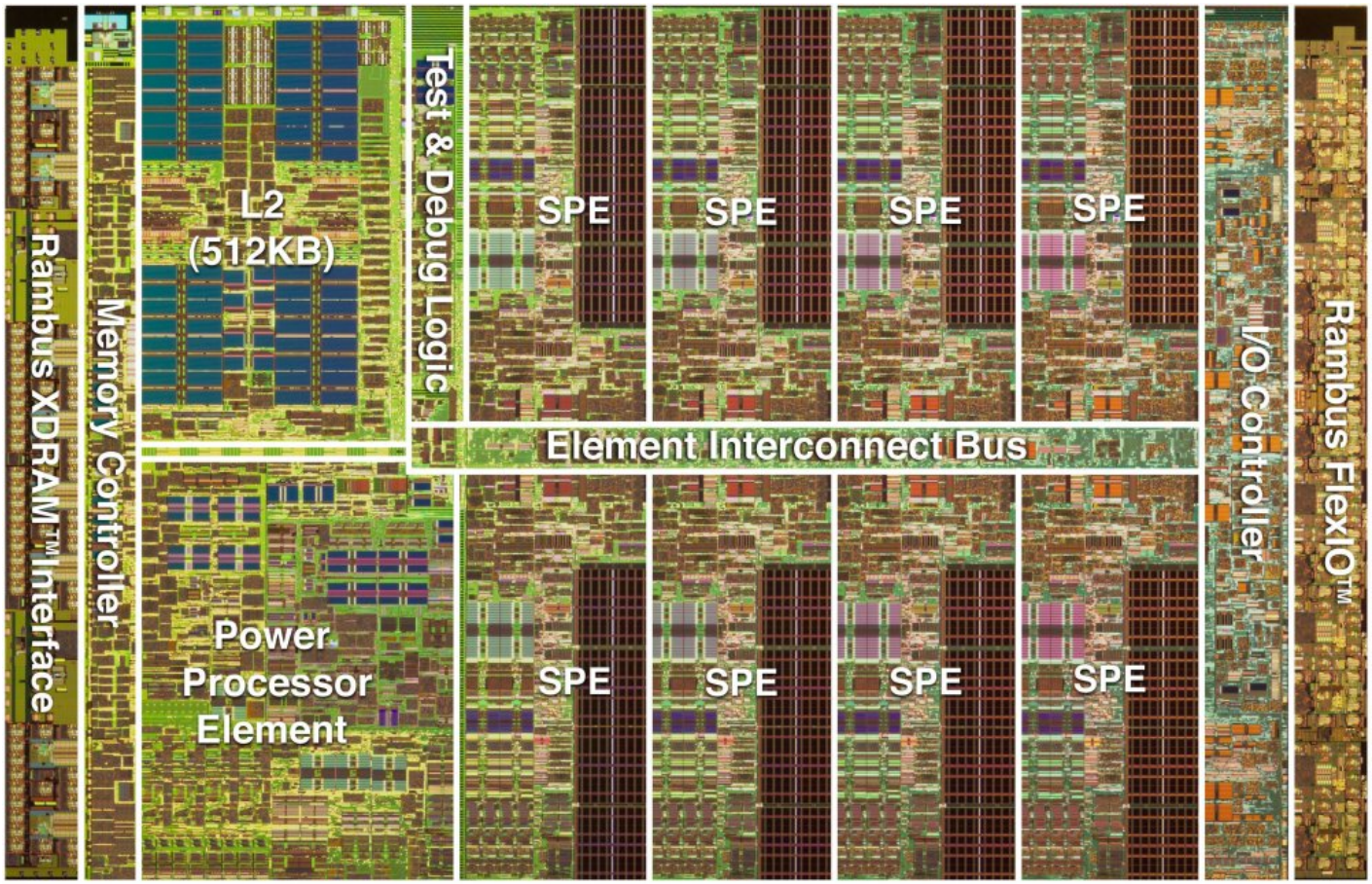
Linux on the Cell Broadband Engine

Arnd Bergmann <arnd@arndb.de>

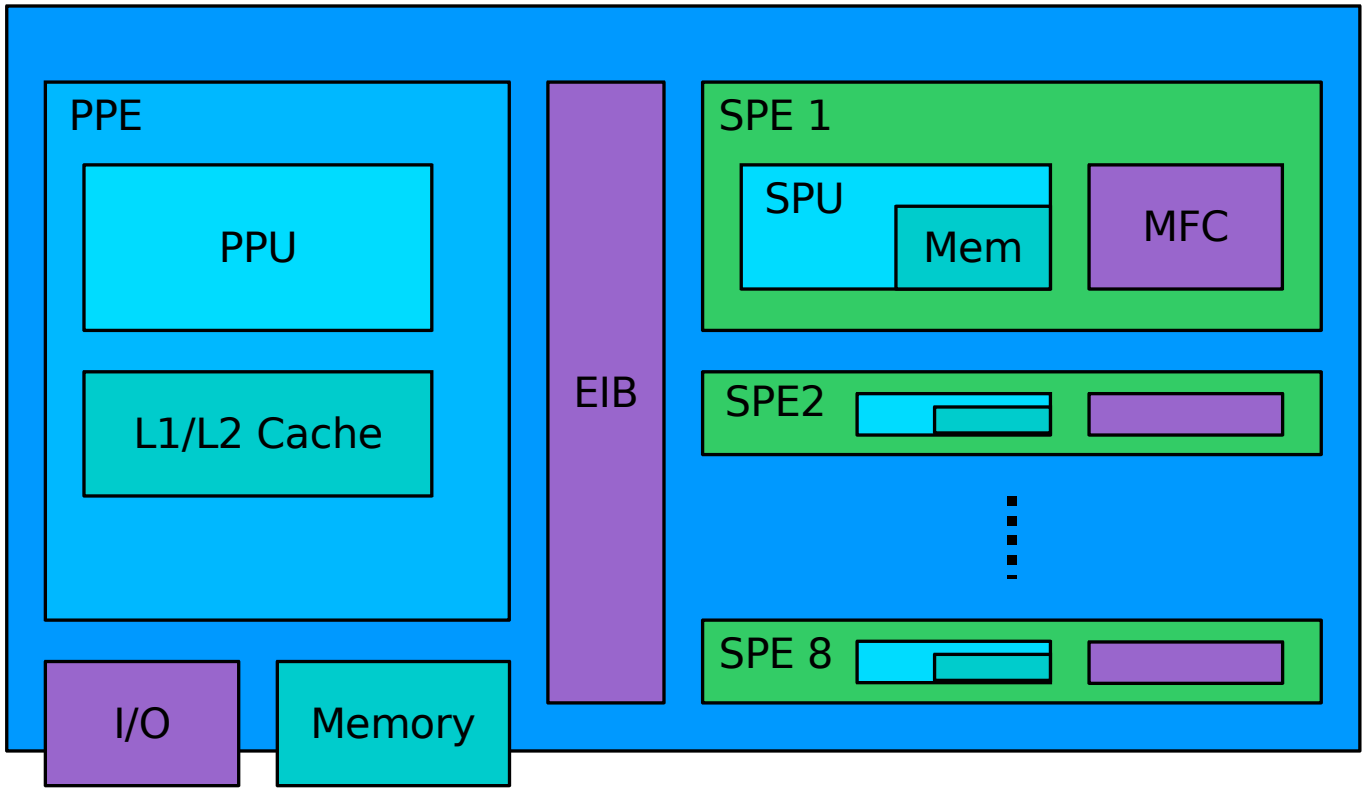
January 16, 2007

Outline

Cell Broadband Engine Processor



Cell Broadband Engine Processor



Features per SPE

- 128 bit SIMD
- 128 registers
- 3.2 Ghz clock speed
- 256 KiB local memory
- Memory flow controller for DMA
- 25 GB/s DMA data transfer
- “I/O Channels” for IPC
- No protected instructions

