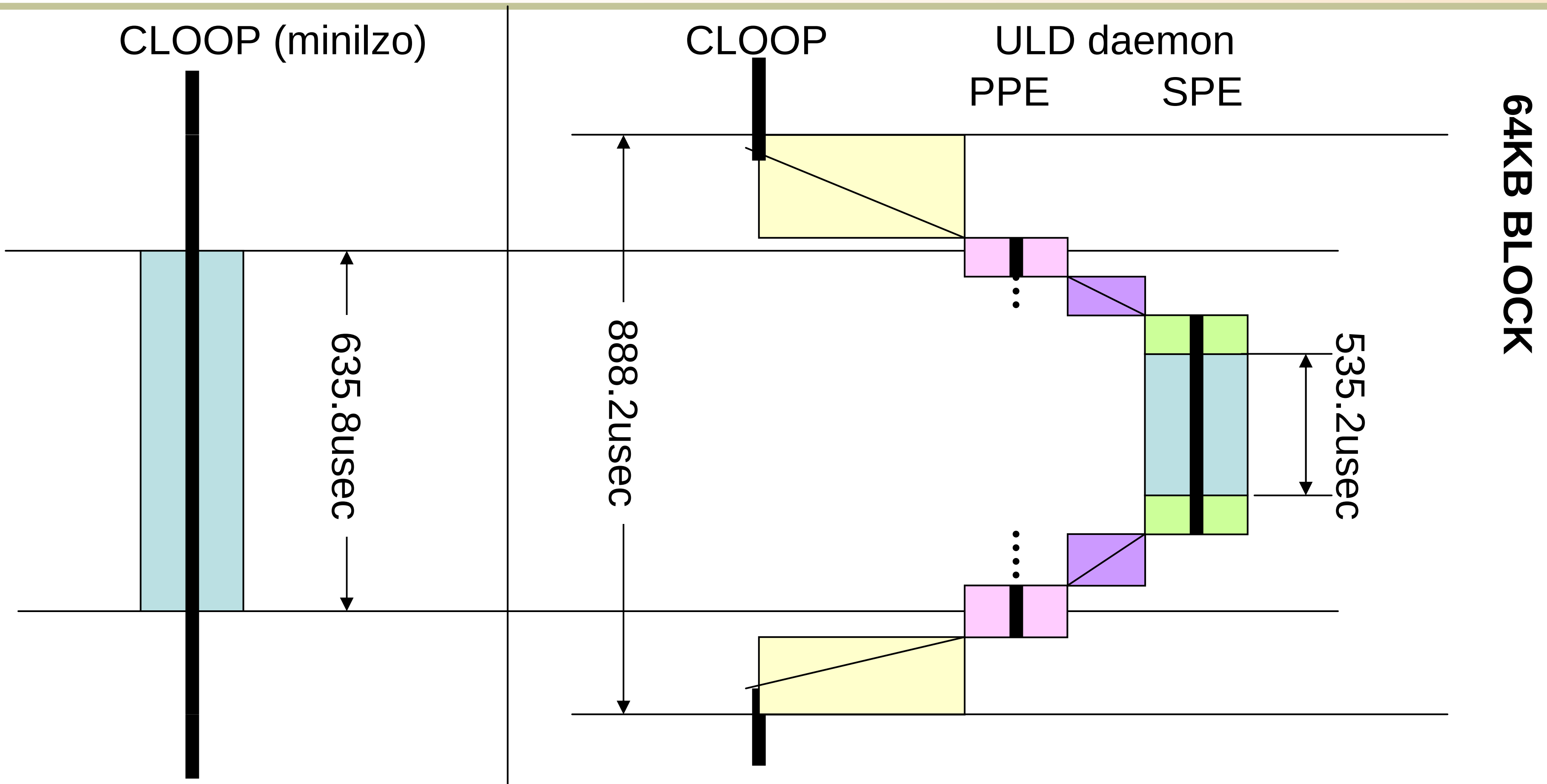
* Insert modules
  - `insmod uio.ko`
  - `insmod spe_uio.ko`
  - `insmod cloop.ko file=cd-image.iso.mini-lzo`
* Repeat 10 times
  - `dd if=/dev/cloop0 of=/dev/null bs=1M`
  - `mount –t iso9660 /dev/cloop0 /mnt/a`
  - `diff –upr /mnt/a original image`
  - `unmount /mnt/a`
* Remove modules
  - `rmmod cloop.ko`
  - `rmmod spe_uio.ko`
  - `rmood uio.ko`

