PlayStation3

Michał Zarychta

Plan prezentacji

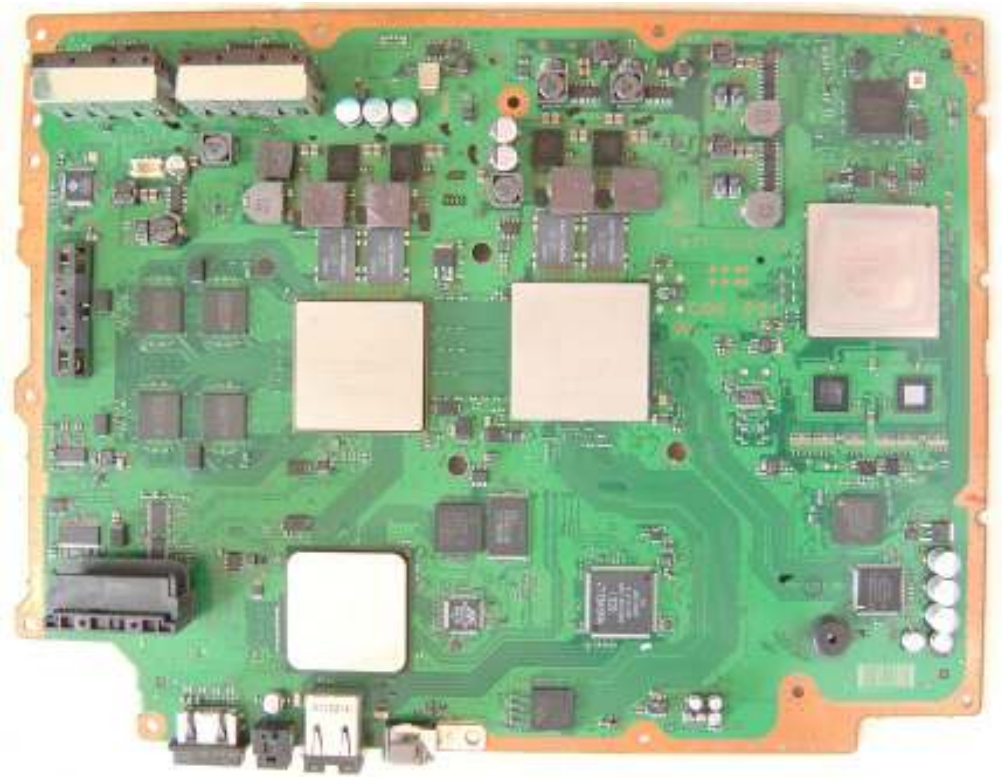
- Hardware
- Procesor CELL
- Techniki programowania
- Modele programistyczne
- Ograniczenia
- Hello WORLD

PlayStation3 Lab

- 200 x PS3
- atak na MD5 (RapidSSL)

Hardware

- Płyta główna



Hardware

- Pamięć
  - 256 MB – Rambus XDR
    - taktowanie: 400 MHz (3,2 GHz efektywnie)
    - przepustowość: 25,6 GB/s
  - 256 MB – GDDR3
    - taktowanie: 650 MHz (1,3 GHz efektywnie)
    - przepustowość: 22,4 GB/s

Hardware

- RSX
  - Hybryda G70/G71 (nVidia 7800GTX)
  - 8 jednostek vertex shader
  - 28 jednostek pixel shader
  - 8 potoków operacji rastrowych (ROP)
  - 256 MB GDDR3
  - 128 bitowa szyna danych dla pamięci
  - Architektura 90nm/65nm
  - API: PSGL lub LibGCM