Registers

L1 Cache


YOU NEED THESE VITAMINS EVERY DAY
because your body can't store them up!
VEGEMITE
RICH in VITAMIN B1
RICH in VITAMIN B2

L2 Cache

Main Memory

File System

Linux on Cell/B.E. kernel components

- Platform abstraction
  arch/powerpc/platforms/{cell,ps3,beat}
- Integrated Interrupt Handling
- I/O Memory Management Unit
- Power Management
- Hypervisor abstractions
- South Bridge drivers (Spider, SCC, Axon)
- *SPU* file system

SPU file system

- Virtual File System
- */spu* holds SPU contexts as directories
- Files are primary user interfaces
- New system calls: *spu_create* and *spu_run*
- SPU contexts abstracted from real SPU
- Preemptive context switching (W.I.P)

spu_create

*int spu_create(const char *pathname, int flags, mode_t mode);*

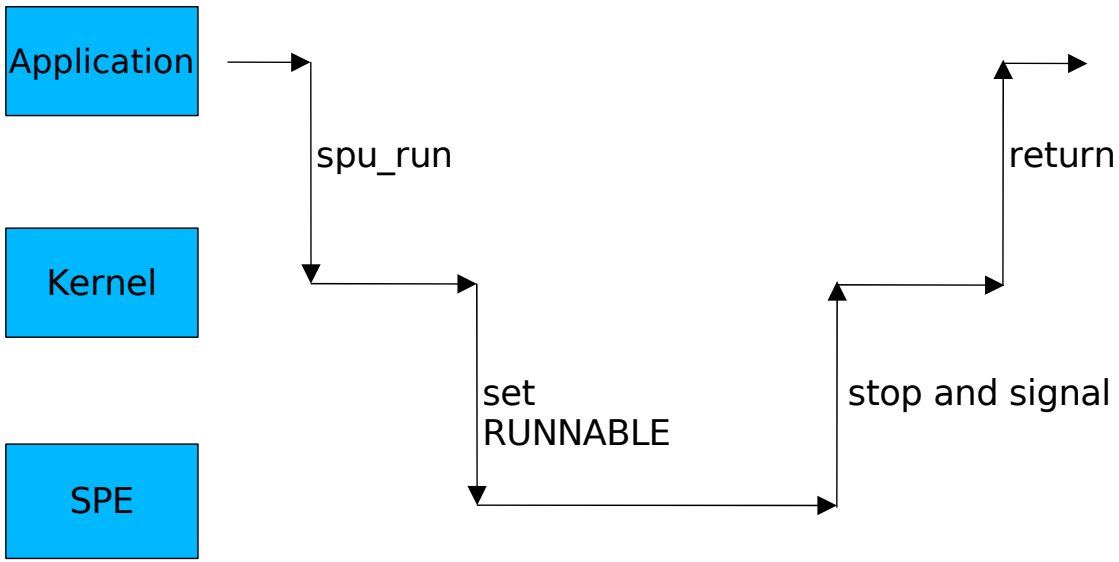
- creates a new context in pathname
- returns an open file descriptor
- context is gets destroyed when fd is closed