Data for the Evaluation

* ISO image contains
  - RAW
    * 28.35MB
  - miniLZO-ed
    * 17.03MB 60%
    * (GZIP-9 13.04MB)
* Data size is arranged so that all data can reside in main memory.
  - Don’t want to measure HDD I/O speed.
* miniLZO 64KB block

```
$ ls -Rl cd-image
cd-image:
total 12
drwxrwxr-x 2 machida machida 4096 Apr 11 07:54 boot
drwxr-xr-x 2 root    root    4096 Apr 14 05:03 sbin
drwxr-xr-x 3 root    root    4096 Apr 14 04:59 usr

cd-image/boot:
total 28456
-r--r--r-- 1 machida machida  1110582 Dec 21 19:39 initrd.img
-rwxr-xr-x 1 machida machida 27985528 Dec  7 18:19 vmlinux-2.6.16

cd-image/sbin:
total 44
-rwxr-xr-x 1 root root 42468 Feb 24  2006 syslogd

cd-image/usr:
total 4
drwxr-xr-x 3 root root 4096 Apr 14 04:59 share
cd-image/usr/share:
total 4
drwxr-xr-x 3 root root 4096 Apr 14 04:59 doc

cd-image/usr/share/doc:
total 4
drwxr-xr-x 2 root root 4096 Dec 21 18:16 zsh-4.2.5

cd-image/usr/share/doc/zsh-4.2.5:
total 232
-rw-r--r-- 1 root root  1986 Mar 10  2002 BUGS
-rw-r--r-- 1 root root 21687 Jul 23  2003 completion-style-guide
-rw-r--r-- 1 root root 11839 Feb 27  2004 CONTRIBUTORS
-rw-r--r-- 1 root root 82695 Apr  6  2005 FAQ
-rw-r--r-- 1 root root  5005 Jul  3  2004 FEATURES
-rw-r--r-- 1 root root  1477 Mar 14  2004 LICENCE
-rw-r--r-- 1 root root  9560 Jul  3  2004 MACHINES
-rw-r--r-- 1 root root 28518 Jan 12  2005 NEWS
-rw-r--r-- 1 root root  8768 Apr  6  2005 README
-rw-r--r-- 1 root root 30453 Jan  8  2004 zsh-development-guide
-rw-rw-r-- 1 root root  6330 Sep 10  2004 zshprompt.pl
$
```

Where are measured and Results



decompression	635.8use

WITHOUT OFFLOADING

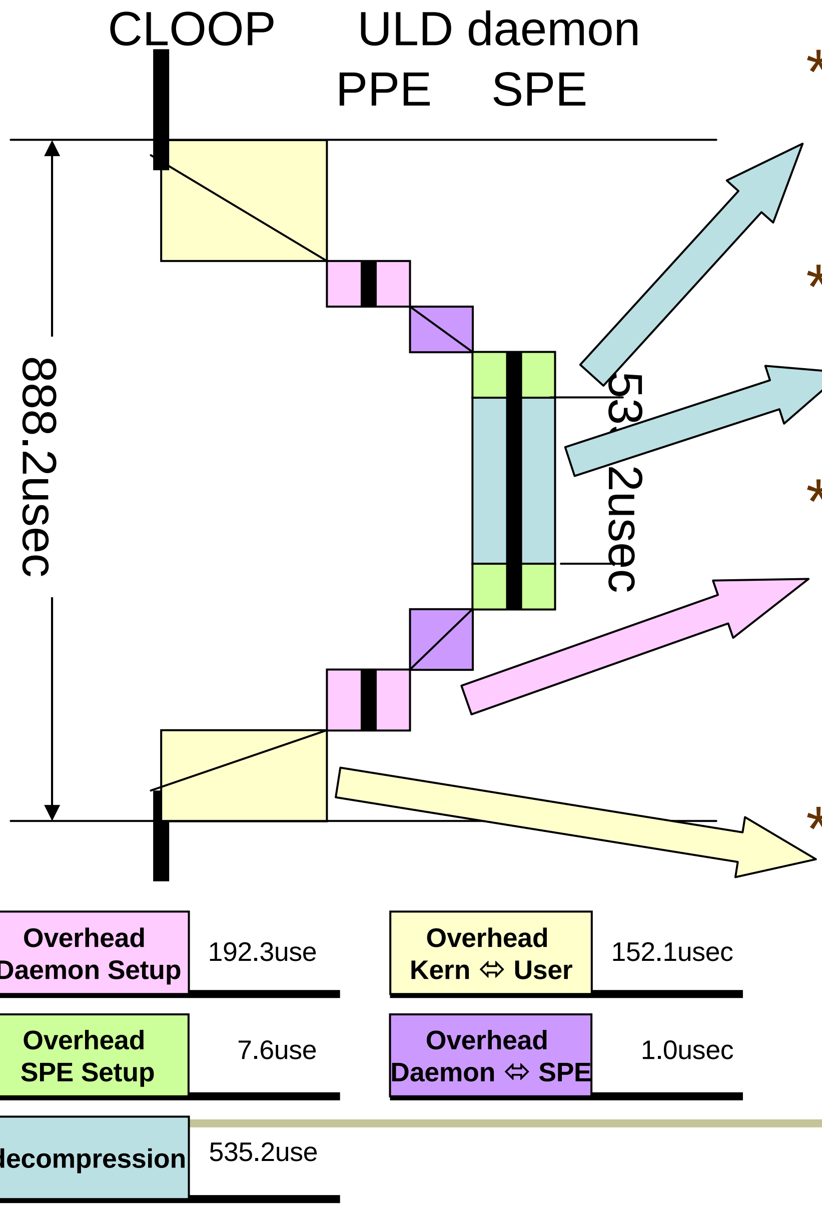
Overhead Daemon Setup	192.3use	Overhead Kern ⇔ User	152.1usec
Overhead SPE Setup	7.6use	Overhead Daemon ⇔ SPE	1.0usec
decompression	535.2use		

