UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Ausarbeitung zum Vortrag

Multicore-Architekturen

Philipp Wendler

Seminar
Multicore-Programmierung
am
Lehrstuhl für Programmierung

30. April 2009

Zusammenfassung

Das Ziel dieser Arbeit ist es, aktuelle Multicore-Architekturen vorzustellen. Dabei werden sowohl grundlegende Technologien und Architekturen, als auch konkrete Implementierungen in Gestalt von aktuellen Multicore-Chips präsentiert. Außerdem wird ein Vergleich mit anderen Varianten von Parallelität in Hardware vorgenommen.

Inhaltsverzeichnis

1 Einführung

1.1 Inhalt

In dieser Arbeit werden aktuelle Multicore-Architekturen vorgestellt. Dabei wird zunächst diese noch relativ neue Art Hardware-Parallelität allgemein erläutert und in Beziehung zu den wichtigsten bereits etablierten Varianten gesetzt (Abschnitt 1.3). Im Hauptteil werden zum einen verschiedene Technologien und Architekturen vorgestellt, die für Multicores benutzt werden können. Zum anderen werden fünf konkrete Chips ausgewählt und als Beispiele näher vorgestellt. Mit wenigen Ausnahmen wird sich dabei auf Prozessoren und Technologien beschränkt, die in dieser Form aktuell auf dem Markt verfügbar sind. Außerdem werden nur Chips betrachtet, die zumindest prinzipiell fähig zu General-Purpose-Computing sind und weit verbreitet sind. Auf Grund der hohen Bedeutung bei Multicores werden auch auf Stream-Computing optimierte Chips mit eingeschlossen. Diese Form der Datenverarbeitung beinhaltet Probleme, die auf einer große Menge von unabhängigen Daten immer gleiche Berechnungen ausführen. Dies tritt z.B. bei der Berechnung von Echtzeit-3D-Grafik in Computerspielen auf. Ohne diese Einschränkungen wäre der Umfang dieser Arbeit nicht zu halten. Ebenfalls verzichtet wird auf die Angabe von konkreten Geschwindigkeitswerten. Auf Grund der unterschiedlichen Zielsetzung der Prozessoren wäre eine Vergleichbarkeit sowieso nicht gegeben.

1.2 Definitionen

Da Multicore-Prozessoren erst seit wenigen Jahren allgemein verbreitet sind, hat sich in manchen Bereichen noch kein einheitlicher Sprachgebrauch herausgebildet. Die Marketing-Abteilungen einiger Hersteller nutzen dies, um ihr Produkt besonders leistungsfähig erscheinen zu lassen, so dass gerade bei direkten Vergleichen Vorsicht geboten ist. Gerade so grundlegende Begriffe wie “Kern” und “Prozessor” werden unterschiedlich verwendet. Letzterer z.B. kann alles zwischen einem kompletten Chip und einer einzelnen

Recheneinheit bedeuten. Daher erscheint es nötig, für diese Arbeit eine klare Definition dieser Begriffe anzugeben.

Definition 1 *Ein Prozessor ist eine nicht teilbare Einheit von Rechen- und Hilfseinheiten.*

In den meisten Fällen besteht ein Prozessor aus genau einem Chip, allerdings gibt es Multicores, die aus mehreren Chips auf einem Sockel bestehen.

Definition 2 *Ein Kern ist ein Teil eines Prozessors, der alles enthält um mindestens einen Thread eigenständig ausführen zu können.*

Dies beinhaltet, dass die Ausführung zweier Threads, die gleichzeitig auf verschiedenen Kernen laufen, sich nicht gegenseitig beeinflussen darf.

Ein Multicore-Prozessor ist demnach ein Prozessor mit mehr als einem Kern, auf dem also mehrere Threads gleichzeitig und unabhängig voneinander ausgeführt werden können.

1.3 Multicore im Vergleich mit anderen Arten von Hardware-Parallelität

In einem System können unterschiedliche Arten von Parallelität benutzt werden. Die drei am weitesten verbreiteten Klassen sind [Fly72]:

SIMD Single Instruction Multiple Data: Ein einzelner Befehl wird für mehrere Werte gleichzeitig ausgeführt (z.B. die Vektoraddition als parallel Ausführung mehrer herkömmlicher Additionen).

SPMD Single Program Multiple Data: Ein ganzes Programm wird mehrfach gleichzeitig ausgeführt, die Eingabedaten können aber jeweils unterschiedlich sein.

MPMD Multiple Program Multiple Data: Es werden mehrere Programme parallel ausgeführt, die nicht den gleichen Code enthalten müssen.

Diese Klassen können wiederum auf verschiedene Art in Hardware implementiert werden. Um Multicores einordnen zu können, werden hier die aktuell gebräuchlichen Varianten aufgelistet, wie mehrere Operationen parallel ausgeführt werden können:

- als SIMD-Operationen innerhalb einer Funktionseinheit (z.B. MMX, SSE)
- in mehreren SPMD-Threads innerhalb eines Kerns
- in mehreren MPMD-Threads, wobei die Threads hierbei verschieden "nah" beieinander ausgeführt werden können:
  - innerhalb eines Kerns (Hardware-Threads)
  - in verschiedenen Kernen eines Prozessors (Multicore-Prozessoren)
  - in verschiedenen Prozessoren (Multiprozessor-Systeme)

Es gibt einige Gründe, auch die anderen Varianten in einer Arbeit über Multicores kurz zu betrachten. So kombinieren aktuelle Prozessoren fast immer mehrere davon, und ihre Nutzung ist nötig um die volle Leistungsfähigkeit ausschöpfen zu können. Teilweise muss der Programmierer in seinem Programm diese unterschiedlich ansprechen, teilweise versteckt dies das Betriebssystem vor ihm. So weiß er im Normalfall z.B. gar nicht, ob sein Programm auf zwei Kernen desselben Prozessors oder auf zwei unterschiedlichen Prozessoren läuft. Dabei kann dies jedoch deutliche Unterschiede in der Performance machen. Ein weiterer Grund ist, dass bei den noch recht neuen Multicores allein mit mehreren Kernen derzeit keine hohe Parallelität erreichbar ist. So kann es sinnvoll sein, auf andere Varianten mit höherer Parallelität auszuweichen. Diese sind jedoch nicht immer für alle Aufgabenstellungen nutzbar.

Im folgenden werden also kurz SPMD- und Hardware-Threads und ihre Unterschiede zu Multicores näher erläutert. SIMD wird nicht weiter besprochen, da es sich im Gegensatz zu den anderen auf einzelne Instruktionen beschränkt. Multiprozessor-Systeme sind bereits deutlich älter und verbreitet, daher wird hier ein größeres vorhandenes Wissen angenommen.

1.3.1 Single Program Multiple Data (SPMD)

Bei dieser erst seit kurzem verbreiteten Technologie wird ein Thread, also eine Code-Sequenz, gleichzeitig mehrfach mit verschiedenen Daten ausgeführt. Dazu müssen alle wichtigen Teile eines Kerns wie Funktionseinheiten und Register entsprechende oft vorhanden sein. Die einzige Ausnahme ist die Instruktionseinheit mit Instruktionszähler und Code-Cache, die nur einmal für alle Threads zusammen existiert. Dies verringert die nötigen Kosten, schränkt die Verwendung aber auf Aufgabenstellungen mit reiner Datenparallelität ein. SPMD-Einheiten werden aktuell hauptsächlich in Grafikkarten und

für Stream-Computing gedachten Anwendungsbeschleunigern verbaut, wobei zwischen 2 und 96[1] Threads parallel ausgeführt werden.

1.3.2 Hardware-Threads

Hardware-Threads sind eine andere Variante der Parallelität, die hauptsächlich unter dem von Intel eingeführten Begriff "Hyper-Threading" bekannt ist. Dabei führt ein Prozessorkern mehrere unterschiedliche Threads parallel aus, es sind also Register und Instruktionseinheit pro Thread nötig. Dafür teilen sich die Threads die eigentlichen Funktionseinheiten. So kann z.B. ein Thread eine Ganzzahl-Berechnung und ein anderer eine Gleitkomma-Berechnung parallel durchführen, nicht jedoch beide eine Berechnung des selben Typs. So werden die vorhandenen Recheneinheiten besser ausgelastet, vor allem wenn ein Thread länger auf einen Speicherzugriff warten muss. Da nur einige Hilfseinheiten mehrfach vorhanden sein müssen, ist der Aufwand deutlich geringer als für einen Multicore, dafür ist der Performance-Gewinn durchschnittlich nur ca. 33 % groß [hyp03]. Da mit einer größeren Anzahl die Auslastung der Recheneinheiten praktisch nicht mehr zunimmt, beschränkt sich hier die Parallelität auf zwei bis vier Threads pro Kern.

Im Gegensatz zu SPMD-Threads sind Hardware-Threads für alle Aufgaben benutzbar und werden deshalb in CPUs eingesetzt. Dabei stellen sich für das Betriebssystem und für die Anwendungen mehrere Threads genauso dar wie mehrere Kerne, es gibt allerdings Spezialbefehle um die genaue Topologie herauszufinden. Aktuelle Betriebssysteme benutzen diese Informationen in ihrem Scheduler um Prozesse optimal zu verteilen.