spu_run

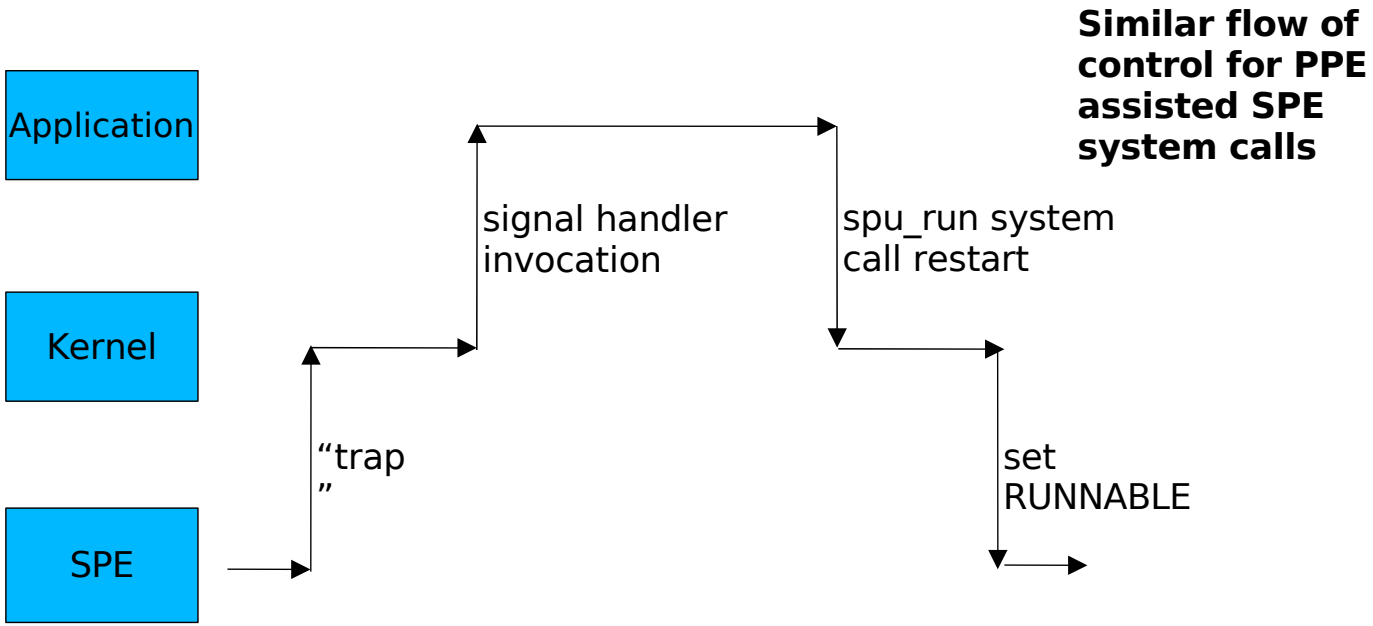
*uint32_t spu_run(int fd, uint32_t *npc, uint32_t *status);*

- transfers flow of control to SPU context fd
- returns when the context has stopped for some reason, e.g.
  - exit or forceful abort
  - callback from SPU to PPU
  - can be interrupted by signals

SPE execution control



SPE execution control – signals



SPE memory management

- Memory Flow Controller does DMA
- SPE local store $<-> >$ Process Virtual memory
- Page faults handled in *spu_run*

Asynchronous DMA

Virtual mapping

may include

- RAM
- Files
- Other SPEs
- I/O devices

PPE programming interfaces

- Asynchronous SPE thread API (“libspe 1.x”)
- *spe_create_thread*
- *spe_wait*
- *spe_kill*
- . . .

spe_create_thread implementation

- Allocate virtual SPE (*spu_create*)
- Load SPE application code into context
- Start PPE thread using *pthread_create*
- New thread calls *spu_run*

libspe sample code

```
#include <libspe.h>
int main(int argc, char *argv[], char *envp[])
{
        spe_program_handle_t *binary;
        speid_t spe_thread;
        int status;

        binary = spe_open_image(argv[1]);
        if (!binary)
                return 1;
        spe_thread = spe_create_thread(0, binary, argv+1, envp, -1, 0);
        if (!spe_thread)
                return 2;

        spe_wait(spe_thread, &status, 0);
        spe_close_image(binary);
        return status;
}
```

libspe sample code

```
#include <libspe.h>
int main(int argc, char *argv[], char *envp[])
{
        spe_program_handle_t *binary;
        speid_t spe_thread;
        int status;

        binary = spe_open_image(argv[1]);
        if (!binary)
                return 1;
        spe_thread = spe_create_thread(0, binary, argv+1, envp, -1, 0);
        if (!spe_thread)
                return 2;

        spe_wait(spe_thread, &status, 0);
        spe_close_image(binary);
        return status;
}
```

libspe sample code

```
#include <libspe.h>
int main(int argc, char *argv[], char *envp[])
{
        spe_program_handle_t *binary;
        speid_t spe_thread;
        int status;

        binary = spe_open_image(argv[1]);
        if (!binary)
                return 1;
        spe_thread = spe_create_thread(0, binary, argv+1, envp, -1, 0);
        if (!spe_thread)
                return 2;

        spe_wait(spe_thread, &status, 0);
        spe_close_image(binary);
        return status;
}
```

