TECHNISCHE UNIVERSITÄT DRESDEN

Institut für Technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur, Prof. Spallek

Implementierung und Evaluierung eines Multi-Core-Systems auf Basis der Java-Plattform SHAP

Tobias Berndt

Dresden, 29.07.2009

Inhalt

1. Aufgabe

2. Motivation und bisherige SHAP-Kopplung

3. Konzeptvorstellung für den SHAP
   - 3.1 Kopplung über gemeinsamen Startup Memory
   - 3.2 Kopplung über einfachen FIFO

4. Implementierung
   - 4.1 Implementierung der Architektur
   - 4.2 Synchronisation zwischen den Cores

5. Benchmarks

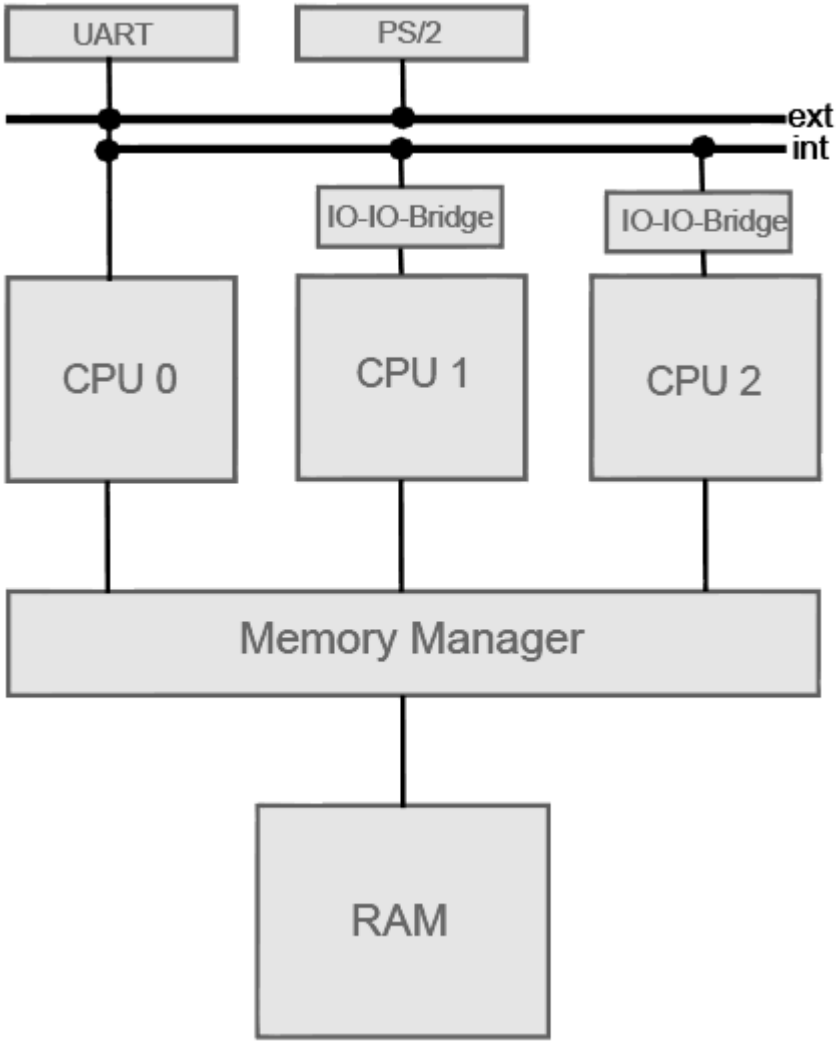
6. Auswertung

7. Zusammenfassung

1. Aufgabenstellung

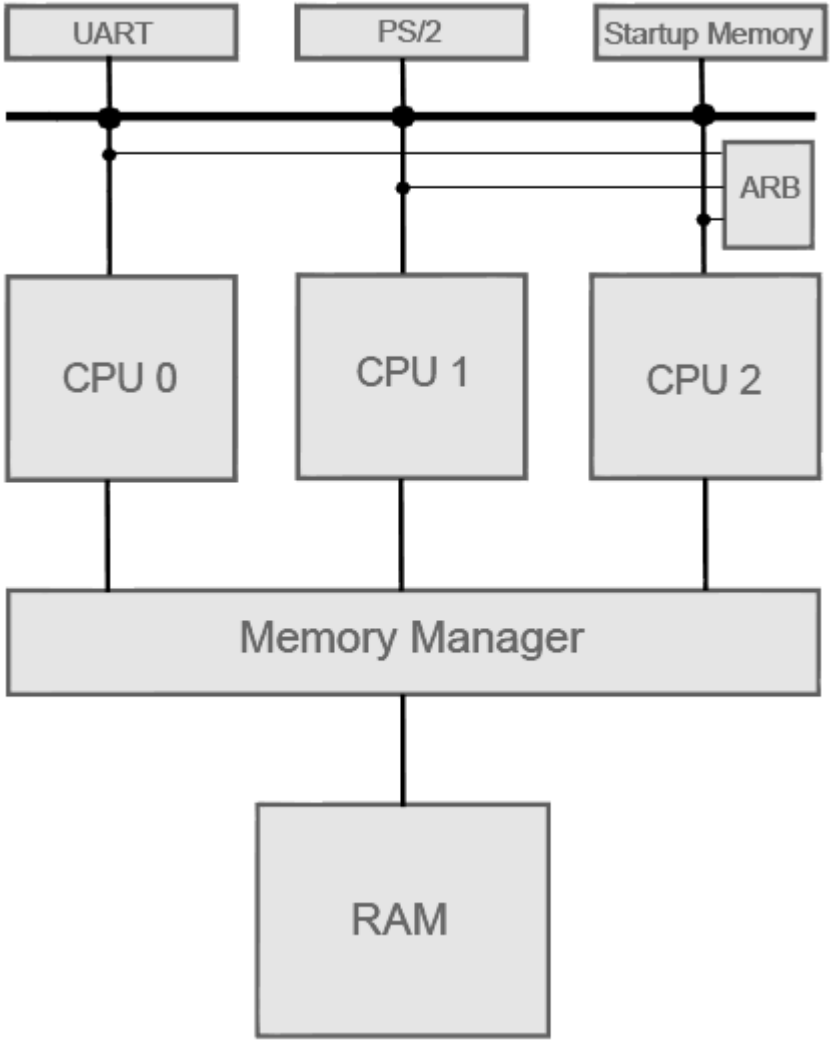