1.4 Multicores

Multicores sind deutlich leistungsfähiger als SPMD und Hardware-Threads, da nur sie es erlauben mehrere komplett unabhängige Threads in einem Prozessor auszuführen. Meistens können sie betrachtet werden, wie als ob mehrere herkömmliche Singlecores eines Multiprozessor-Systems in einem Chip integriert wären. Dies bedeutet, dass fast alle Einheiten eines Prozessors für jeden Kern separat vorhanden sein müssen. Dies schließt hauptsächlich die Berechnungseinheiten, die Register und die Instruktionseinheit ein. Gemeinsam haben die Kerne hauptsächlich die Anbindung nach außen zu I/O und Speicher und eventuell einen Cache. Zusätzlich braucht es noch eine Einheit des Prozessors, der die Kommunikation zwischen den Teilen wie Kernen, Cache, Anbindung

[1] z.B. im ClearSpeed CSX700 (`http://www.clearspeed.com/products/csx700.php`)

etc. übernimmt. Je höher die Anzahl der Kerne und je höher die Anforderungen an eine schnelle Kommunikation, desto größer wird dieser Teil. Falls weder die Umgebungsbedingungen wie Speichergeschwindigkeit der limitierende Faktor eines Programms sind, noch die Kommunikation zum Flaschenhals wird und das Problem sich perfekt parallelisieren lässt, ist bei Multicores ein linearer Speedup möglich. Dieser Fall wird jedoch nur selten erreicht [HM08]. Die am weitesten verbreiteten Multicores heute haben zwei bis acht Kerne (z.B. der AMD Opteron, der Intel Core oder der Sun Niagara 2). Es existieren jedoch bereits Prozessoren mit 64 Kernen[2]. In Spezialgebieten wie programmierbaren DSPs sind bereits mehrere hundert Kerne üblich[3]. Ankündigungen wie der Intel Larrabee (s. Abschnitt 3.2.1) zeigen, dass die Anzahl der Kerne in der nächsten Zeit weiter steigen wird.

[2] Tilera TILEPro64 (`http://www.tilera.com/products/TILEPro64.php`)

[3] z.B. von picoChip erhältlich (`http://www.picochip.com/products_and_technology/picoarray_architecture`)

2 Architekturen und Technologien

2.1 Homogene vs. heterogene Multicores

Fast alle aktuellen Multicore-Prozessoren sind homogen, d.h. alle Kerne sind zumindest logisch identisch. Eventuelle Unterschiede auf der physikalischen Seite, z.B. beim Layout der Einheiten auf dem Prozessor-Die, werden hier nicht betrachtet. Es besteht aber auch die Möglichkeit, unterschiedliche Kerne in einem Chip unterzubringen. Diese können z.B. in einigen Spezialfähigkeiten variieren, es können aber auch mehrere verschiedene Architekturen in einem Prozessor vereint werden. Der Cell-Prozessor (s. Abschnitt 3.1.3) mit einem PowerPC-Kern und mehreren eigens entwickelten Stream-Prozessing-Kernen ist ein Beispiel für letzteres. Er ist vergleichbar mit der Integration einer klassischen CPU und einer auf Grafikerzeugung spezialisierten GPU innerhalb eines Chips.

Eine andere Architektur von heterogenen Multicores ist derzeit noch nicht am Markt, aber zumindest bei Intel in der Entwicklung [HBK06]. Dabei handelt es sich um einen Prozessor mit vielen Kernen, von denen die meisten für General-Purpose-Computing gedacht sind. Einige wenige sollen jedoch für Spezialaufgaben wie Ver- und Entschlüsselung oder Video-(De-)Kodierung optimiert werden.

Ein Vorteil von heterogenen Multicores ist, dass Kerne, die für eine Aufgabe spezialisiert sind, diese meist schneller und energieeffizienter erledigen als ein nicht-spezialisierter Kern. Dafür können andere Berechnungen meist nur langsam oder gar nicht ausgeführt werden. Auch muss die ausgeführte Software diese Spezialkerne explizit nutzen, wofür ein erhöhter Aufwand bei der Entwicklung nötig ist. Nicht-optimierte Programme oder solche, die diese Kerne nicht brauchen können, lassen Ressourcen des Prozessors brach liegen. Heterogene Multicores sind also sowohl in der Entwicklung der Hardware, als auch der Software aufwändiger. Einen Vorteil bieten sie jedoch, wenn dadurch mehrere sonst separate Chips zusammengefasst werden. Die PlayStation 3 etwa benötigt mit dem Cell-Prozessor nur noch einen Chip statt wie ein PC mit CPU und GPU derer zwei.

2.2 Kommunikation

Im Gegensatz zu Multiprozessor-Systemen, die heute in den allermeisten Fällen als symmetrische Multiprozessor-Architektur (SMP) mit identischen Prozessoren, globalem Speicher und ohne Kommunikation realisiert werden, gibt es bei Multicores eine Reihe von unterschiedlichen Architekturen mit und ohne Kommunikation. Die einfachste Variante dabei verpackt einfach mehrere SMP-Prozessoren in einen Chip. Diese können alle auf denselben Speicher zugreifen, dafür gibt es jedoch keine Kommunikation der Kerne untereinander. Ein jeder Informationsaustausch muss daher über den meist deutlich langsameren Hauptspeicher erfolgen. Um dies zu beschleunigen, enthalten viele Multicores einen Cache. Falls dieser auf die einzelnen Kerne aufgeteilt ist, kann der Fall eintreten, dass mehrere Kerne denselben Speicherbereich in ihren lokalen Cache laden wollen. Findet dann mindestens ein Schreibzugriff auf diesen Bereich statt, muss die Kohärenz gewährleistet werden [DB82]. Dazu müssen die anderen Kerne mitgeteilt bekommen, dass ihre Cache-Kopie des Speichers veraltet ist. Es findet also eine Kommunikation zwischen den Kernen statt, allerdings kann diese genau wie die restliche Cache-Benutzung weder vom Betriebssystem, noch von der Anwendung in irgendeiner Art und Weise gesteuert oder kontrolliert werden. Dies ist die am weitesten verbreitete Variante unter aktuellen Multicores, z.B. vorhanden in den aktuellen Dual- und Quad-Cores von AMD und Intel. Eine Architektur mit lokalem Speicher und durch die Anwendung gesteuertem Nachrichtenversand zwischen den Kernen ist theoretisch möglich, jedoch in keinem Multicore implementiert.

Der Cell-Prozessor von IBM, Sony und Toshiba weist stattdessen eine Art Mischform mit Kommunikation und globalem Speicher auf. Seine Architektur enthält sowohl einen großen globalen Speicher, als auch kleinere lokale Speicher in jedem der Kerne. Diese können weder explizit Nachrichten an andere versenden, noch direkt auf den globalen Speicher oder den Speicher eines anderen Kerns zugreifen. Für einen nicht-lokalen Speicherzugriff müssen sie die entsprechenden Daten per DMA-Transfer in den lokalen Speicher kopieren. Eine Ausnahme davon bildet der eine PowerPC-Kern, der zwar keinen eigenen lokalen Speicher besitzt, aber direkt auf denjenigen der anderen Kerne und den globalen Speicher zugreifen kann. Auch hier gibt es zudem Kommunikation zwischen den Kernen zur Einhaltung der Kohärenz der per DMA kopierten Daten.

2.3 Topologie

Sobald mehr als 2 Kerne in einem Prozessor vorhanden sind, stellt sich die Frage, wie die Kommunikationsverbindungen angeordnet werden. Eine direkte 1:1-Verbindung von jedem Kern zu jedem anderen Kern kommt bereits bei sehr kleinen Kernzahlen aufgrund des exponentiellen Wachstums nicht in Frage. Daher werden meistens die von Multiprozessor-Architekturen und von Clustern bekannten Kommunikationsmuster übernommen (s. [Qui94]).

Die einfachste derartige Topologie ist die lineare Verbindung aller Kerne zu einem Bus. Dieser hat den Vorteil, dass der Aufwand pro Kern konstant ist, da jeder Kern mit maximal 2 Partnern verbunden ist. Der mittlere Abstand zwischen 2 Kernen steigt jedoch mit steigender Anzahl linear an. Außerdem blockieren sich mehrere gleichzeitige Kommunikationen, wenn ihre Wege sich überlappen. Dennoch wird der Bus in Chips mit wenigen Kernen, wie z.B. dem Cell-Prozessor verwendet. Mit nur wenig Mehraufwand an Hardware lässt sich der Bus zu einem Ring ausbauen, was mehr gleichzeitige Kommunikation zulässt. Im Intel Larrabee (s. Abschnitt 3.2.1) wird ein bi-direktionaler Ring Verwendung finden, bei dem sich nach jeder Zeiteinheit die Kommunikationsrichtung ändert. So sinkt die mittlere Weglänge auf die Hälfte, ohne dass es zu sich kreuzenden Kommunikationen kommt.

Bei Prozessoren mit noch mehr Kernen werden diese meistens in einem 2-dimensionalen Grid angeordnet, z.B. im Intel Terascale [HBK06] und im 64-kernigen Tilera TILEPro64[4]. Dabei steigt der durchschnittliche Abstand zweier Kerne nur noch logarithmisch. Noch komplexere Topologien wie 3-dimensionale Gitter oder Butterfly-Netze finden derzeit in Multicores keine Verwendung. Ein Hauptgrund ist, dass die Konstruktion von 3-dimensionalen Strukturen auf einem Chip sehr teuer ist.

