

Seminar Parallele Algorithmen

**Cell-Prozessor
und
Paralleles Sortieren**

Christian Himmelsbach & Michael Hobmeier

Struktur:

- **Einleitung**
- Architektur
- Cell Programmierung
- Paralleles Sortieren
- Quellen

Einleitung

„Ein Kern“ Architekturen waren ausgereizt und nur noch geringer Leistungszuwachs war zu erwarten.

Bsp:

- Speicherlatenz
 - Große Differenz zwischen CPU und Speicherzugriffen.
- Schlechte Energie-Effizienz
 - hohe Verlustleistung

Lösungsansätze

- Hohe Nebenläufigkeit
 - Multicore
- Verbergen von Speicherlatenz
 - Double-Buffering/Asynchronität
- Trennung von Kontroll- und Berechnungsebene
 - Funktionsspezialisierung

=> Cell Broadband Engine Architecture(CBEA)

Cell Broadband Engine

Der Cell-Prozessor wurde gemeinsam von IBM, Toshiba und Sony entwickelt.

Ziele :

- Hohe Performanz
- Energieeffizienz
- Kosteneffizienz
- Großes Anwendungsfeld

Historie

- 2000
 - Gründung von STI (**S**ony **T**oshiba **I**BM)
- 2001
 - Entwicklungsstart der Cell Broadband Engine (CBE)
- 2005
 - Erste technische Einzelheiten werden bekannt
 - Linux-Cell-Blade-Prototyp auf E3 vorgestellt
 - Linux-Distribution vom Barcelona Supercomputing Center veröffentlicht

Es wurden ca. 400Mio Dollar investiert und an die 400 Menschen waren beteiligt.

Anwendungen Hardware

- Hardware :
 - IBM: Blade-Server
 - Sony Playstation 3



Anwendungen Hardware

- Toshiba plant Cell-Einsatz in End-Verbraucher Geräten (zB. HDTV) und stellte kürzlich ein Notebook mit zusätzlichen Cell-Prozessor vor.
- Mercury Computing Systems bietet Workstations auf Blade-Server-Basis an und vertreibt PCIe Beschleunigerkarten mit Cell-Prozessor.