- Literaturstudium von Multi-Core- und Multi-CPU-Architekturen
- Literaturstudium von effizienten parallelen Testalgorithmen
- Erarbeitung von Konzepten zur Kopplung mehrerer SHAP-Cores
- Implementierung eines ausgewählten Konzeptes
- Evaluierung eines Konzeptes mittels Testalgorithmen
- Bewertung der Leistungsfähigkeit im Vergleich zur Ein-Core-Variante
- Zusammenfassung und Dokumentation

2. Motivation und bisherige SHAP Kopplung



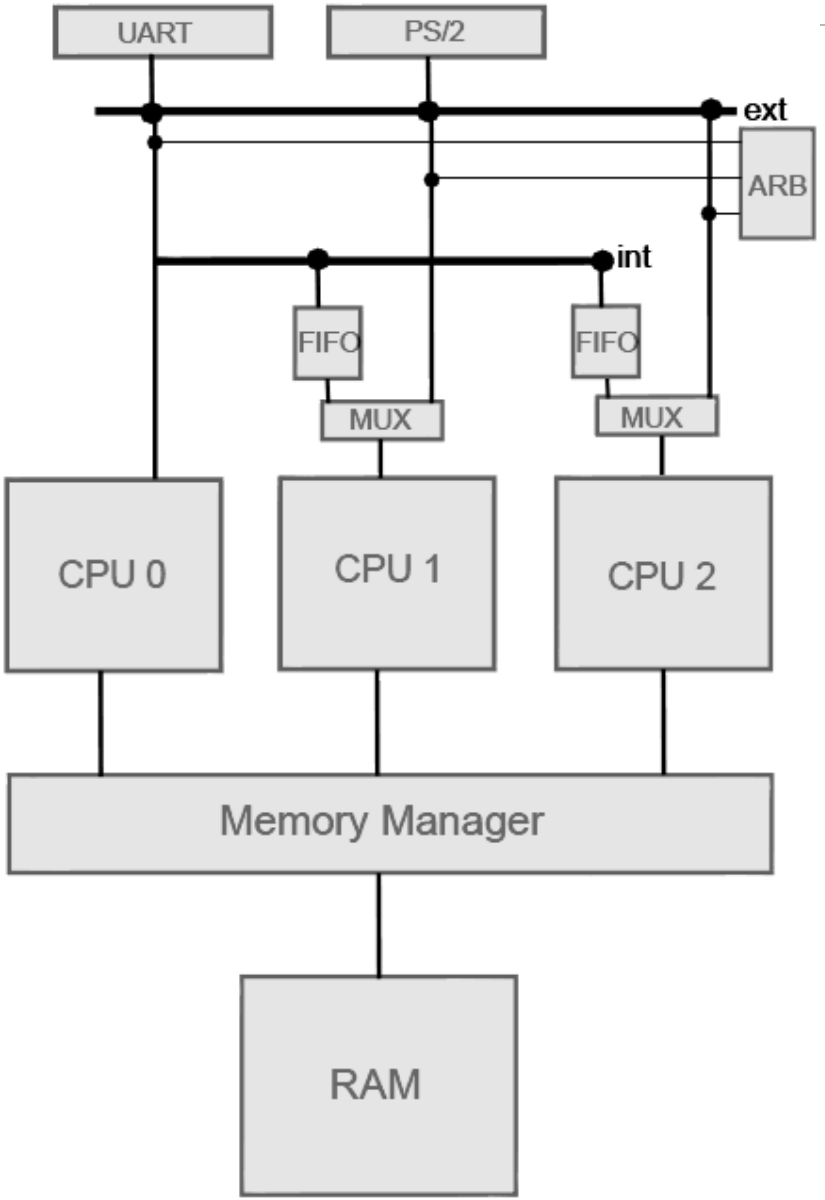
- Core 0 direkt an den externen IO-Bus angebunden
- Zusätzliche Cores am internen IO-Bus über eine IO-IO-Bridge angeschlossen
- Problem: nur ein Core hat direkten Zugriff auf den externen Bus