WITH OFFLOADING

:-<

“Game Over”

No, We still have chance, …



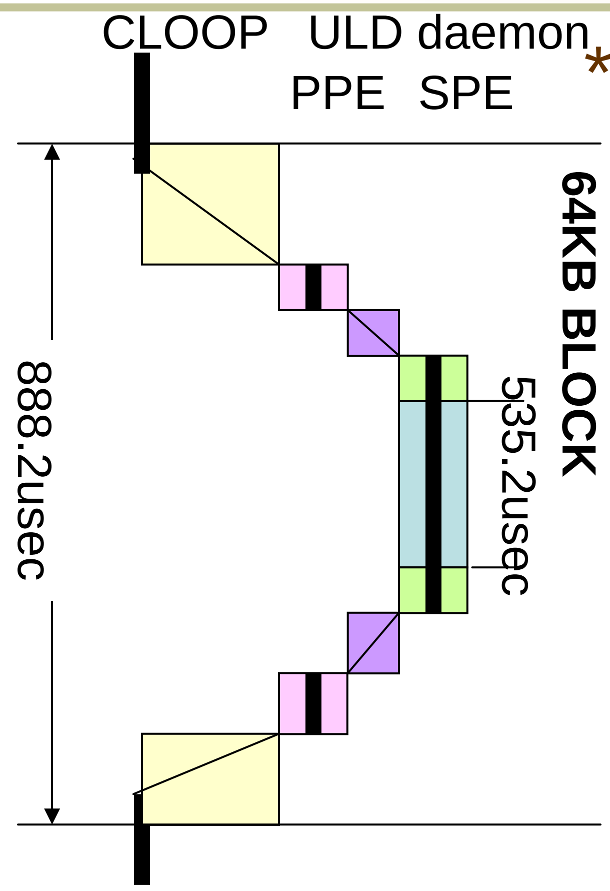
* Just one SPE used, we have 5 more..

* miniLZO is not still well optimized to SPE SIMD instructions.

* On demand PTE installation using page fault and nopage(), would be painful.
  - Install PTEs on mmap()

* Cost of Context Switch would be painful.
  - Due to Large memory address space ? - 64 KB page??
  - Bigger Block Size ?

Quick Estimations



* 512KB block (64KB x 8) decompress
  - Non- Offloading - 5086.08 usec
    * 635.76 usec x 8 = 5086.08 usec
  - Offloading with 4 SPEs
    * Best case - 1423.3 usec (x 3.6 faster)

Overhead Daemon Setup	x1	Overhead Kern ⇔ User	x1
Overhead SPE Setup	x1	Overhead Daemon ⇔ SPE	x1
decompression	x2		

    * Worst case – 3834.03 usec(x 1.3 faster)

Overhead Daemon Setup	x8	Overhead Kern ⇔ User	x8
Overhead SPE Setup	x1	Overhead Daemon ⇔ SPE	x1
decompression	x2		

Overhead Daemon Setup	192.3use	Overhead Kern ⇔ User	152.1usec
Overhead SPE Setup	7.6use	Overhead Daemon ⇔ SPE	1.0usec
decompression	535.2use		

Next Steps

- Feedback to UIO driver
- Clean up codes
- Make it fast
  - Install PTEs on mmap()
  - Try 64KB PAGE SIZE
  - Try 4 SPEs and 512KB block
- Make it general
  - Make ULD and ULD helper to be generalized
- New possible issues
  - SPE scheduling
    - ***Gang*** scheduling might be helpful
- Other Candidate of User Space SPE Device Drivers
  - Discuss applications on other device drivers and additional requirements on helper functions.
  - Crypto API -> OCF looks very good candidate？
  - VFB/color space converter
  - USB web cam ?? It could be user space driver even now.
    - Decompression/ Color space converter

Questions ?

Comparison to PC

* **These results are very preliminary, not accurate.**

* **DD with miniLZO version of CLOOP**

  - `dd if=/dev/cloop0 of=/dev/null bs=1M`

  - Core Duo 1.3GHz 90-100MB/sec(wo statistics code)
    * -O3 –msse3 –funroll-loops

  - PPE Cell 3.2GHz 70-90MB/sec(w statistic code)