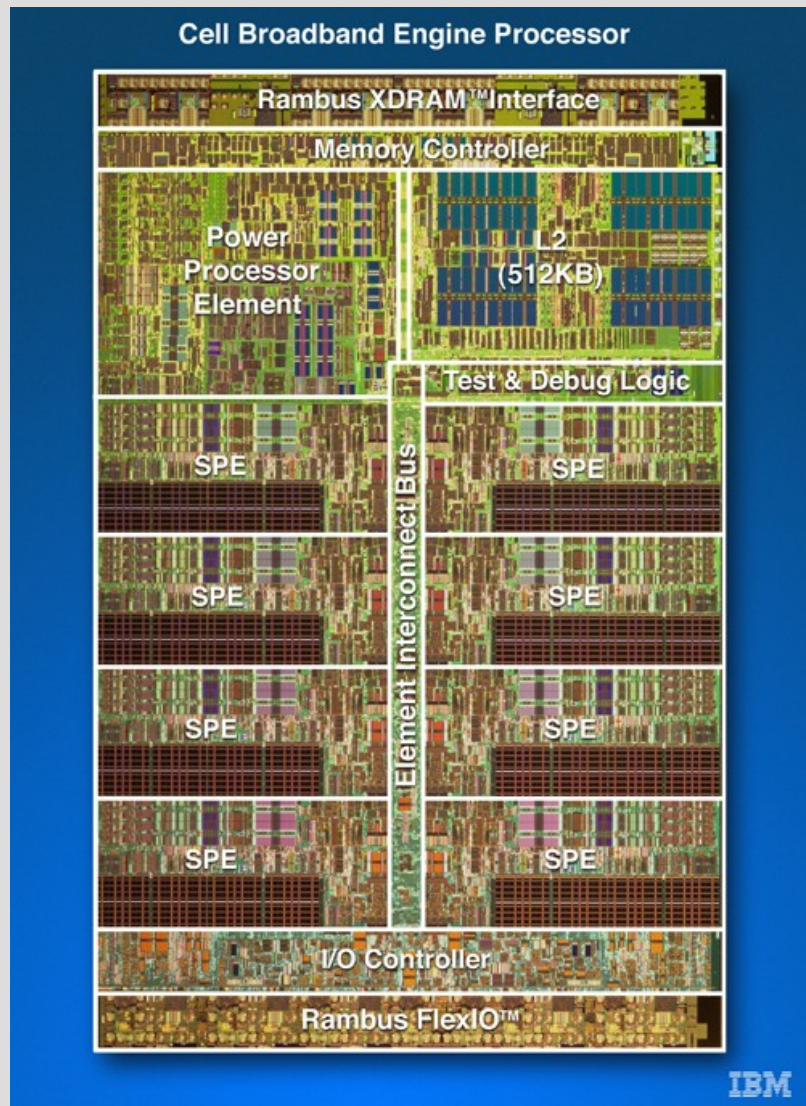
Anwendungen Software

- Denkbare Software-Anwendungen
 - Multimedia-Anwendungen
 - Simulationen
 - Bild-/Video-Verarbeitung

Struktur:

- Einleitung
- **Architektur**
- Cell Programmierung
- Paralleles Sortieren
- Quellen

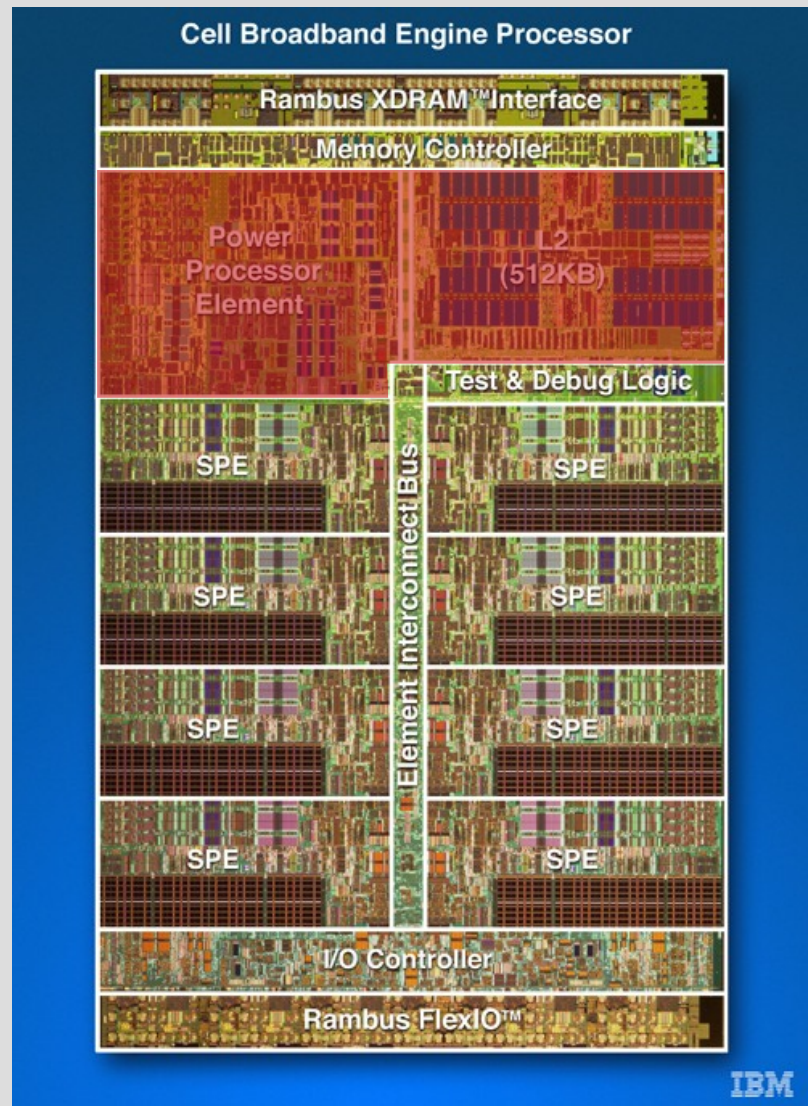
Architektur



- 64-bit Hauptkern (PPE)
- 8 x 32-bit Kerne (SPE)
- EIB
- Memory Controller
- I/O Controller