More libspe interfaces

- Event notification
  - int spe_get_event(struct spe_event *, int nevents, int timeout);
- Message passing
  - spe_read_out_mbox(speid_t speid);
  - spe_write_in_mbox(speid_t speid);
  - spe_write_signal(speid_t speid, unsigned reg, unsigned data);
- Local store access
  - void *spe_get_ls(speid_t speid);

GNU tool chain

- PPE support
  - Just another PowerPC variant...
- SPE support
  - Just another embedded processor...
- Cell/B.E. support
  - More than just PPE + SPE!

Object file format

- PPE: regular ppc/ppc64 ELF binaries
- SPE: new ELF flavour EM_SPU
  - 32-bit big-endian
  - No shared libraries
  - Manipulated via cross-binutils
  - New: Code overlay support
- Cell/B.E.: combined object files
  - embedspu: link into one binary
  - .rodata.spuelf section in PPE object
  - CESOF: SPE$->$PPE symbol references

gcc on the PPE

- handled by “rs6000” back end
- Processor-specific tuning
- pipeline description

gcc on the SPE

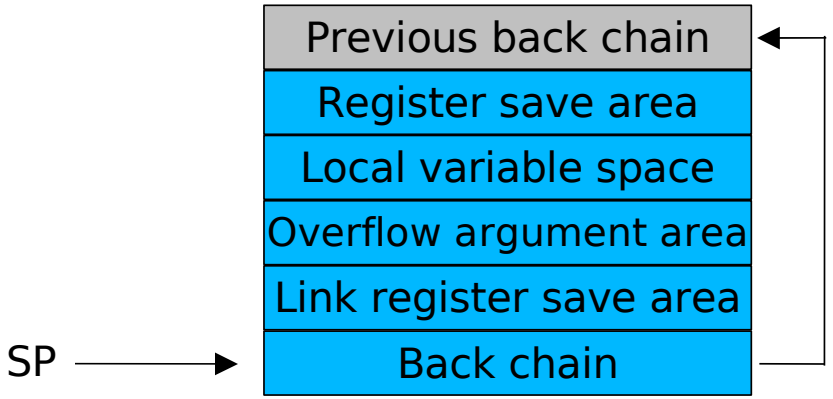
- Merged Jan 3rd
- Built as cross-compiler
- Handles vector data types, intrinsics
- Middle-end support: branch hints, aggressive if-conversion
- GCC 4.1 port exploiting auto-vectorization
- No Java

Combined Cell/B.E.

- Nothing for gcc yet
- single-source?
- OpenMP?
- some work from Barcelona Supercomputing Center

SPE Application Binary Interface

- Register usage
  - R0: link register, R1: stack pointer, R2: volatile
  - R3-R79: function arguments & return value, volatile
  - R80-R127: local variables, non-volatile
- Stack frame