Hardware

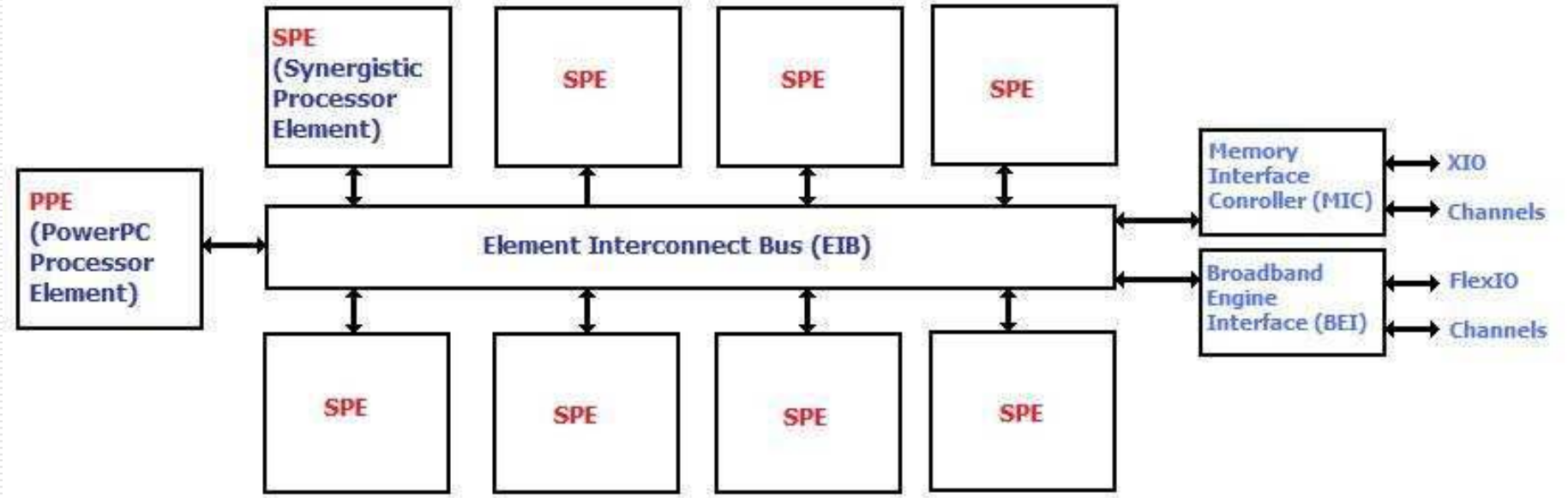
- Muzyka
  - Linear PCM, Dolby TrueHD, Dolby Digital Plus, dts-HD Master Audio, dts Digital Surround, SACD, DVD-Audio, dts 5.1 Music Disc, HDCD, CD, ACC, MP4, MP3, WMV, ATRAC, WAV
  - HDMI, TOSLINK, AV Multi Cable
- Obraz
  - MPEG-1, MPEG-2, MPEG-4, AVI, DivX, WMV, AVCHD
  - Full High Definition – 1920x1080p
  - HDMI, DVI-D, Component, SCART, S-Video
- Pamięć masowa
  - Blu-ray, DVD, CD
  - HDD SATA
- Komunikacja
  - Bluetooth, USB, WiFi, Gigabit Ethernet

CELL Broadband Engine

- 1 PPE (PowerPC Processor Element)
  - 3,2 GHz
  - 64 bit
  - 2 wątki
  - L1 cache: 32 kB danych + 32 kB instrukcje
  - L2 cache: 512 kB
- 8 SPE (Synergistic Processor Element)
  - 3,2 GHz
  - 64 bit
  - 1 SPE „uśpione”
  - 1 SPE dedykowane do bezpieczeństwa
  - 128 rejestrów dla SPE
  - 2 instrukcje na takt zegara
- Wydajność – 204,8 GFLOPS (Xbox – 115,2 GFLOPS)

CELL

- Architektura



Software

- System operacyjny
  - Game OS
  - Linux
    - Fedora Core 5
    - YellowDog 5.0
    - Gentoo PowerPC 64 edition
    - Debian
- IBM CELL SDK
- Kompilator
  - gcc –lspe2
  - spu-gcc

Short Vector SIMD

- Wysoka wydajność na wektorach
- Bardzo niska wydajność na skalarach
  - Tworzenie szkieletu
  - Optymalizacja
  - Przejście na wektory

Intra-Chip Communication

- DMA
  - Wymiana danych pomiędzy pamięcią główną i pamięciami lokalnymi jednostek SPE
  - Rozmiar wiadomości: 1,2,4,6,16 bajtów lub wielokrotność 16 bajtów (max 16kB)
  - Brak szeregowania wiadomości DMA
  - Transfer wiadomości nieblokujący
  - Maksymalnie 16 odwołań w kolejce (dla każdego z SPE)
- spe_get_ls()

Intra-Chip Communication

- Mailboxes
  - Krótkie wiadomości – 32 bity
  - PPE do SPE, pomiędzy SPE
  - Kolejki FIFIO
  - Transfer wiadomości blokujący