Architektur

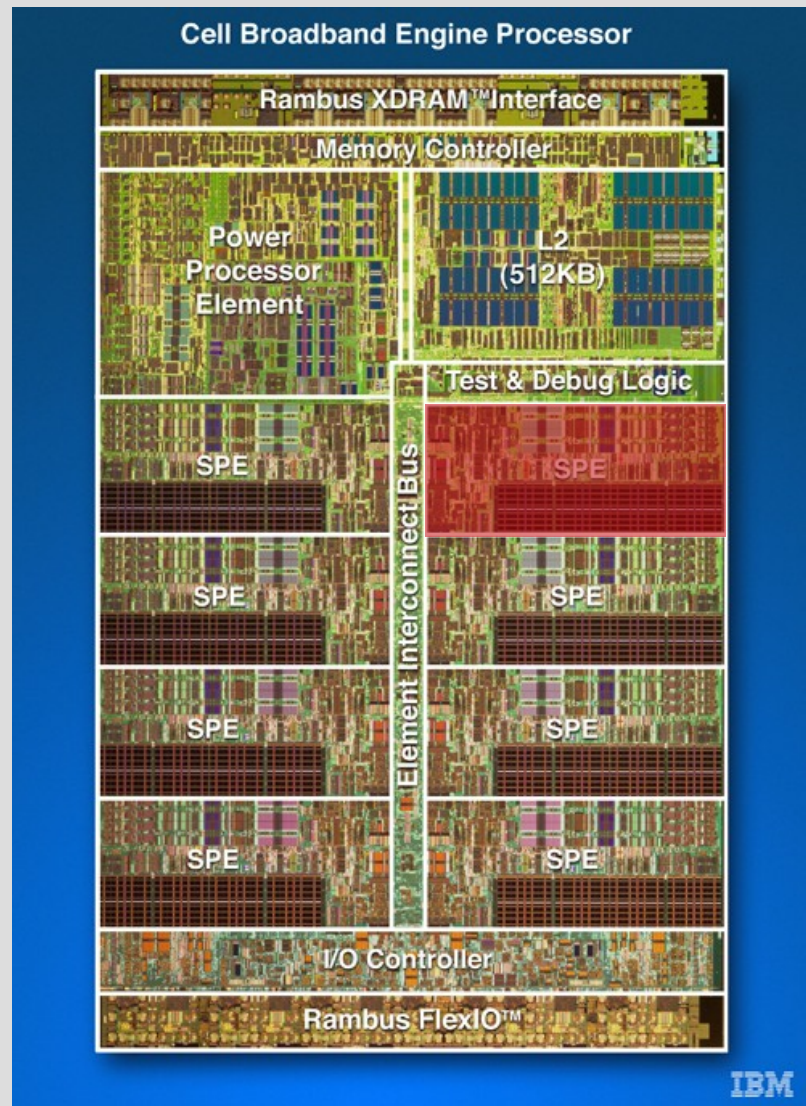
Power Processor Element



- RISC-Basis
- Multithreadfähig
- 3.2 Ghz
- Cache
 - L1: 64 KB
 - 32 KB Instruction-Cache
 - 32 KB Data-Cache
 - L2: 512 KB
- AltiVec
 - 128-bit SIMD