3.1 Kopplung über gemeinsamen Startup Memory



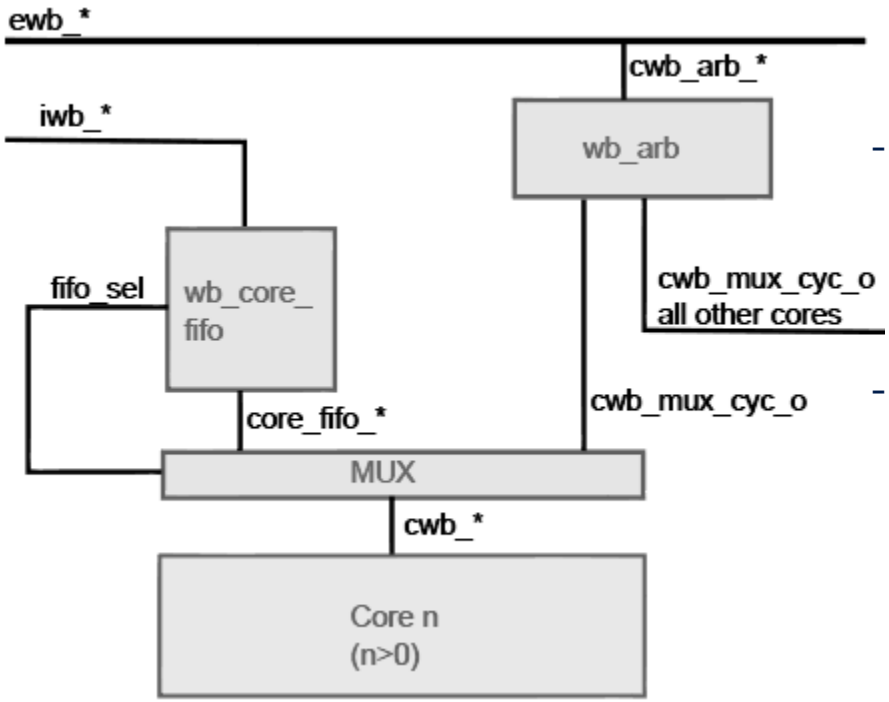
- Cores haben Zugriff auf einen gemeinsamen Bus, welcher arbitriert wird
- Start der zusätzlichen Cores über einen gemeinsam nutzbaren „Startup Memory“
- „Startup Memory“ besteht aus 3 einfachen Register und einer Auswertelogik

3.2 Kopplung über einfachen FIFO



- Zugriff aller Cores auf den gemeinsamen externen Bus über einen Arbiter
- Start der zusätzlichen Cores über einen einfachen FIFO
- FIFO besteht nur aus einem einzigen Register, in das 3-malig geschrieben und 3-malig gelesen wird
- „Select-Signal“ -> steuert MUX
  -> schaltet um nach Übertragung der Startinformationen
- Hardwareaufwand ca. gleich, wie bei dem Konzept mit dem „Startup Memory“

4.1 Implementierung der Architektur



- „wb_core_fifo“ besitzt intern einen Counter (schaltet nach 3-maligem erfolgreichem Lesen ein Select-Signal „fifo_sel“ von 0 auf 1)
- „wb_arb“ arbiertiert die Cores nach Round-Robin-Prinzip