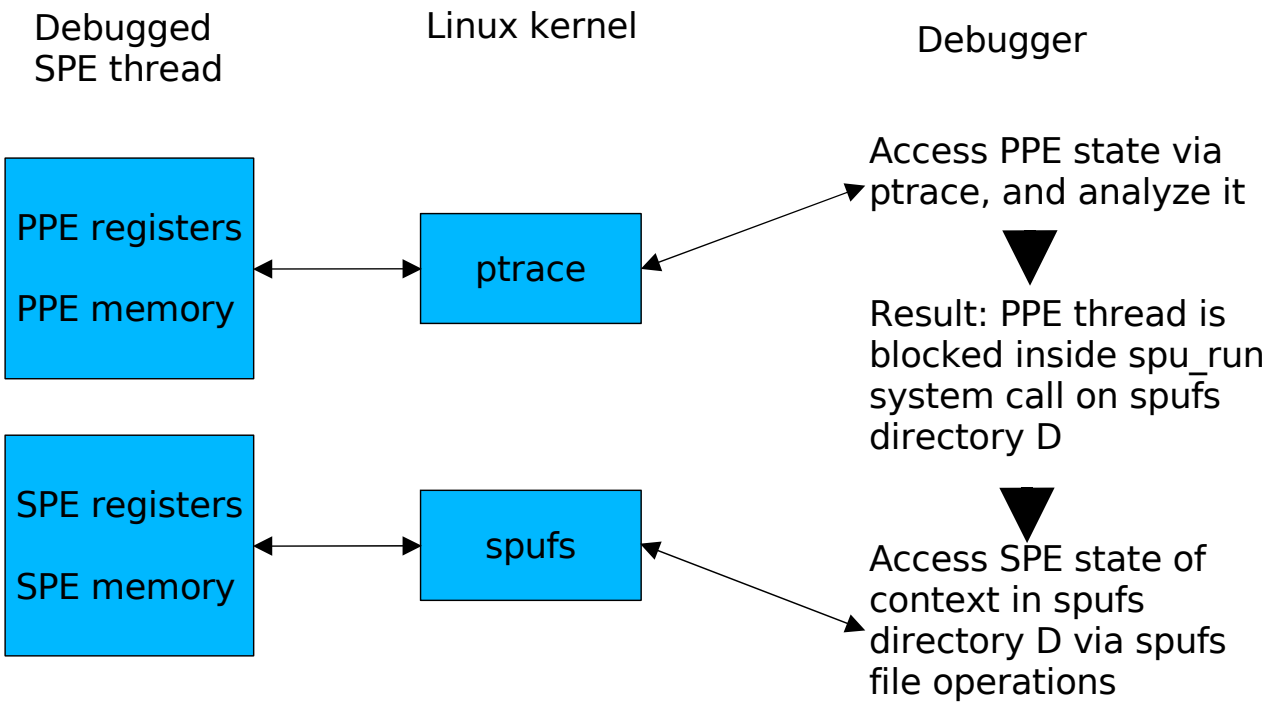
Cell/B.E. architecture documents

- Cell Broadband Engine Architecture
- SPU Instruction Set Architecture
- SPU Application Binary Interface Specification
- SPU Assembly Language Specification
- SPU C/C++ Language Extensions
- `http://cell.scei.co.jp/`