Architektur

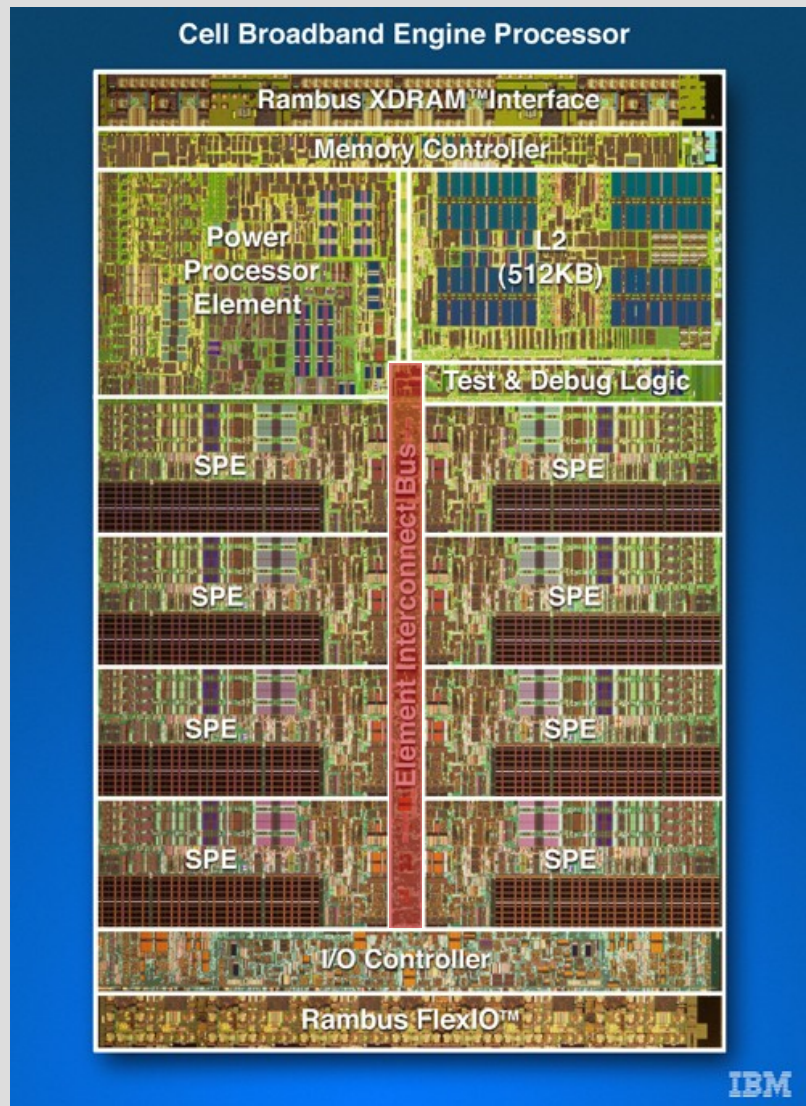
Synergistic Processing Element



- RISC-Basis
- 3.2 Ghz
- 128-bit SIMD
 - 8,16,32 und 64 bit
- 256 KB Local Store (LS)
 - Instruction & Data
 - 16 Byte/Takt
- Memory Flow Controller
 - Kommunikation

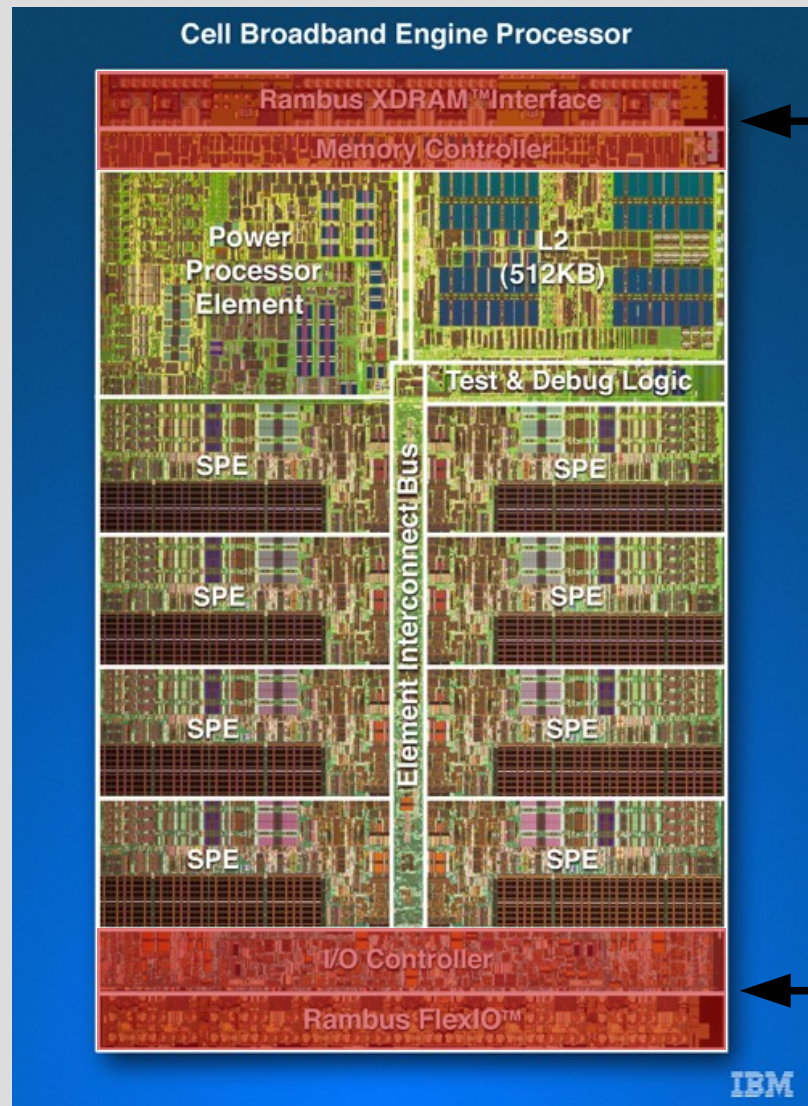
Architektur

Element Interconnect Bus



- Kommunikations-Kanal für:
 - PPE
 - 8 SPEs
 - MIC
 - 2 externe Schnittstellen
=> maximal 12 Beteiligte
- 4 Kanäle a 16 Byte
- 96 Byte/Takt
- 300 GB/s nicht realistisch
 - ca. 200 GB/s