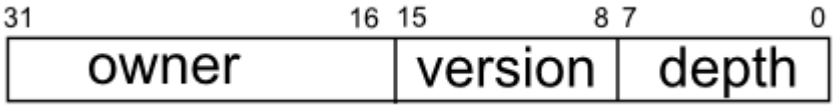
4.2 Synchronisation zwischen den Cores

Motivation

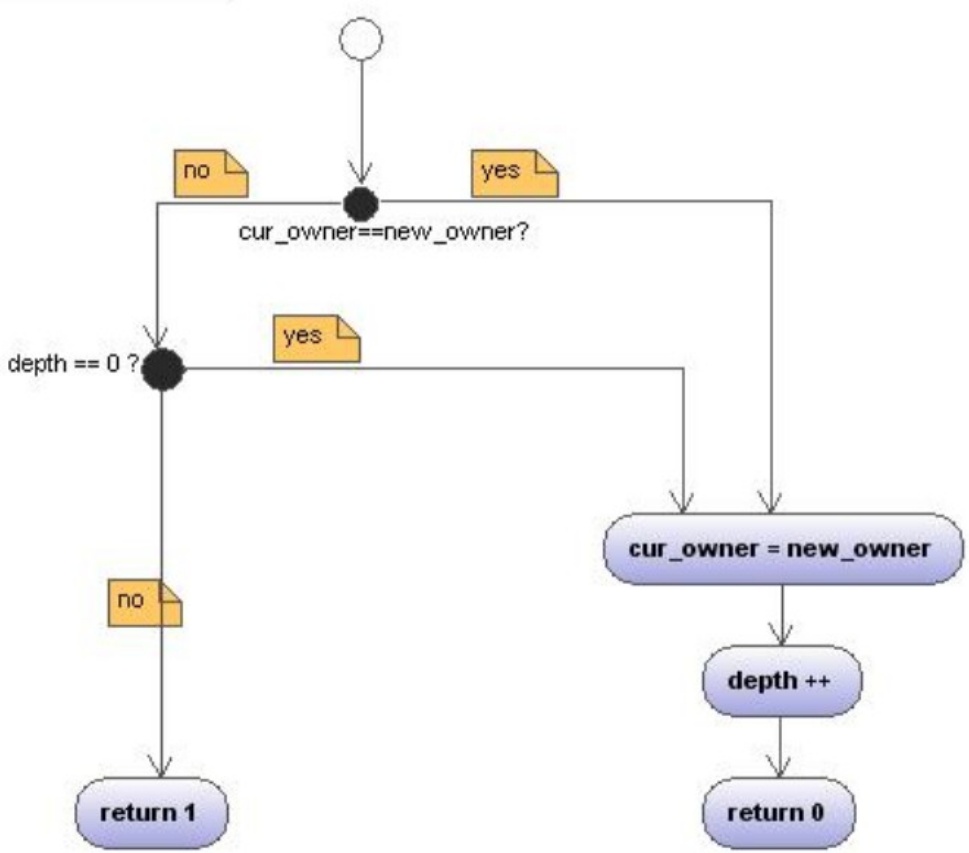
- Mehrere Cores können zeitgleich den gleichen Programmabschnitt betreten, in denen Betriebsmittel nicht geteilt werden dürfen.
- Solche „Kritischen Abschnitte“ müssen geschützt werden.
- Software-Anweisungen wie „i++“: bestehen aus mehreren Maschienenbefehlen und jeder Zeit unterbrechbar
- Notwendig: atomare Lock-Operation in Hardware (belegt Lock-Feld und ist nicht unterbrechbar)

4.2 Synchronisation zwischen den Cores

Lock Feld



Realisierung der atomaren Funktion



5. Benchmarks

Metriken

p Anzahl der Prozessoren

T_1 Dauer auf einem Core

T_p Dauer auf p Cores

Speedup

$$S_p = \frac{T_1}{T_p}$$

Effizienz

$$E = \frac{S_p}{p}$$

5. Benchmarks

Lineare Suche

- benötigt keine vorsortierte Folge wie zum Beispiel Binäre Suche
- vergleicht Element für Element mit dem Suchschlüssel in einem 15000 Elemente großen Array
- sehr gut parallelisierbar
- $S_p = p \qquad E = 1$

```
gegeben:          S          // zu suchendes Element
                  A[n]       // Array mit n unsortierten Elementen
gesucht:          result     // Indize des zu suchenden Elements
                             // result = -1 für Misserfolg
Hilfsvariablen:   ip         // Laufindex für jeden Prozessor
                  pCount     // Anzahl an Prozessoren


begin
        result = -1
                for ip = 0 to n and step pCount do
                        if A[ip] = S then result = ip, exit for
                endfor
end
```