CorePy

- projekt Indiana University (open source)
- język programowania Python
- libspe 1.x – bezpośrednia komunikacja z SPU
- konsola SPU – instrukcje dla SPU w czasie rzeczywistym (+ debugger)

Octopiler

- kompilator badawczy IBM
- 2 ważne zmiany
  - automatyczne SIMD
  - automatyczne zarządzanie wykorzystaniem pamięci używanej do obliczeń równoległych

RapidMind

- komercyjne narzędzie
- rozszerzenie umożliwiające korzystanie ze zwykłego C++
- framework do uruchamiania procesów równoległych
- dynamiczna kontrola przepływów

PeakStream

- platforma do tworzenia palikacji
- 4 komponenty
  - PeakStream APIs
  - PeakStream VM
  - PeakStream Profiler
  - PeakStream Debugger
- podstawowy typ - tablice

MPI Microtask

- projekt badawczy IBM
- optymalizacja procesowania równoległego
- odpowiedzialność preprocesora
  - generowanie grafu procesów
  - klastrowanie zadań
  - harmonogramowanie zadań
- odpowiedzialność środowiska uruchomieniowego
  - synchronizacja
  - Zmiana kontekstu
  - buforowanie komunikacji
- mikro zadania = virtualne SPE

Cell Superscalar

- framework (Barcelona Supercomputer Center)
- fragment projektu – Grid Superscalar
- kod pisany sekwencyjnie
- uruchomienie z wykorzystaniem grafu zależności
  - węzeł – funkcje SPE
  - krawędź – przekazanie danych
- przetwarzanie równoległe – niezależne wierzchołki dla różnych SPE w tym samym czasie

The Sequoia Language

- projekt badawczy na Stanford University
- język programowania dla programów przenośnych wykorzystujących hierarchię pamięci
- implementacja zadań jako funkcji
- mapa wykonania
- zadanie związane z węzłem w drzewie pamięci
- wirtualne poziomy pamięci
- globalna wirtualna pamięć dzielona

Mercury Multi-Core Framework

- komercyjny produkt – biblioteka MFC
- PPE – „manager” odpowiedzialny za uruchomienie i kontrolę
- SPE – „pracownicy” odpowiedzialni za przetwarzanie danych
- synchronizacja „pracowników” z wykorzystaniem semaforów
- dwustronna komunikacja (podobna do MPI)
- Data Reorganization Interface – komunikacja DMA z wykorzystaniem *put* i *get*

IBM Accelerated Library Framework

- środowisko programistyczne mające na celu uproszczenie kodowania aplikacji równoległych
- ALF to część CELL SDK
- podział programistów
  - programiści jądra
  - programiści bibliotek
  - programiści aplikacji
- dwa rodzaje zadań
  - zadania kontrolne – generowanie zadań z PPE do SPE
  - zadania obliczeniowe – uruchamiane równolegle na SPE
- blokujące bufory danych

Hello WORLD

- SPE

```
1 #include <stdio.h>
2
3 int main(unsigned long long spe, unsigned long long argp,_
4                                   unsigned long long envp)
5 {
6     printf("Hello WORLD");
7
8     return 0;
9 }
```

Hello WORLD

- PPE

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <libspe2.h>
 4
 5 int main(int argc, char **argv)
 6 {
 7     int ret;
 8
 9     spe_context_ptr_t spe;
10     spe_program_handle_t *prog;
11     unsigned int entry;
12     spe_stop_info_t stop_info;
13
14     prog = spe_image_open("hello_world_spe.elf");
15     if (!prog) {
16         perror("spe_image_open");
17         exit(1);
18     }
19
20     spe = spe_context_create(0, NULL);
21     if (!spe) {
22         perror("spe_context_create");
23         exit(1);
24     }
25
26     ret = spe_program_load(spe, prog);
27     if (ret) {
28         perror("spe_program_load");
29         exit(1);
30     }
31
32     entry = SPE_DEFAULT_ENTRY;
33     ret = spe_context_run(spe, &entry, 0, NULL, NULL, &stop_info);
34     if (ret < 0) {
35         perror("spe_context_run");
36         exit(1);
37     }
38
39     ret = spe_context_destroy(spe);
40     if (ret) {
41         perror("spe_context_destroy");
42         exit(1);
43     }
44
45     ret = spe_image_close(prog);
46     if (ret) {
47         perror("spe_image_close");
48         exit(1);
49     }
50
51     return 0;
52 }
```

Koniec

- Pytania?