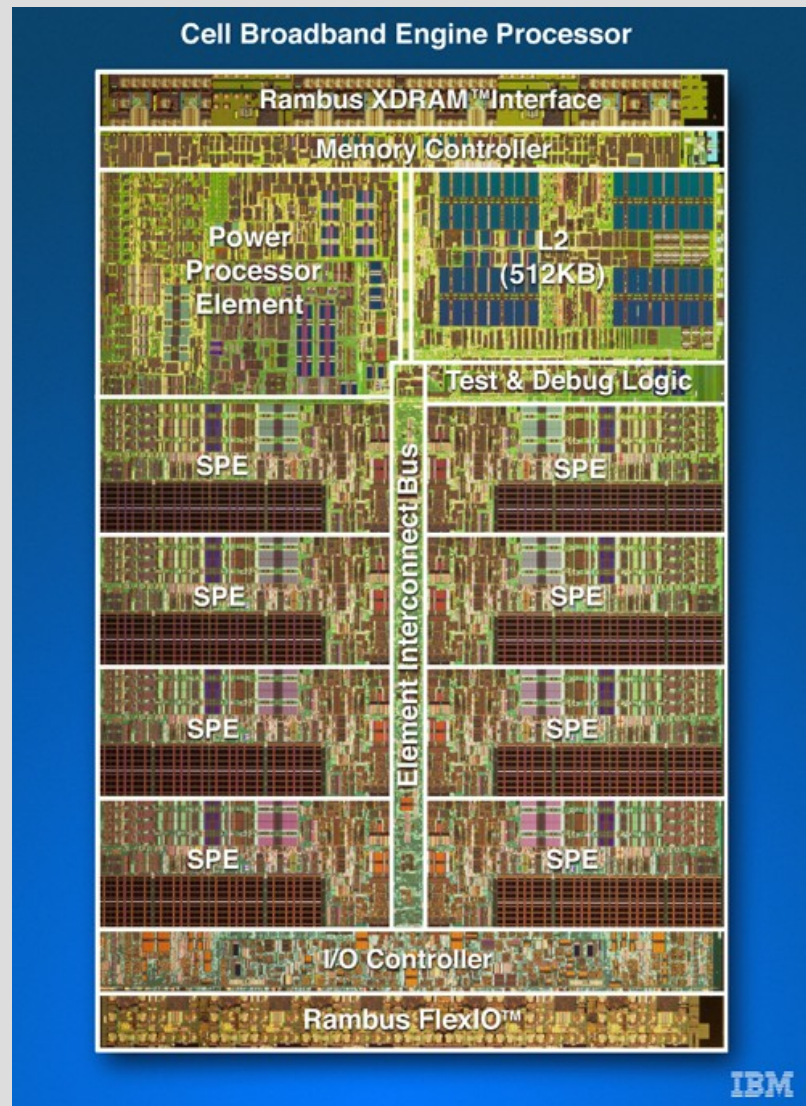
Architektur RAM-Interface



- Anbindung an Arbeitsspeicher (MIC / Rambus XIO)
 - ca. 25.6 GB/s Durchsatz
- Externe Schnittstelle (FlexIO / I/O Controller)
 - ca. 60 GB/s Durchsatz