5. Benchmarks

Matrix Multiplikation

- Multiplikation zweier 48x48 Matrizen
- im seriellen Fall mit 3 Schleifen realisierbar
- Beschleunigung durch mehrere Cores messbar durch die Parallelsierung der äußeren Schleife
- sehr gut parallisierbar
- $S_p = p \qquad E = 1$

```
for i = 0 to n – 1 do
        for j = 0 to n – 1 do
                skalProd = A[i][0]*B[0][j]
                for k = 1 to n – 1 do
                        skalProd = skalProd + A[i][k]*B[k][j]
                endfor
                C[i][j] = skalProd
        endfor
endfor
```

```
gegeben:        A[n][n],B[n][n]     // Eingabematrizen
gesucht:        C[n][n]             // Ausgabematrix
Hilfsvariablen: skalPdp             // Skalarprodukt für jeden Prozessor
                ip,jp,kp            // Laufindizes für jeden Prozessor
                pCount              // Anzahl an Prozessoren

begin
        par p = 0 to pCount – 1 do
                for ip = p to n – 1 step pCount do
                        for jp = 0 to n – 1 do
                                skalPdp = A[ip][0]*B[0][jp]
                                        for kp= 1 to n – 1 do
                                                skalPdp = skalPdp + A[ip][kp]*B[kp][jp]
                                        endfor
                                C[ip][jp] = skalPdp
                        endfor
                endfor
        endpar
end
```

5. Benchmarks

Bitonisches Sortieren

Bsp.: 16 elementige bitonische Folge, 8 Cores

	2	4	6	8	10	12	14	16	18	17	15	13	11	9	7	5
1. Schritt	Core 0															
	2	4	6	8	10	9	7	5	18	17	15	13	11	12	14	16
2.Schritt	Core 0								Core 1							
	2	4	6	5	10	9	7	8	11	12	14	13	18	17	15	16
3. Schritt	Core 0				Core 1				Core 2				Core 3			
	2	4	6	5	7	8	10	9	11	12	14	13	15	16	18	17
4. Schritt	Core 0		Core 1		Core 2		Core 3		Core 4		Core 5		Core 6		Core 7	
	2	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

$$T_1 = \frac{n}{2} * ld(n)$$

5. Benchmarks

Bitonisches Sortieren

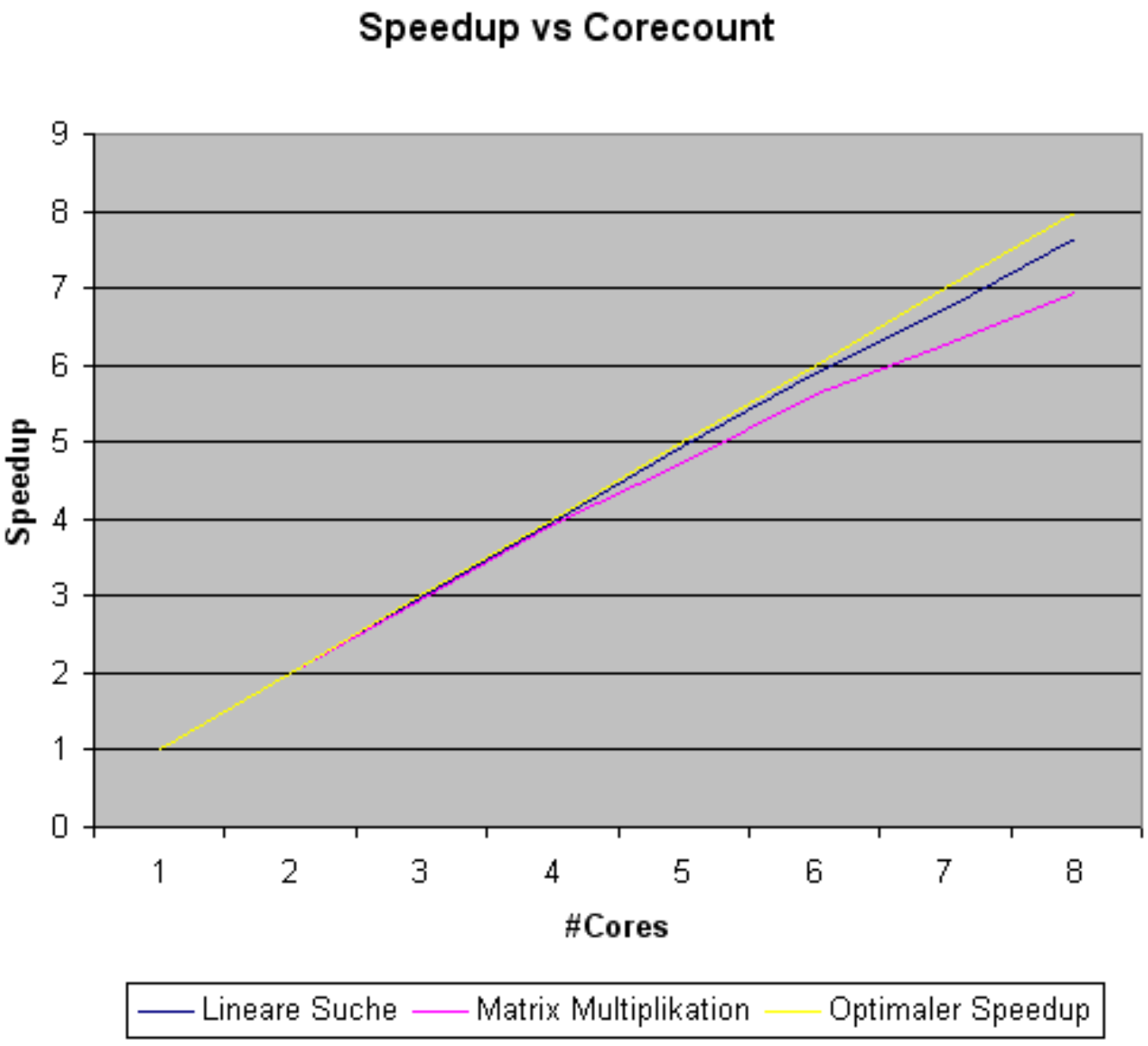
- Test mit 8192 Elementen
  - Maximaler Speedup für 2 Cores: $S_p = \frac{(4096 * ld(8192))}{(4096 + 12 * 2048)} = 1{,}857$
  - Maximaler Speedup für 4 Cores: $S_p = \frac{(4096 * ld(8192))}{(4096 + 2048 + 11 * 1024)} = 3{,}05$
  - Maximaler Speedup für 8 Cores: $S_p = \frac{(4096 * ld(8192))}{(4096 + 2048 + 1024 + 10 * 512)} = 4{,}33$