Eine verbreitete Alternative ist, den Cache nicht auf die Kerne aufzuteilen, sondern zusammen vorzuhalten, z.B. im Sun Niagara 2 (s. Abschnitt 3.1.1). Dann ist keine Kommunikation zur Cache-Synchronisation mehr nötig, dafür allerdings Kommunikation zwischen den Kernen und dem Cache. Um mehrere Zugriffe gleichzeitig zu erlauben, wird der Cache dann in mehrere Bänke aufgeteilt. Diese werden über einen Crossbar Switch, wie er z.B. auch für Ethernet benutzt wird, mit den Kernen verbunden. So können alle Kerne gleichzeitig auf den Cache zugreifen, wenn jeder eine andere Bank nutzt.

[4] `http://www.tilera.com/products/TILEPro64.php`

2.4 Speicher

Wie bereits im Abschnitt 2.2 beschrieben, dominieren außerhalb von Multicores SMP-Systeme mit einem globalen Speicher. Wenn immer mehr Threads pro Prozessor gleichzeitig ausgeführt werden, wird dessen Anbindung jedoch schnell zum Flaschenhals. Große Caches können dies zwar verbessern, benötigen jedoch viele Transistoren und eine komplexe Verwaltung.

2.4.1 Lokale Speicher

Daher haben hauptsächlich die für Stream-Computing gedachten Multicores wie GPUs stattdessen einen kleinen Speicher (auch "scratchpad" genannt) in jedem Kern. Dieser kann nur lokal angesprochen werden, sodass sich der Zugriff sehr einfach und schnell gestalten lässt. Oft ist der lokale Speicher in nur einem Prozessortakt erreichbar. Die Größe beträgt bei heutigen Multicores zwischen 16 und 256 Kilobyte pro Kern.

Im Gegensatz zu einem Cache muss die Anwendung dem Prozessor durch die Verwendung passender Befehle oder Speicheradressen explizit mitteilen, dass der lokale Speicher benutzt werden soll. Dies bedeutet, dass nicht darauf angepasste Programme keinen Vorteil davon ziehen und deutlich langsamer laufen werden. Die einfachere Hardware zieht also einen größeren Aufwand bei der Software-Entwicklung nach sich. Benötigen mehrere Kerne dieselben Daten, die aus Performance-Gründen im lokalen Speicher liegen müssen, muss sich der Entwickler manuell um die Einhaltung der Kohärenz und geeignete Synchronisationsmechanismen kümmern.

2.4.2 Cache

In den Prozessoren, in denen kein lokaler Speicher vorhanden ist, sorgen ein oder mehrere hierarchisch angeordnete Caches für einen schnelleren Speicherzugriff. In Multicores kann eine Cache-Einheit dabei für einen, mehrere oder alle Kerne zuständig sein. Die kleinste und schnellste Stufe, der Level 1-Cache, ist in aktuellen Chips normalerweise für jeden Kern separat vorhanden. Beim Level 2-Cache finden sich alle der genannten Möglichkeiten. Falls nicht ein gemeinsamer Level 2-Cache existiert, gibt es oft noch einen Level 3-Cache für alle Kerne zusammen. Die Caches einer Hierarchiestufe werden dabei von der Hardware garantiert kohärent gehalten. Es wird also dafür gesorgt, dass bei einer Modifikation eines Speicherbereichs alle Kopien desselben Speicherbereichs aus den Caches der anderen Kerne gelöscht werden.

2.5 Zusätzliche Hilfsmittel in Hardware

Mit dem immer weiter steigenden Grad der Parallelität, steigt auch der Aufwand der zur ihrer Verwaltung vom Software-Entwickler getrieben werden muss. Daher bieten einige Chips zusätzliche Hilfen, die dies vereinfachen sollen.

Weit verbreitet und auch schon in Singlecore-Prozessoren zu finden sind atomare Funktionen, die mehrere Operationen auf einem Speicherbereich auf einmal ausführen. Dabei ist von der Hardware garantiert, dass kein anderer Zugriff auf diesen Wert während der Ausführung der gesamten Funktion stattfindet. Ohne dies könnten z.B. Probleme wie Lost-Updates entstehen. Gängige atomare Funktionen sind z.B. die Integer-Addition (bestehend aus Lesezugriff, Berechnung und Schreibzugriff) und die Compare-and-Swap-Operation. Diese vergleicht einen Wert im Speicher mit einem anderen und überschreibt bei Gleichheit ersteren mit einem neuen Wert.

Vor allem in SPMD-Einheiten zu finden ist eine Hardware-Unterstützung für Barrier-Synchronisation. Diese stellt sicher, dass alle Threads einer Gruppe aufeinander warten und dann gleichzeitig mit der weiteren Ausführung des Programms fortfahren. In SPMD-Gruppen ist z.B. dies nötig, nachdem die Threads unterschiedlich aufwändige Operationen ausgeführt haben. Ist die Synchronisation in Hardware implementiert, kann sie deutlich schneller sein als in Software, etwa in Nvidias GPUs, die nur einen Taktzyklus dafür benötigen.

Derzeit noch in keinem verfügbaren Prozessor enthalten, aber für den Ende 2009 erwarteten Rock-Prozessor von Sun angekündigt (s. Abschnitt 3.2.2), ist eine Implementierung von Transaktionsspeicher [HCU+07]. Dieser bietet ähnlich wie transaktionsfähige Datenbanken die Möglichkeit, mehrere Operationen zu gruppieren, im Fehlerfall den vorherigen Zustand wiederherzustellen und gleichzeitigen Zugriff anderer Threads auf denselben Speicher zu verhindern. Solcher Speicher kann auch in Software simuliert werden, ist aber in Hardware ebenfalls deutlich schneller.

3 Multicore-Chips

Im Folgenden werden einige konkrete Multicores detaillierter vorgestellt. Es wurden fünf Prozessoren ausgewählt, die derzeit oder in naher Zukunft auf dem Markt verfügbar sind. Alle besitzen außerdem mindestens ein Alleinstellungsmerkmal, dass sie interessant genug macht, um einen genaueren Blick auf sie zu werfen. Der Sun Niagara 2-Prozessor vertritt dabei die Klasse der konventionellen Multicores für General-Purpose-Computing, wie sie in aktuellen Desktops, Workstations und Servern eingesetzt werden. Optimiert für Stream-Computing und daher u.a. für Berechnungen von Echtzeit-Graphik eingesetzt werden der Cell-Prozessor und die GPUs von AMD und Nvidia. Letztere werden auf Grund ihrer ähnlichen Architektur zusammen betrachtet. Beide Bereiche verbinden soll der Larrabee von Intel, der Ende des Jahres 2009 erwartet wird. Sehr viele neue Funktionen wird der Sun Rock-Prozessor bieten, der ebenfalls noch dieses Jahr auf den Markt kommen soll.

Diese Auswahl stellt nur einen kleinen Teil der erhältlichen und geplanten Prozessoren dar. Es gibt einige Multicores, die aus Gründen wie zu wenig vorhandener Information, fehlender General-Purpose-Computing-Fähigkeit oder zu kleiner Verbreitung hier fehlen müssen. Dies umfasst z.B. den Cisco QuantumFlow-Prozessor[5], der nur in Cisco-Routern eingesetzt wird, die Klasse der Massively Parallel Processor Arrays (MPPA), die aus mehreren hundert programmierbaren DSP-Kernen bestehen [GR96], und spezielle Multicores für den Embedded-Bereich wie der ARM11 MPCore von ARM[6]. Die wohl bekanntesten Mehrkern-Prozessoren, die Dual- und Quad-Cores von AMD und Intel werden nicht näher erläutert, da sie nur sehr wenige Kerne enthalten und keine besonderen Merkmale besitzen, die sie für diese Übersicht besonders interessant machen würden. Der Intel Terascale [HBK06], der zwar konzeptuell interessant wäre, ist leider erst in einiger Zeit zu erwarten und derzeit sind nur sehr wenige konkrete Informationen verfügbar.

[5] http://newsroom.cisco.com/dlls/videos/asr_030408.html
[6] http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html

3.1 Existierende Chips

3.1.1 Sun UltraSPARC T2 (Niagara 2)

Der Niagara 2 [Gol06] (Foto s. Abbildung 1), offiziell UltraSPARC T2 genannt, ist Suns aktuelle CPU mit SPARC-Architektur. Er ist seit 2007 verfügbar und wird in größeren Servern eingesetzt. Um darin möglichst viele Client-Anfragen bearbeiten zu können, ist er vor allem auf einen großen Durchsatz optimiert und nicht auf eine hohe Performance pro Thread. Deshalb ist er auch der einzige Prozessor, dessen acht Kerne jeweils acht Hardware-Threads erlauben, um wirklich jede Wartezeit mit der Ausführung eines anderen Threads überbrücken zu können. Eine weitere Besonderheit ist, dass die Ganzzahl-Berechnungseinheit pro Kern doppelt vorhanden ist. So können zwei Hardware-Threads auch dann parallel laufen, wenn sie eine solche Berechnung ausführen und nicht nur, wenn sie unterschiedliche Berechnungen ausführen oder in unterschiedlichen Stufen der Befehls-Pipeline stecken. Diese Architektur benötigt sehr viel weniger Hardware-Aufwand als die doppelte Anzahl an herkömmlichen Kernen mit vier Hardware-Threads, kann jedoch laut Sun in typischen Server-Einsatzbereichen an deren Geschwindigkeit herankommen.