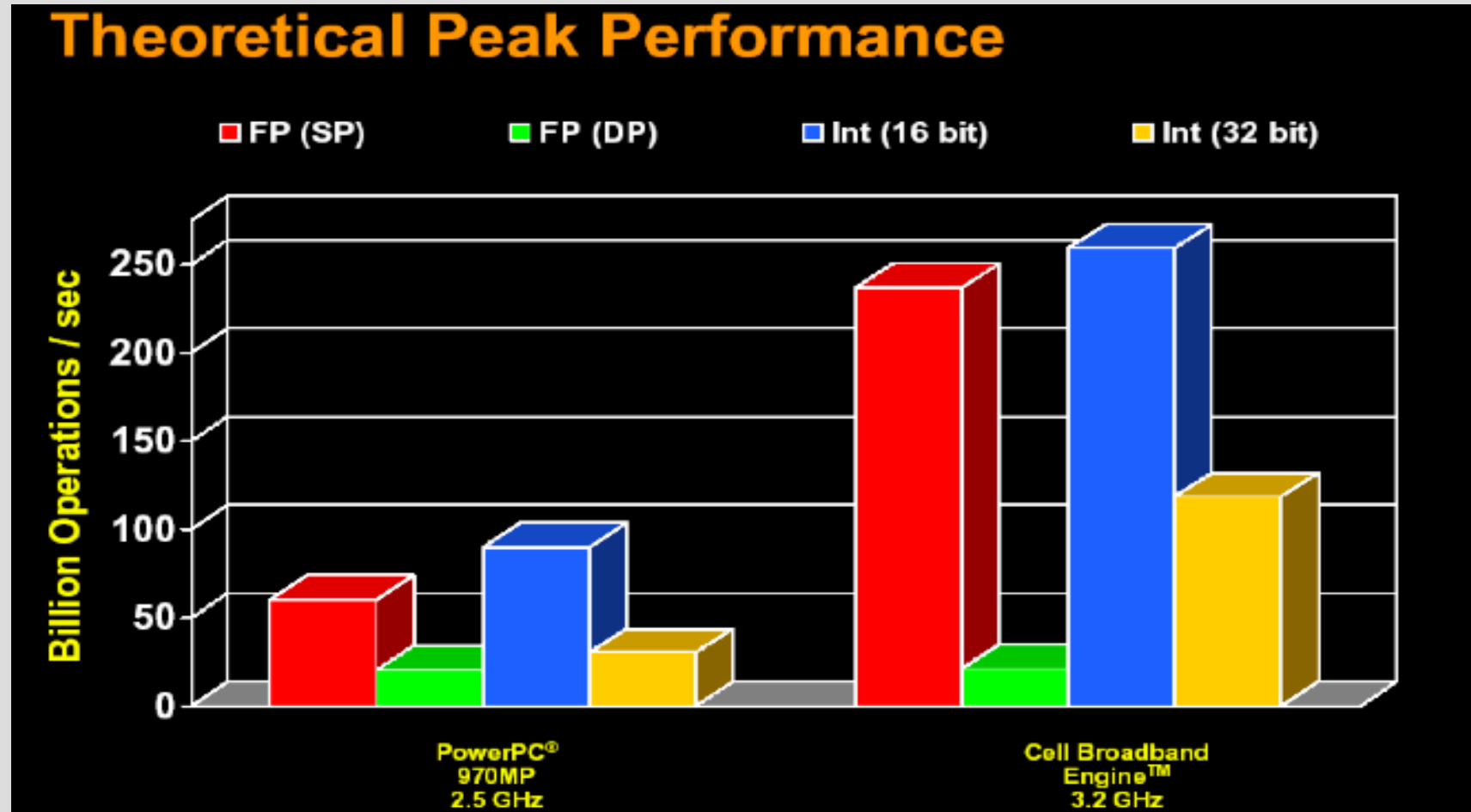
Architektur

Kennzahlen/Daten



- Transistoren: 241 mio.
- Fläche: 235 mm²
- Fertigung
 - anfangs 90 nm
 - heute 65 nm
 - zukünftig 45 nm
- Takt: 3.2 GHz
 - im Labor > 4.0 Ghz!
- > 200 GFlops (SP)
- > 20 Gflops (DP)

Architektur Vergleich



Cell im Vergleich mit einem PowerPC-CPU (wurde zB. von Apple in Mac's verwendet)

Struktur:

- Einleitung
- Architektur
- **Cell Programmierung**
- Paralleles Sortieren
- Quellen

Cell Programmierung

Aufgabenverteilung

- Wie werden Aufgaben effizient verteilt?
- PPE dient als Kontrolleinheit
 - => Kontrollebene
 - Hauptsächlich Verteilung von Aufgaben
 - Zusammentragen von Ergebnissen
 - Dateizugriffe
- SPEs werden als „Arbeitstiere“ eingesetzt
 - => Berechnungsebene
 - Datenintensive Berechnungen

Cell Programmierung

Erste Schritte

- Mögliche Hardware für Cell-Programmierung
 - BladeCenter
 - Play Station 3
 - Emulation auf
 - Linuxsystemen (indirekt Windows)
- Unterbau ist i.d.R. Linux
- Entwicklungsumgebung => SDK
 - Von IBM zur Verfügung gestellt
- Programmierung in (ASM,) C, C++
 - Spezifikationen für
 - Assembler-Befehle
 - Erweiterungen der Sprachen C/C++
- Bibliotheken

Cell Programmierung

Erste Schritte

- Was haben wir benutzt?
 - Play Station 3
 - 6 SPEs
 - 256 MB Arbeitsspeicher
 - Yellow-Dog-Linux
 - IBM-SDK
- Verwendete Programmiersprache
 - C/C++ plus „Language Extensions“
- Bibliotheken
 - Posix-Threads
 - Libspe Version 2

Cell Programmierung

PPE Programmieren

- Standard C/C++ Programme
- Posix-Threads
 - Standard für Multitasking
- Vektor-Datenverarbeitung mit AltiVec-Einheit
 - Einheit für SIMD-Operationen die schon bei früheren PowerPC-Architekturen zum Einsatz kam
 - Überwiegend : `vec_* <=> spu_*`
- SPE-Verwaltung
 - SPE-Programm laden, starten, ...