6. Auswertung

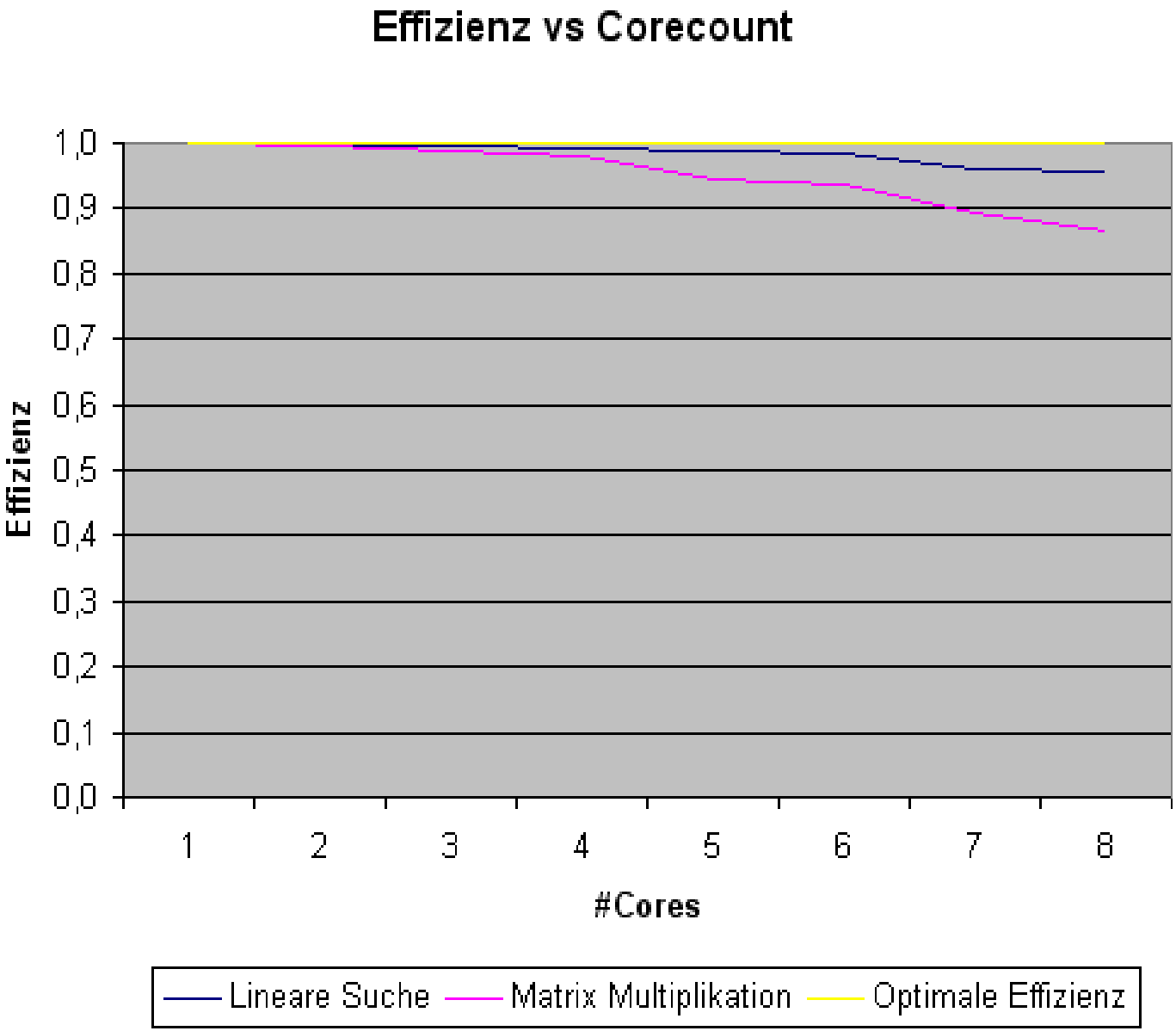
- Jeder Benchmark wurde unter Zeitmessung laufen gelassen.
- Zusätzliche Messung der Zeit, die jeder Core einzeln benötigt
- 5-malige Ausführung jedes Benchmarks auf jedem SHAP
- Tests auf dem S3SK (maximal 3 Cores) und dem ML505 (maximal 8 Cores)