Der Cache des Niagara 2 besteht aus einem Level 1-Cache pro Kern und einem großen gemeinsamen Level 2-Cache. Dieser ist in acht Bänke unterteilt, wobei gleichzeitig auf jede Bank zugegriffen werden kann. Wie in Abbildung 2 gut zu erkennen, sind die Kerne mit dem Cache über einen Crossbar Switch verbunden, so dass parallele Zugriffe verschiedener Kerne auf verschiedene Bänke sich nicht stören. Ebenfalls auf diese Art angebunden ist die I/O-Einheit, um den für einen Server-Prozessor wichtigen schnellen Zugriff auf Netzwerkkarte und PCI-Express-Bus zu gewährleisten. Die Kohärenz des Level 1-Cache wird vom Level 2-Cache verwaltet, es findet also auch hierfür keine Kommunikation zwischen den Kernen statt.

3.1.2 GPGPU Chips von AMD und Nvidia

Traditionelle 3D-Grafikkarten, wie sie seit über zehn Jahren in Desktops und Workstations verbaut sind, waren strikt auf die ihnen zugedachte Aufgabe der Grafikerzeugung optimiert. Sie waren kaum programmierbar und erledigten die Berechnungen in fester Hardware-Logik, weshalb sie dabei deutlich schneller waren als herkömmliche CPUs. Mit den steigenden Anforderungen an die Grafik und daher an die Möglichkeiten der Hardware, steigt die Flexibilität der Chips jedoch deutlich. Seit ca. 2007 wird es von den

	ATI Radeon HD 4890	Nvidia GTX 285
SPMD-Einheiten (Kerne)	10	30
Threads pro Kerne	16	8
SIMD-Breite	5	1
Theoretisches Maximum an Parallelität	800	240

Tabelle 3.1: Parallelität in aktuellen GPUs

beiden Herstellern AMD (ehemals ATI) und Nvidia unterstützt, auch Programme zu anderen Zwecken als zur Berechnung von 3D-Grafik auf den Grafikkarten auszuführen. Dies wird unter dem Schlagwort General Purpose GPU (GPGPU) zusammengefasst. Die Möglichkeiten der Chips reichen zwar noch nicht an die von CPUs heran, durch die einfachere Architektur sind jedoch deutlich mehr Kerne möglich. Die Leistung der Chips bei darauf spezialisierten Programmen ist daher um bis zu eine Größenordnung höher als die aktueller CPUs.

Die aktuellen Multicore-GPUs von AMD [ati09] und Nvidia [nvi08] haben die gleiche Architektur, auch wenn sie von den Herstellern sehr unterschiedlich dargestellt werden (vergleiche Abbildungen 3 und 4). Beide bestehen aus SPMD-Einheiten (Kernen, bei Nvidia "Multiprocessor" genannt), die jeweils einige SPMD-Threads parallel auf ihren "(Thread) Processors" ausführen können. Durch die zusätzliche Verwendung von SIMD-Operationen, sind eine enorme Anzahl von gleichzeitigen Berechnungen möglich (s. Tabelle 3.1[7]). Jeder Kern enthält einige Kilobyte lokalen Speicher, auf den alle Threads der SPMD-Einheit zugreifen können (bei ATI fehlt dieser in der Grafik, ist jedoch vorhanden). Dieser Speicher ist sehr schnell, bei passenden Zugriffsmustern ist er in einem Prozessortakt erreichbar. Kommunikation zwischen den Kernen ist nur über den globalen Speicher möglich, für den kein Cache existiert. Divergiert die Ausführung der Threads einer SPMD-Gruppe (weil z.B. eine von der Thread-Nummer abhängige If-Klausel existiert), wird die Gruppe aufgespalten und die entstehenden Thread-Gruppen werden vom Chip abwechselnd ausgeführt. Die Anzahl der Gruppen, die ein Kern verwalten kann, ist jedoch beschränkt, so dass bei zu großer Diversität ein Programmabbruch erfolgt. Um dies zu vermeiden und parallel Ausführung zu erreichen lassen sich alle Threads wieder per Barrier-Synchronisation zusammenfassen. Für diese und für einige atomare Funktionen ist Hardware-Unterstützung vorhanden.

Aktuelle GPUs haben noch ein paar Einschränkungen gegenüber klassischen CPUs. Neben der Optimierung auf SPMD-Programme ist das Hauptproblem die fehlende Mög-

[7] Technische Daten von `http://ati.amd.com/products/radeonhd4800/specs-4890.html` und `http://www.nvidia.com/object/product_geforce_gtx_285_us.html`

lichkeit der dynamischen Allokation von Speicher. Dies hat zur Folge, dass keine Rekursion benutzt werden kann. Eine gute Performance kann außerdem nur erreicht werden, wenn genau auf die architekturellen Feinheiten des jeweiligen Chips Rücksicht genommen wird, was die Entwicklung deutlich verkomplizieren kann. Derzeit muss außerdem je nach Hersteller des Chips eine eigene Programmiersprache benutzt werden (AMD: Brook+, Nvidia: CUDA), so dass Kompatibilität weder auf Binärprogramm-, noch auf Source-Code-Ebene gegeben ist. Die OpenCL-Initiative arbeitet daran, eine einheitliche Programmier-Schnittstelle zur Verfügung zu stellen.

3.1.3 Cell Broadband Engine

Die Cell Broadband Engine [Sto05], [CRDI05] ist ein Multicore-Prozessor, der von IBM, Sony und Toshiba entwickelt wurde und seit 2006 verfügbar ist. Sein Haupteinsatzzweck sind Grafik-Berechnungen in der PlayStation 3 und Stream-Computing. Er enthält einen PowerPC-Kern mit zwei Hardware-Threads, genannt PowerPC Processing Element, und acht Synergistic Processing Elements (SPEs), auf die die rechenintensiven Teile der Programme ausgelagert werden können. Dadurch werden in einem Chip die hohe Rechenleistung für Spezialanwendungen, wie sie etwa in Grafikchips vorhanden ist, und die General-Purpose-Fähigkeiten und Kompatibilität zu vorhandener Software von herkömmlichen CPUs vereint.

Die Architektur (s. Abbildung 6) des Cell-Prozessors sieht vor, dass alle Kerne und der Speicher-Controller an einen linearen Bus angebunden sind, was das Chip-Layout vereinfacht. Dies wird besonders deutlich, wenn man in den Abbildungen 1 und 5 den Platz-Anteil auf dem Chip vergleicht, der nicht für die Kerne und den Cache zur Verfügung steht. Die SPEs enthalten jeweils einen lokalen Speicher, können jedoch nicht direkt auf den Hauptspeicher zugreifen. Jegliche von ihnen benutzte Daten müssen zuvor per DMA in den lokalen Speicher kopiert werden. DMA-Transfers sind auch zwischen den einzelnen Kernen möglich. Dies sorgt für eine große Flexibilität, ohne dass ein sehr viel größerer Aufwand in Hardware nötig ist, wie dies etwa bei einem freien direkten Zugriff auf jeden Speicherbereich nötig wäre. Im Gegensatz zu klassischen Datentransfers arbeitet DMA (Direct Memory Access) asynchron, ein Kern kann also weitere Befehle ausführen während ein Transfer läuft. Beim Cell-Prozessor garantiert ein spezielles kohärentes DMA-Verfahren die Einhaltung der Kohärenz zwischen allen vorhandenen Speichereinheiten. Der PowerPC-Kern kann direkt sowohl auf den Hauptspeicher als auch auf die lokalen Speicher der SPEs zugreifen.

3.2 Zukünftige Chips

3.2.1 Intel Larrabee

Eine andere Art der Verbindung von CPU und GPU, als sie der Cell-Prozessor darstellt, hat Intel mit dem Larrabee [SCS+08] angekündigt. Er soll gegen Ende des Jahres 2009 auf den Markt kommen. Auf Grund des Prototypenstatus sind noch nicht alle Informationen verfügbar (z.B. findet sich nur eine sehr grobe Skizzierung der Architektur wie Abbildung 7), aber es genügt um ihn hier in diesen Überblick aufzunehmen. Statt unterschiedliche Kerne für General-Purpose- und Stream-Computing wird er viele gleiche Kerne enthalten, die beides schnell erledigen sollen. Diese basieren auf dem Intel Pentium-Prozessor und sind zu dessen Software kompatibel. Initial sollen der Chip 32 Kerne enthalten, die über einen bi-direktionalen Ring miteinander kommunizieren. Dabei findet die Kommunikation nur statt, um die Teile des Level 2-Cache kohärent zu halten. Intel hat angekündigt, die Zahl der Kerne auf 48 oder noch mehr steigern zu wollen, die dann wahrscheinlich um mehrere miteinander verbundene Ringe angeordnet sind. Im Gegensatz zu aktuellen x86-Multicores sind die einzelnen Kerne dabei relativ einfach gehalten. Sie werden in einer reinen In-Order-Architektur ähnlich dem Atom-Prozessor für kleine Notebooks ausgeführt sein. Um für Stream-Computing geeignet zu sein, werden sie zusätzlich sehr große SIMD-Einheiten enthalten, die 16 Werte gleichzeitig verarbeiten kann. Dabei wird es Hardware-Unterstützung für Scatter- und Gather-Operationen geben, d.h. die 16 Werte für eine SIMD-Operation müssen nicht hintereinander im Speicher liegen, um ohne Geschwindigkeitseinbußen geladen werden zu können. Die erste Version des Larrabee, die primär als Grafikprozessor eingesetzt werden wird, soll eine Hardware-Einheit enthalten, die Texturberechnungen durchführt, da diese derzeit in Software noch deutlich langsamer sind. Es sollen jedoch auch andere Konfigurationen des Chips ohne diese Einheit oder vielleicht sogar mit anderen Spezialbeschleunigern erscheinen. Diese werden einfach an den Kommunikations-Ring angeschlossen und sind so von allen Kernen nutzbar.