Beispiele folgen


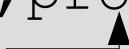
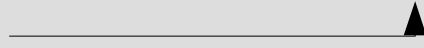
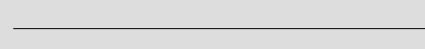

Cell Programmierung

PPE Programmieren - Threads

- POSIX-Threads
- Thread erzeugen
 - `pthread_create(&thread, 0, func, args)`
 - Handle auf Thread 
 - Auszuführende Funktion 
 - Parameter für Funktion 
- Auf Beendigung eines Threads warten
 - `pthread_join(thread, NULL)`
- Eigenen Thread beenden
 - `pthread_exit(NULL)`

Cell Programmierung

PPE Programmieren - SPE

- Spe-Kontext anfordern
 - `spe_context_create(0, 0) : id`
 - Spe-Id 
- Programm an Kontext binden
 - `spe_program_load(id, prog)`
 - Programm-Code-Handle 
- Programm starten und warten bis beendet
 - `spe_context_run(id, &startPoint, 0, args, 0, &info)`
 - Start-Adresse 
 - Argument für Spe-Programm 
 - Terminierungs-Infos 
- Spe-Kontext zerstören
 - `spe_context_destroy(id)`

Cell Programmierung

SPU Programmieren

- C/C++ Programme mit Erweiterungen
 - Vektor-Datentypen und SIMD-Verarbeitung
- Anmerkungen :
 - Speicherplatz beachten (nur 256kB für alles!)
 - Möglichst immer SIMD-Befehle benutzen
 - Optimiert für Single Precision, Double Precision nicht performant
 - Double Buffering bei großen Datentransfer anwenden.
 - „Kontrollstruktur armer Code“, da keine Hardware für Branch Prediction vorhanden.
 - Rekursion gefährlich, da Stack nicht verwaltet ist und Daten/Instruktionen überschreiben kann

Cell Programmierung

SPU Programmieren - SIMD

- Typen
 - Vektoren elementarer Datentypen (128 bit)
 - Beispiele:
 - `vector float <=> float[4]`
 - `vector char <=> char[16]`
- **Arithmetische Befehle**
 - `spu_add(a,b) : c`
- **Vergleiche**
 - `spu_cmp[gt|eq](a,b) : t`
 - `spu_sel(a,b,t) : c`

Cell Programmierung

SPU Programmieren - SIMD

- **Logik-Befehle**
 - `spu_and(a,b) : c`
 - `spu_or(a,b) : c`
- **Shift, Rotation**
 - `spu_sl(a,count) : c`
 - `spu_rl(a,count) : c`
- **und viele mehr ...**

Cell Programmierung

Kommunikation

- Kommunikations-Arten
 - Direct Memory Access (DMA)
 - Mailbox
 - Notification

Cell Programmierung

Kommunikation - DMA

- SPE greift direkt auf Speicher der PPE oder anderer SPEs (lesend/schreibend) zu.
 - PPE übergibt SPE Adresse auf Kontrollblock.
 - In Kontrollblock stehen Adressen, auf die von SPE aus zugegriffen werden können.
- Asynchroner Zugriff
- Alignment = $n \times 16$ byte !
- Kommunikation über Kanäle

Cell Programmierung

Kommunikation - DMA

- Kommunikations-Tag anfordern
 - `mfc_tag_reserve() : tag`
- Daten aus Hauptspeicher beziehen
 - `mfc_get(la, ea, size, tag, 0, 0)`
- Daten in Hauptspeicher schreiben
 - `mfc_put(la, ea, size, tag, 0, 0)`

Varianten: `mfc_[get|put][b|f]`

Barrier: Alle folgenden DMA-Befehle werden danach ausgeführt

Fence: Alle vorherigen DMA-Befehle werden zuerst ausgeführt

- Warte bis Operationen auf allen Kanälen aus mask abgeschlossen sind
 - `mfc_write_tag_mask(mask)`
 - `mfc_read_tag_status_all()`