TECHNISCHE UNIVERSITÄT DRESDEN

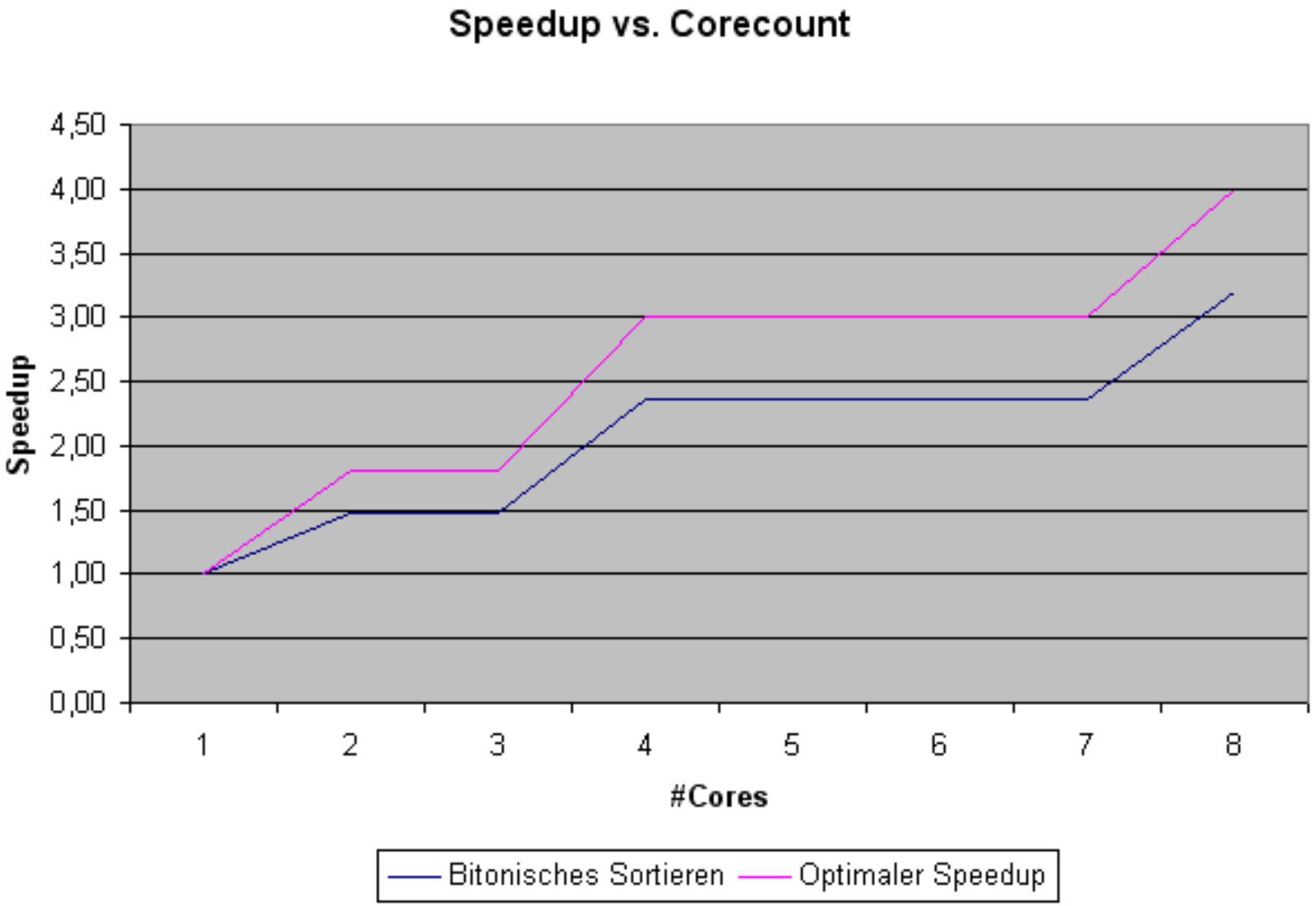
6. Auswertung



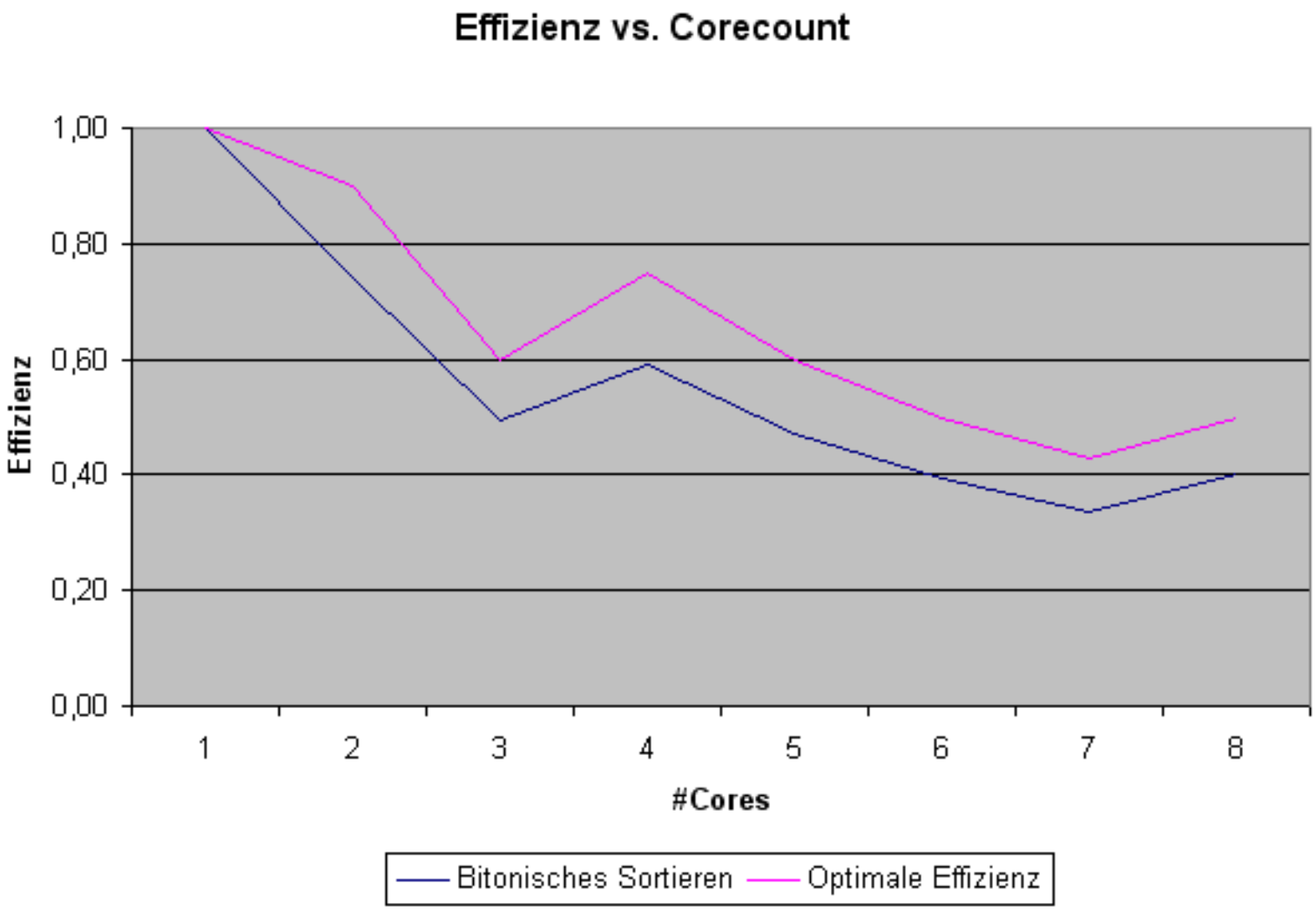
6. Auswertung



6. Auswertung



6. Auswertung



7. Zusammenfassung

- Neue Kopplungsvariante für den SHAP implementiert
- Synchronisation zwischen den Cores durch Lock-Funktion geregelt
- Evaluierung auf Basis der vorhandenen Testalgorithmen und der 3 neuen Benchmarks erfolgreich
- Optimale Effizienz erreichbar bei geeigneter Algorithmenauswahl

Vielen Dank für Ihre Aufmerksamkeit !