3.2.2 Sun Rock

Ebenfalls noch nicht verfügbar, aber für Ende des Jahres 2009 angekündigt ist der Rock-Prozessor von Sun [TC08], [KRL+08], [Cha08]. Seine SPARC-Architektur wird einige Neuerungen bieten (s. Abbildung 8). Mit 16 Kernen enthält er doppelt so viele Kerne wie aktuelle Chips, die erstmals in Gruppen zu vier Kernen aufgeteilt sind. Innerhalb

einer Gruppe teilen sich die Kerne sogar den Level 1-Cache. Der Level 2-Cache wird wie beim Niagara 2 in Bänke aufgeteilt und über einen Crossbar Switch an die Kern-Gruppen angebunden. Damit ist der Rock besonders für SPMD-ähnliche Anwendungen geeignet, wenn also ein Programm oder Code-Abschnitt mehrfach parallel ausgeführt wird. Auch Hardware-Threads werden in einer neuen Variante enthalten sein. Zusätzlich zu zwei herkömmlichen Threads wird jeder Kern zwei "Scout"-Threads ausführen können. Dies sind Kopien existierender Threads, die in der Ausführung voran eilen, wenn der Thread auf die Ausführung einer langwierigen Operation wie einen Speicherzugriff warten muss. Etwas ähnliches, jedoch weniger leistungsfähiges, ist auch unter dem Begriff "Branch prediction" [Smi98] in aktuellen CPUs von AMD und Intel zu finden. Die interessanteste Neuerung ist jedoch wohl die Implementierung von Hardware-Transaktionsspeicher [HCU+07]. Details sind auch hier noch nicht bekannt, nur dass es über das Sperren von Cache-Inhalten für andere Kerne gehen soll.

4 Zusammenfassung

Als Fazit lässt sich aus dieser Arbeit ziehen, dass die Unterschiede zwischen aktuellen Multicores enorm groß sind. Keiner der fünf näher vorgestellten Prozessoren gleicht einem anderen. Beachtet man außerdem zusätzlich die Masse an hier nicht oder nur kurz erwähnten Chips, ergibt sich eine unüberschaubare Vielfalt. Dies stellt insbesondere Software-Entwickler vor zwei Probleme: Zum einen muss für ein gegebenes Projekt die am Besten geeignete Hardware ausgewählt werden, da eine falsche Wahl beträchtliche Geschwindigkeitseinbußen mit sich bringen kann. Die Architektur der Hardware bestimmt außerdem zu einem großen Teil den Aufwand, der bei der Entwicklung der darauf laufenden Software getrieben werden muss. Zum anderen benötigt der Entwickler ein profundes Wissen über die Feinheiten des benutzten Prozessors, da gerade bei den Leistungsfähigeren eine gute Anpassung der Software an die Hardware nötig für eine gute Performance ist. Dies erschwert die durch fehlende direkte Code-Kompatibilität beeinträchtigte Portierung von Programmen von einer Architektur zur anderen weiter.

Es ist also zu hoffen, dass diese Probleme in der nächsten Zeit verstärkt in Angriff genommen werden, z.B. durch die Entwicklung von Abstraktionsschichten, die Parallelität in Hardware einfach und schnell für Software-Entwickler benutzbar machen.

A Bildmaterial

Abbildungsverzeichnis

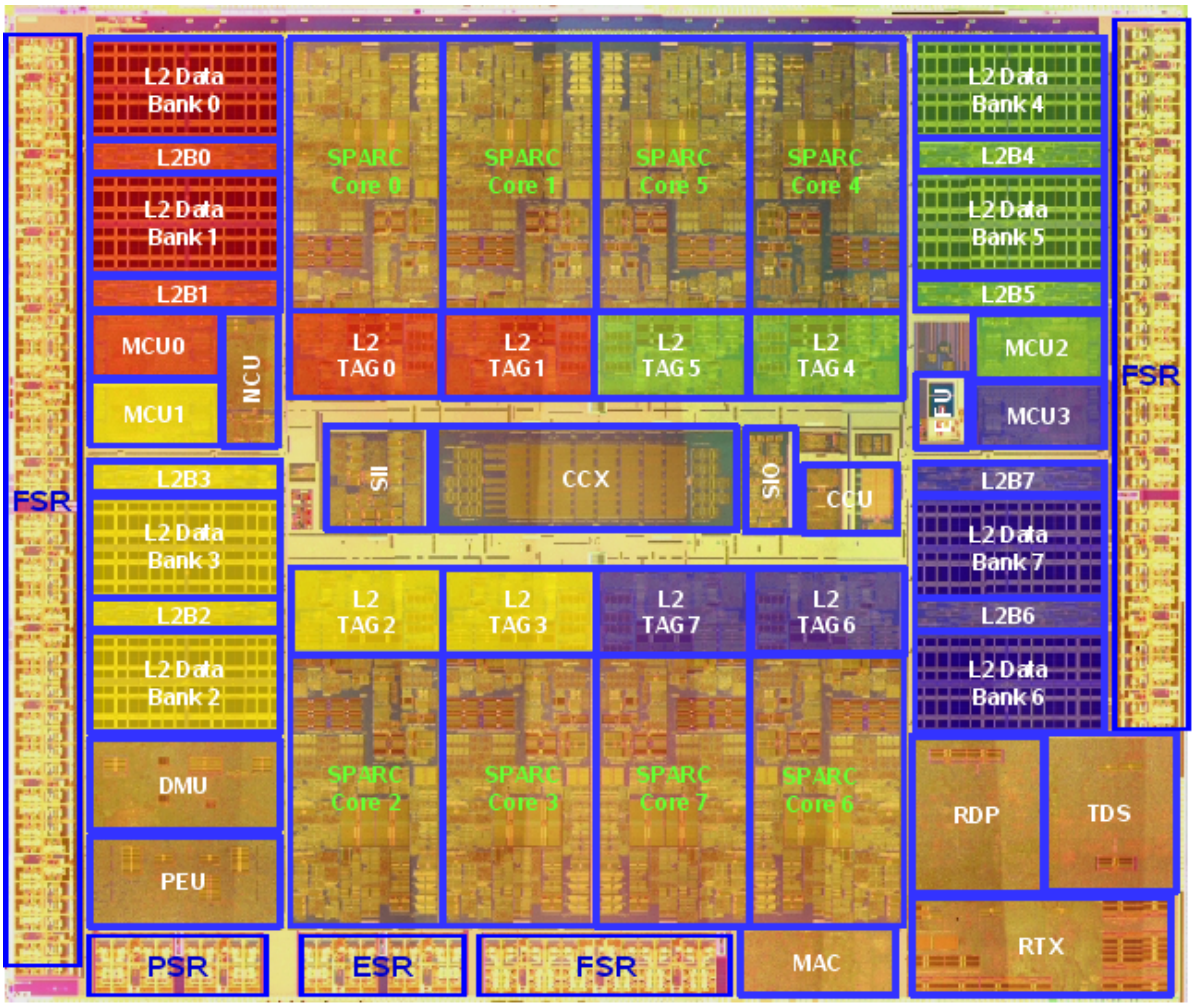

Abbildung 1: Sun UltraSPARC T2 (`http://www.opensparc.net`)