Cell Programmierung

Kommunikation - Mail

- Kommunikationspartner
 - SPEs und PPE
- Nachrichten können zwischen Partnern ausgetauscht werden
- Befehle (von SPE aus)
 - Inbox auslesen
 - `spu_read_in_mbox()` : message
 - `spu_stat_in_mbox()` : message_count
 - Outbox
 - `spu_write_out_mbox(message)`
 - `spu_stat_out_mbox()` : message_count
- Nachrichten in Form von `unsigned int`

Cell Programmierung

Kommunikation - Mail

- Befehle (von PPE aus)
 - In Inbox von Spe schreiben
 - `spe_in_mbox_write(speid, messages*, message_count, behavior)`
 - `spe_in_mbox_status(speid)`
 - Outbox von Spe auslesen
 - `spe_out_mbox_read(speid, messages*, message_count)`
 - `spe_out_mbox_status(speid)`

Cell Programmierung

Kommunikation - Notification

- PPE kann Signale an SPE verschicken
 - Pro SPE zwei Signalkanäle verfügbar
- PPE:
 - `spe_signal_write(speid, signal_reg, data)`
- SPU:
 - `(uint32_t) spu_read_signalX()`
 - `(uint32_t) spu_stat_signalX()`
 - X steht für Kanal (1 o. 2)

Cell Programmierung

Fazit

- Programmierung nicht trivial!
- Um hohe Performanz zu erreichen, ist hoher Programmieraufwand erforderlich.
 - in Assembler programmieren usw.

Struktur:

- Einleitung
- Architektur
- Cell Programmierung
- **Paralleles Sortieren**
- Quellen

Paralleles Sortieren

Parallelisierung von Mergesort

- In „nicht rekursiver“ Variante.

Normaler Mergesort-Algorithmus wird von hinten aufgerollt.

Paralleles Sortieren

Erklärung

5 6 8 2 3 2 9 4

5 6

2 8

← Paare werden verschmolzen

2 3

4 9

5 6 2 8 2 3 4 9

← Paare sind nun aufsteigend sortiert

2 5 6 8

← Schrittweite wird verdoppelt und Folgen werden wieder verschmolzen

2 3 4 9

2 2 3 4 5 6 8 9

Der Vorgang wird solange wiederholt bis die komplette Folge sortiert ist.

Paralleles Sortieren

Implementierung

- 1. Zerlegen der Folge
 - Die Zahlenfolge wird in Teilfolgen zerlegt.
 - Für X SPE's X Teilfolgen
 - Für jeden SPE wird ein Thread gestartet und die Adresse der jeweiligen Teilfolge wird übergeben.

Paralleles Sortieren

Implementierung

- 2. Sortieren der Teilfolgen
 - SPE lädt Chunks der Teilfolge und sortiert diesen mit Mergesort (nicht rekursiv).
 - Mit Hilfe von Double Buffering werden nacheinander alle Chunks geladen und sortiert.
 - Sobald ein Chunk sortiert ist, wird er wieder zurück an PPE geschickt.
 - Diese weitere Unterteilung ist notwendig, aufgrund des begrenzten lokalen Speichers

Paralleles Sortieren

Implementierung

- 3. Merge der sortierten Teilfolgen
 - Sobald alle Threads fertig sind, erledigt PPE den Rest.
 - Der von den SPE's gelieferte 'Unterbau' wird mit Hilfe von Mergesort fertig sortiert.
 - Leider auch der Flaschenhals unserer Implementierung.

Paralleles Sortieren

Fazit

Implementierung so sicher nicht optimal.

Jedoch ist die Portierung vieler Algorithmen auf die Architektur des CELL nicht ganz trivial.

Gelingt es die Vorteile des Cpu's wirklich auszunutzen, kann man sehr hohe Leistung erzielen.

Beispiel: CellSort (Benötigt auf BladeCenter für 0,5 GB Daten nur ca 4s)

Struktur:

- Einleitung
- Architektur
- Cell Programmierung
- Parallels Sortieren
- **Quellen**

Quellen

- IBM
 - [CBE Resource Center](#)
(CBE Programming Guide\Tutorial\Handbook und Spezifikationen zur C/C++ Language Extensions sowie den Libraries)
 - [The Cell Project](#)
- [Wikipedia](#)
- [Linksammlung](#) im Forum von [CellPerformance.com](#)

Cell-Prozessor

Vielen Dank für Ihre Aufmerksamkeit!

Cell-Prozessor

Fragen ?