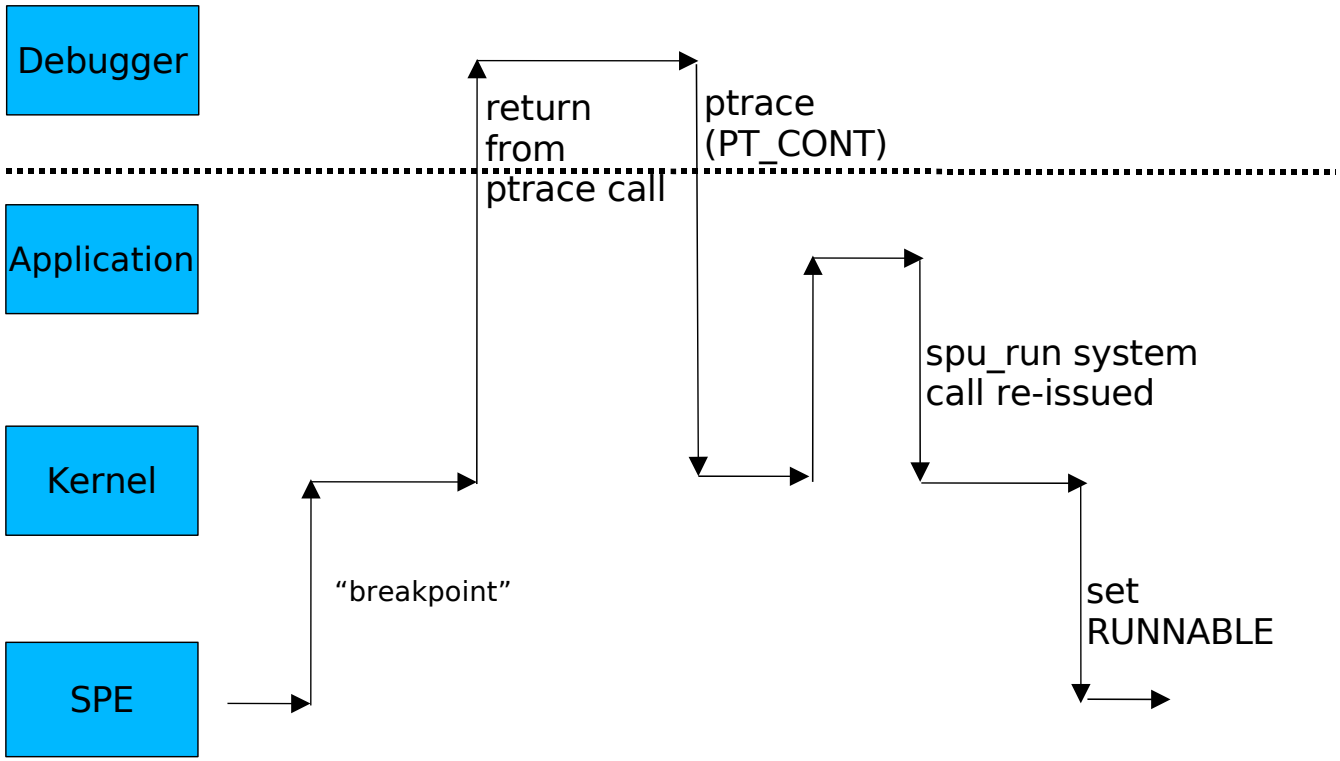
SPU-GDB operating modes

- Debug stand-alone SPE binary
  - Kernel support via binfmt_misc required
  - Allows to execute the full GDB test suite
- Attach to single SPE thread of running Cell application
  - Use simplified by debug assists in libspe runtime:
    print Linux PID of SPE thread on startup and wait for GDB attach
- Remote debugging support via gdbserver
- Combined PPE/SPE debugging

SPE debugger – process state access



SPE debugger – execution control



Remote debugging

- Extensions to the gdbserver protocol required
- Remote access to spufs file contents

Post-mortem debugging

- Core files:
  - LOAD sections for PPE memory
  - NOTE sections for per-thread PPE registers
- SPE support:
  - Additional NOTE sections per virtual SPE
  - Kernel support in 2.6.20

Existing proprietary applications

- Games
- Volume rendering
- Real-time Raytracing
- Digital Video
- Monte Carlo simulation

Obviously missing

- ffmpeg, mplayer, VLC
- VDR, mythTV
- Xorg acceleration
- OpenSSL

Obviously missing

- ffmpeg, mplayer, VLC
- VDR, mythTV
- Xorg acceleration
- OpenSSL
- Your project here

Questions?

Questions?

Credits

- Kernel, device drivers
  - Akira Iguchi, Benjamin Herrenschmidt Christian Krafft, Christophe Lamoureux, Geert Uytterhoeven, Geoff Levand, Ishizaki Kou, Jean-Christophe Dubois, Jens Osterkamp, Jeremy Kerr, Jim Lewis, Kevin Corry, Linas Vepstas, Maynard Johnson, Michael Ellerman
- SPU File System
  - Christoph Hellwig, Jeremy Kerr, Mark Nutter, Masato Noguchi
- SPE Library
  - Daniel Brokenshire, Dirk Herrendoerfer, Gerhard Stenzel, Kazunori Asayama
- Tool Chain
  - Andrew Pinski, Dwayne McConnell, Joel Schopp, Sidney Manning, Ulrich Weigand