Niagara2 Chip Overview

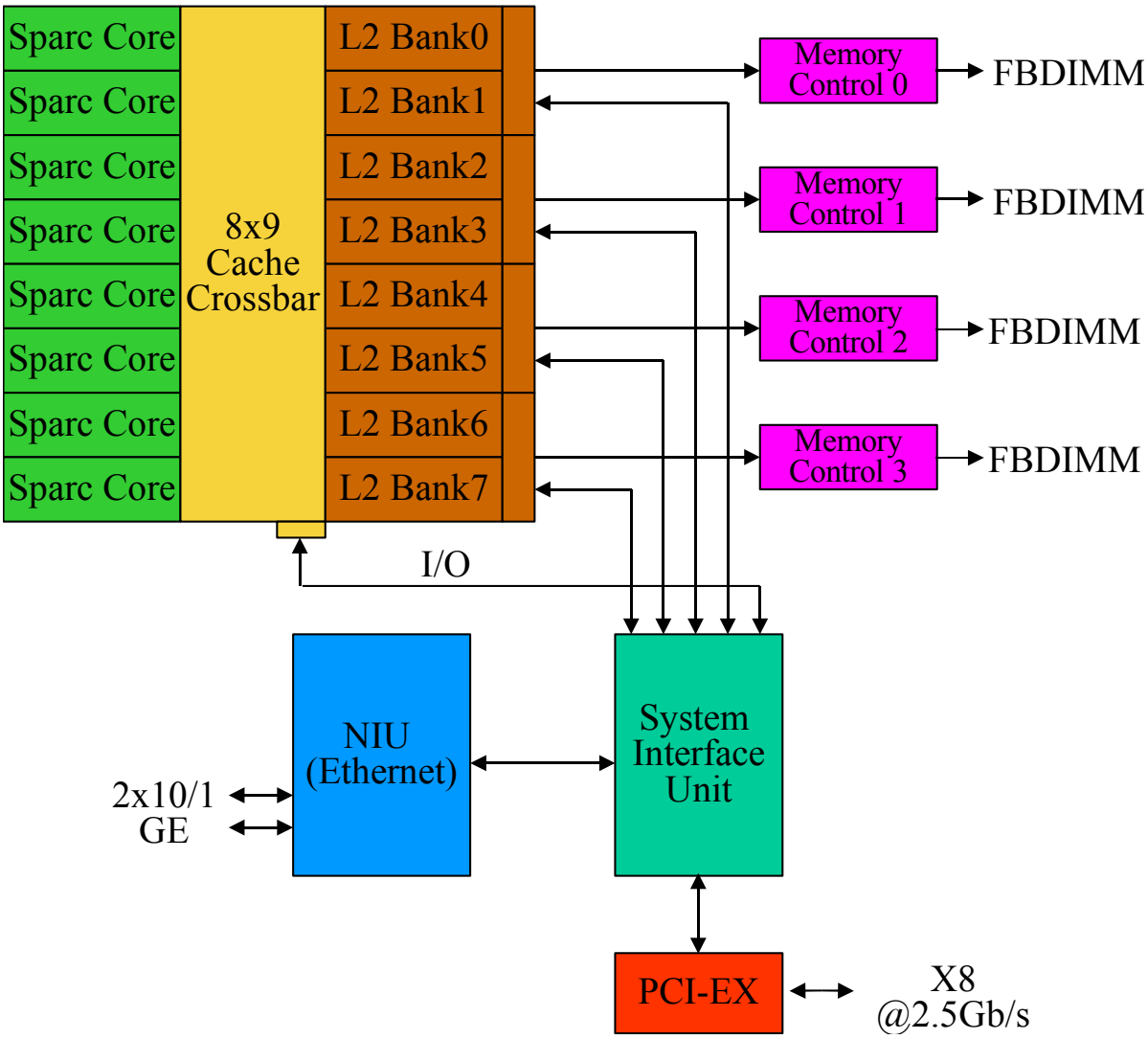

Abbildung 2: Architektur des Sun UltraSPARC T2 [Gol06]

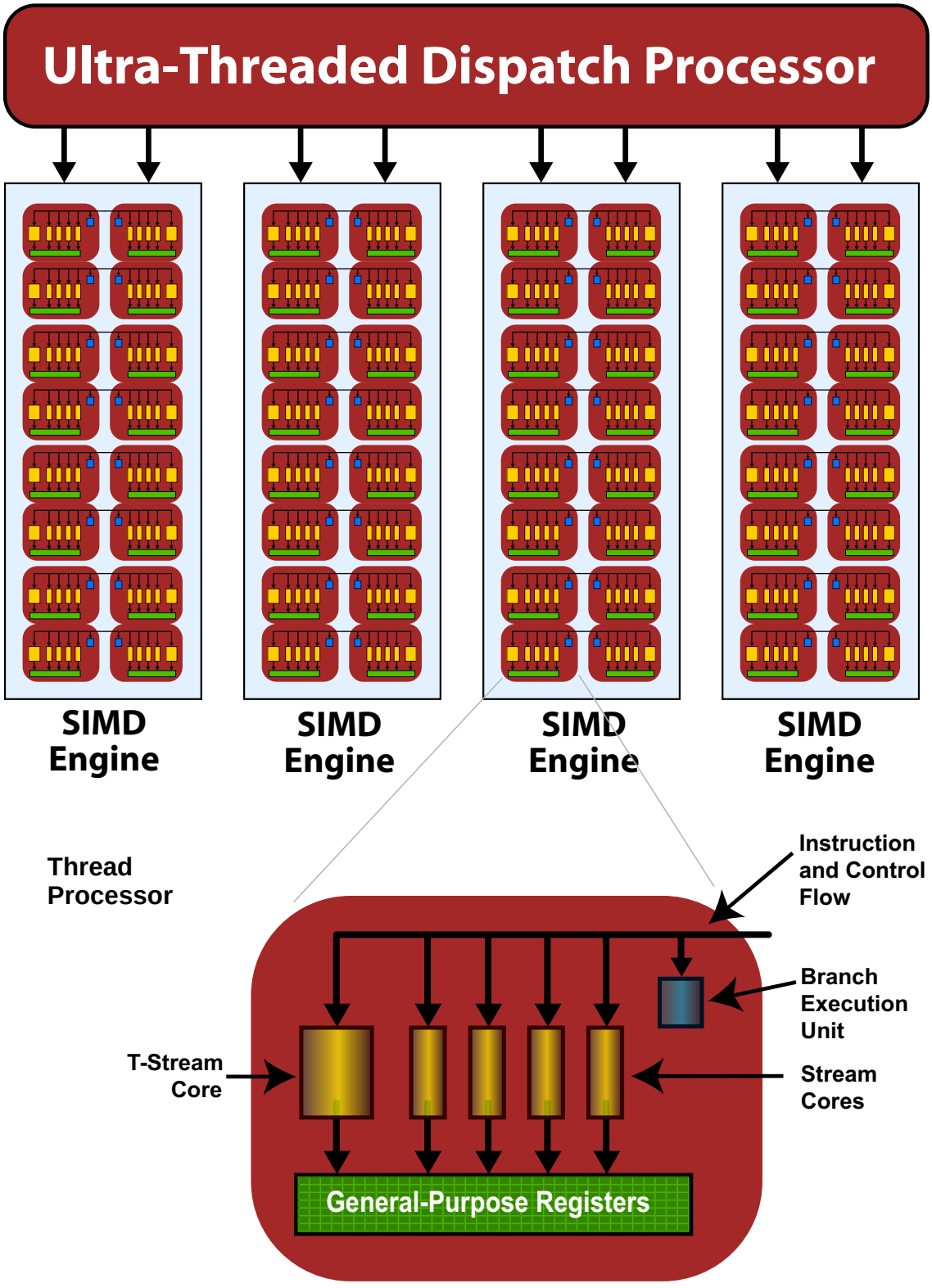

Abbildung 3: Architektur des ATI Stream-Prozessor [ati09]

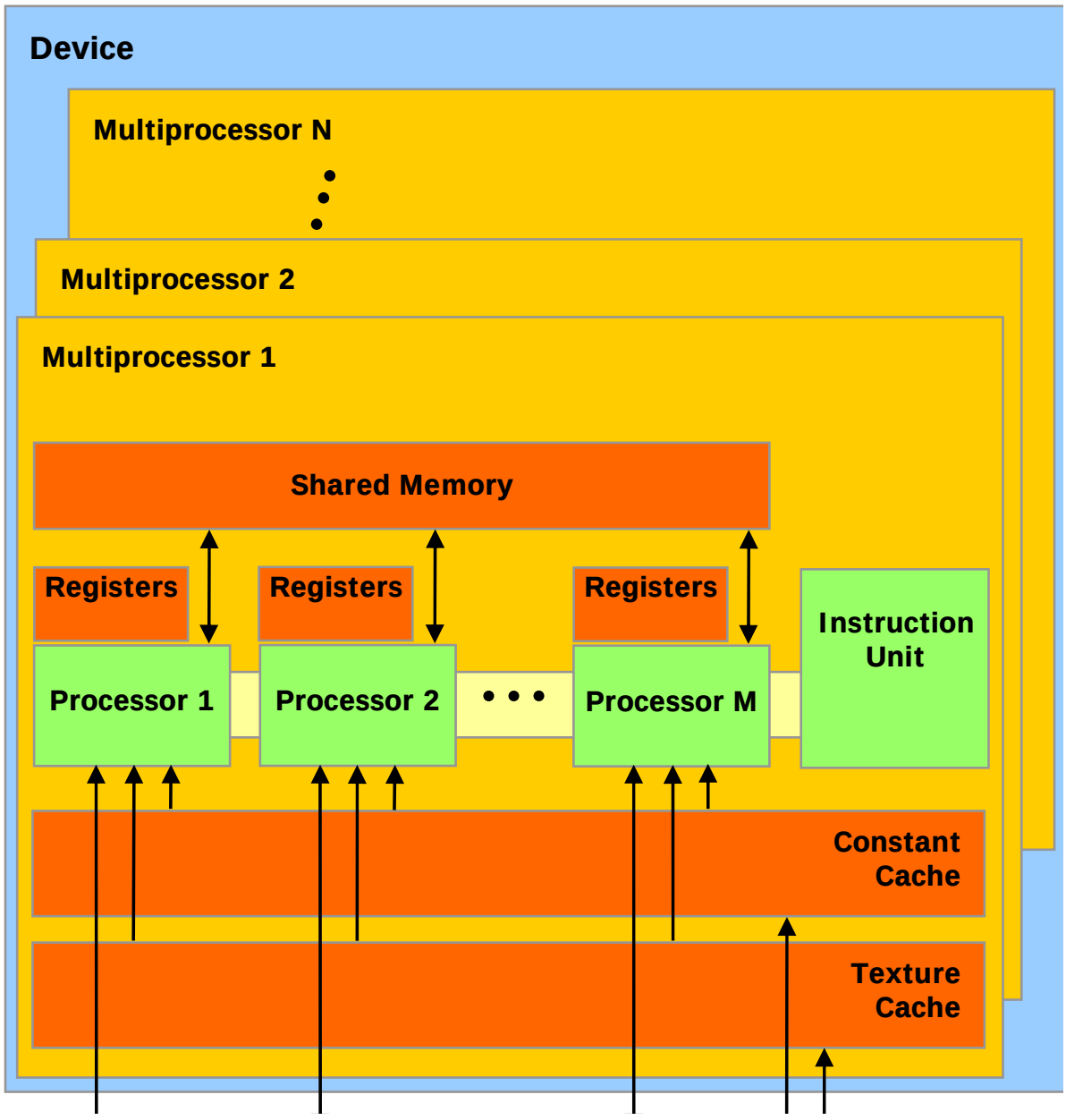

Abbildung 4: Architektur der Nvidia GPUs [nvi08]

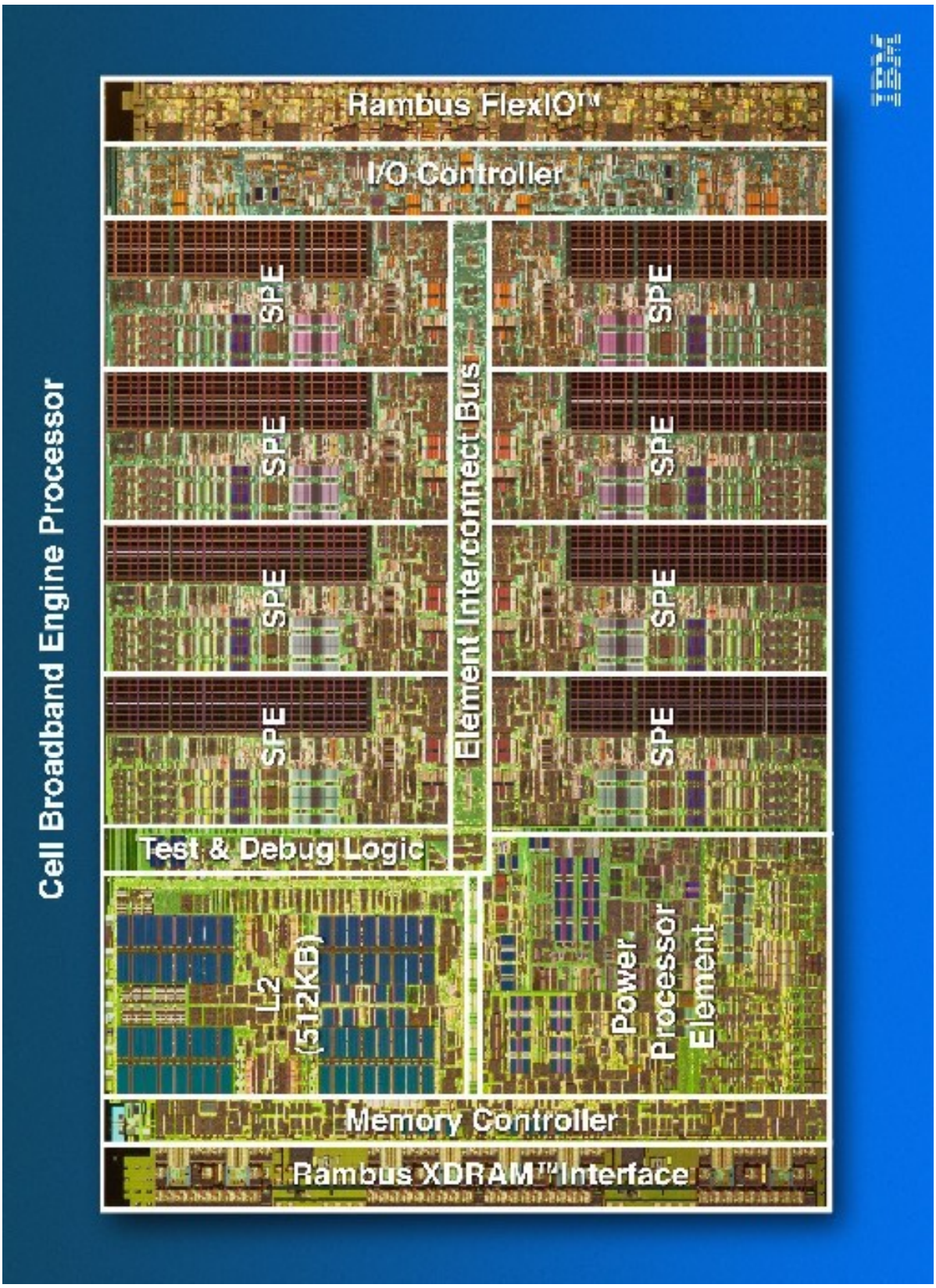

Abbildung 5: IBM Cell-Prozessor
(`http://www.research.ibm.com/cell/cell_chip.html`)

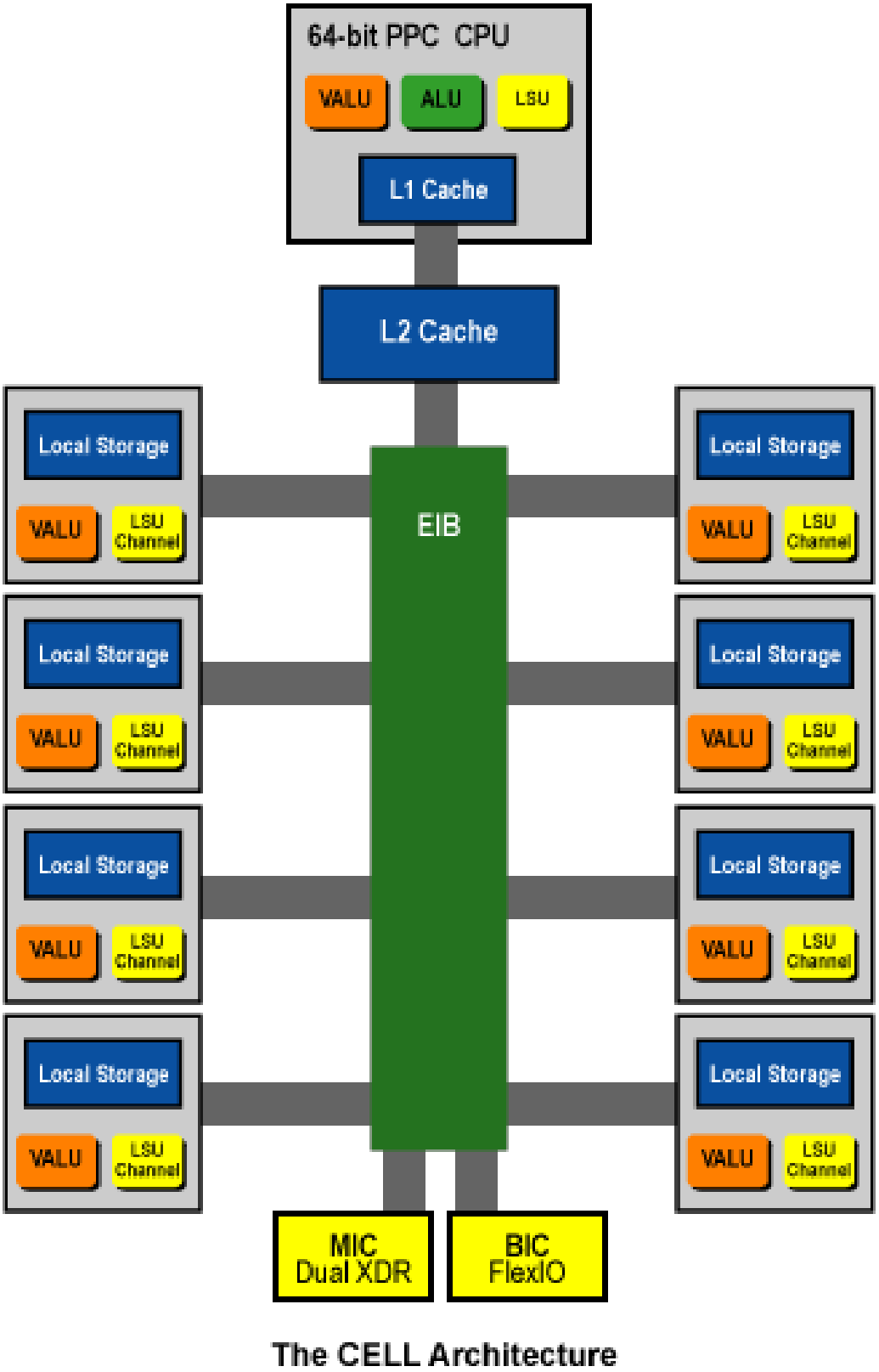

Abbildung 6: Architektur des IBM Cell-Prozessor [Sto05]
(man beachte die Ähnlichkeit zum tatsächlichen Chip-Layout)

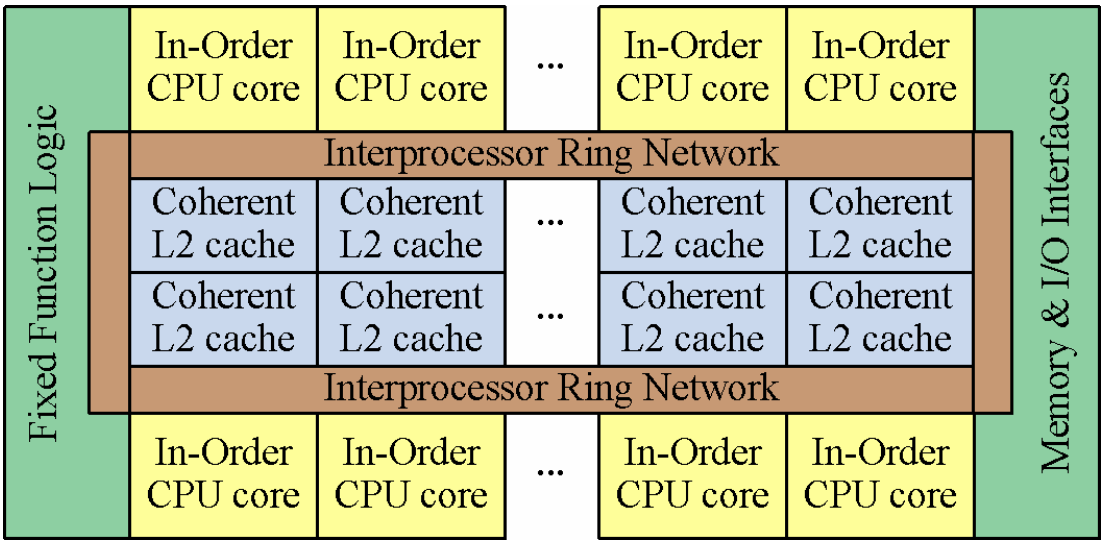

Abbildung 7: Architektur des Intel Larrabee [SCS+08]

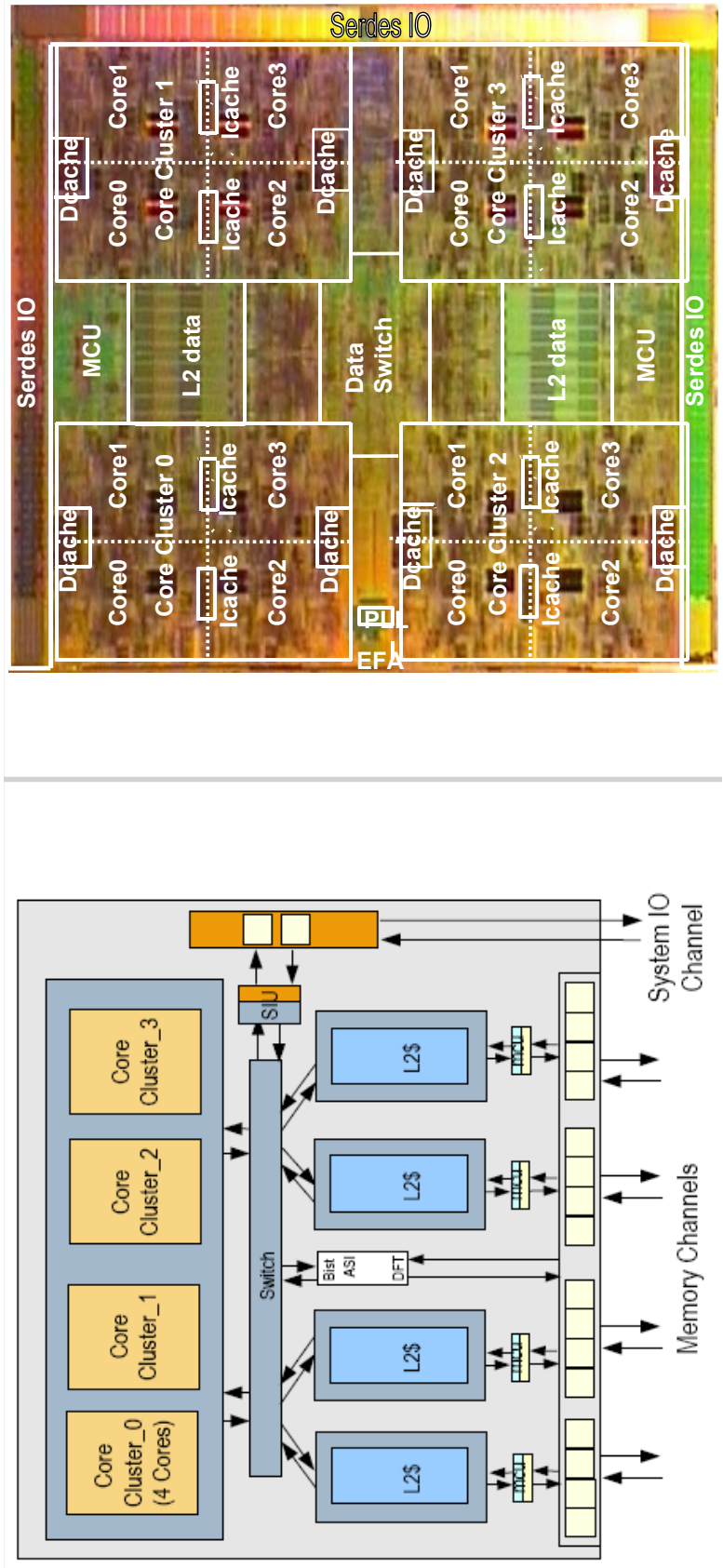

Abbildung 8: Sun Rock-Prozessor [TC08]

Literaturverzeichnis

[ati09] *ATI Stream Computing - Technical Overview*, March 2009. Online verfügbar unter `http://developer.amd.com/gpu_assets/Stream_Computing_Overview.pdf`.

[Cha08] Shailender Chaudhry. *Rock: A SPARC CMT Processor.* http://www.opensparc.net/pubs/preszo/08/RockHotChips.pdf, August 2008.

[CRDI05] Thomas Chen, Ram Raghavan, Jason Dale, und Eiji Iwata. *Cell Broadband Engine Architecture*, November 2005. Online verfügbar unter `http://www.ibm.com/developerworks/power/library/pa-cellperf/`.

[DB82] M. Dubois und F. A. Briggs. *Effects of Cache Coherency in Multiprocessors.* Computers, IEEE Transactions on, C-31(11):1083–1099, 1982.

[Fly72] Michael J. Flynn. *Some Computer Organizations and Their Effectiveness.* IEEE Transactions on Computers, C-21(9):948–960, September 1972.

[Gol06] Robert Golla. *Niagara2: A Highly Threaded Server-on-a-Chip*, October 2006. Online verfügbar unter `http://www.opensparc.net/pubs/preszo/06/04-Sun-Golla.pdf`.

[GR96] Bo Gao und D. J. Rees. *ASAP: an asynchronous array processor for hardware-software coprocessing and codesign.* In ASIC, 1996., 2nd International Conference on, pages 151–154, October 1996.

[HBK06] Jim Held, Jerry Bautista, und Sean Koehl. *From a Few Cores to Many: A Tera-scale Computing Research Overview*, 2006. Online verfügbar unter `http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf`.

[HCU+07] T. Harris, A. Cristal, O. S. Unsal, E. Ayguade, F. Gagliardi, B. Smith, und M. Valero. *Transactional Memory: An Overview.* Micro, IEEE, 27(3):8–29, 2007.

[HM08] M. D. Hill und M. R. Marty. *Amdahl's Law in the Multicore Era.* Computer, 41(7):33–38, July 2008.

[hyp03] *Intel Hyper-Threading Technology – Technical User's Guide*, January 2003. Online verfügbar unter `http://cache-www.intel.com/cd/00/00/01/77/17705_htt_user_guide.pdf`.

[KRL+08] G. Konstadinidis, M. Rashid, P. F. Lai, Y. Otaguro, Y. Orginos, S. Parampalli, M. Steigerwald, S. Gundala, R. Pyapali, L. Rarick, I. Elkin, Yuefei Ge, und I. Parulkar. *Implementation of a Third-Generation 16-Core 32-Thread Chip-Multithreading SPARC Processor.* In Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International, pages 84–597, February 2008.

[nvi08] *NVIDIA CUDA Compute Unified Device Architecture 2.0*, July 2008. Online verfügbar unter `http://www.nvidia.com/object/cuda_develop.html`.

[Qui94] Michael J. Quinn. *Parallel Computing: Theory and Practice.* McGraw-Hill, 2 edition, 1994.

[SCS+08] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, und Pat Hanrahan. *Larrabee: a many-core x86 architecture for visual computing.* ACM Trans. Graph., 27(3):1–15, 2008.

[Smi98] James E. Smith. *A study of branch prediction strategies.* In ISCA '98: 25 years of the international symposia on Computer architecture (selected papers), pages 202–215, New York, NY, USA, 1998. ACM.

[Sto05] Jon Stokes. *Introducing the IBM/Sony/Toshiba Cell Processor – Part II: The Cell Architecture*, February 2005. Online verfügbar unter `http://arstechnica.com/old/content/2005/02/cell-2.ars`.

[TC08] M. Tremblay und Shailender Chaudhry. *A Third-Generation 65nm 16-Core 32-Thread Plus 32-Scout-Thread CMT SPARC Processor*. In Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International, pages 82–83, February 2008.