Abschlussvortrag

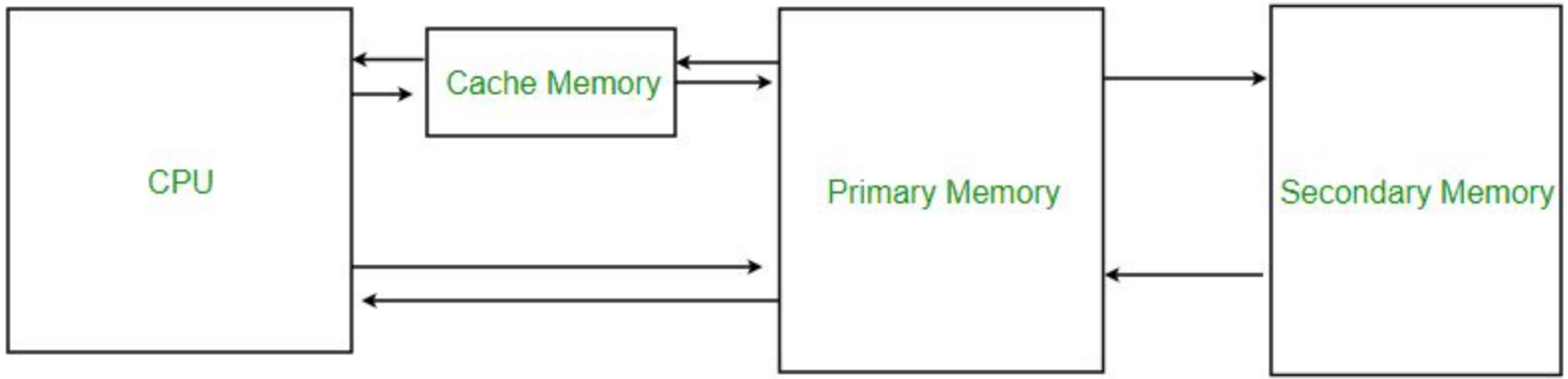
GRA

Problemstellung

Ansatz

Caches simulieren mit C++ und C zusammen mit dem SystemC-Paket

Cache Simulationsprogramm entwickelt was Caches mit User-bestimmte Qualitäten simuliert



Analyze und überlegungen

Wir können mit unserer Simulation die Effekte von verschieden Caches auf verschiedene Programme analyzieren.

Später machen wir fachliche Überlegungen zur Ergebnisse von unserer Analyze

Lösungsansatz

Bevor die Entwicklung

Höheres Abstraktionsniveau durch objektorientierten Ansatz

Für besseres Prototyping während der Entwicklung, haben wir den ersten Entwurf in Java gemacht.

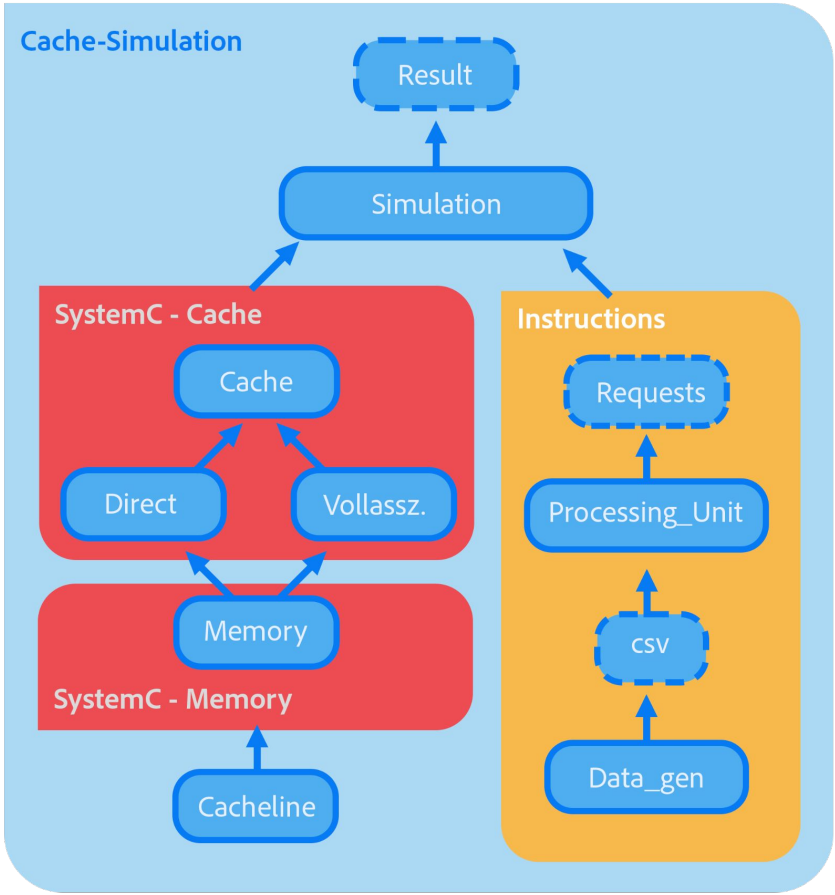
Nach MVP, können wir das programm in C++/C implementieren

Program Struktur

Die Rechte Seite in C

Linke Seite in C++ mit SystemC

Obere Teil ist in die “main.c” Klasse



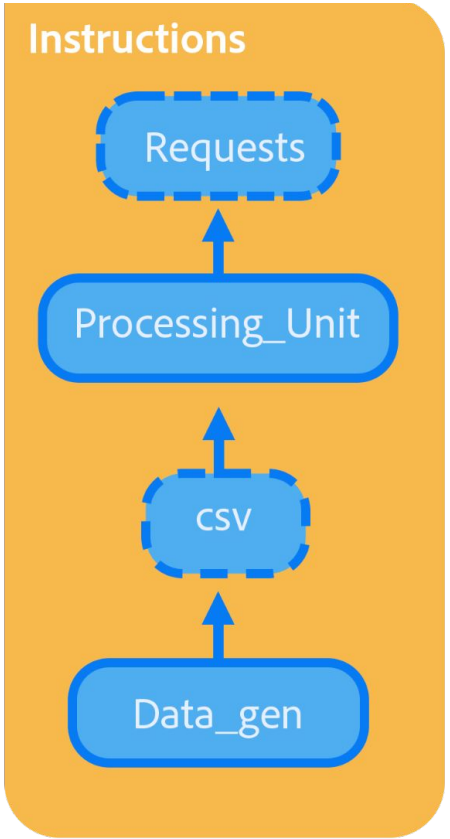
Rechte Seite, C-Teil

Eingabe: “CSV-Datei” bzw. Filepath zu eine CSV Datei.

Ausgabe: Array von Requests

Implementations Idee:

- Csv datei aufmachen
- Header überspringen
- Zeilen parsen und requests erstellen



Datenverarbeitung

Basierend auf Daten für jede Zeile ein Request erstellen mit übergebende Spezifikationen

Um unsinnige Werte und Randfälle zu verhindern überprüfen:

- Enthält die Schreibzeile **3** Werte?
- Enthält die Lesezeile **2** Werte?
- Ist der Lese-/Schreibwert ein **R**/**W** oder etwas anderes?
- Ist die Adresse eine Zahl?
- Ist die Daten in der Schreiboperation eine beschreibbare Größe?

C++, SystemC Teil

Zwei hauptklassen: Cache und Speicher

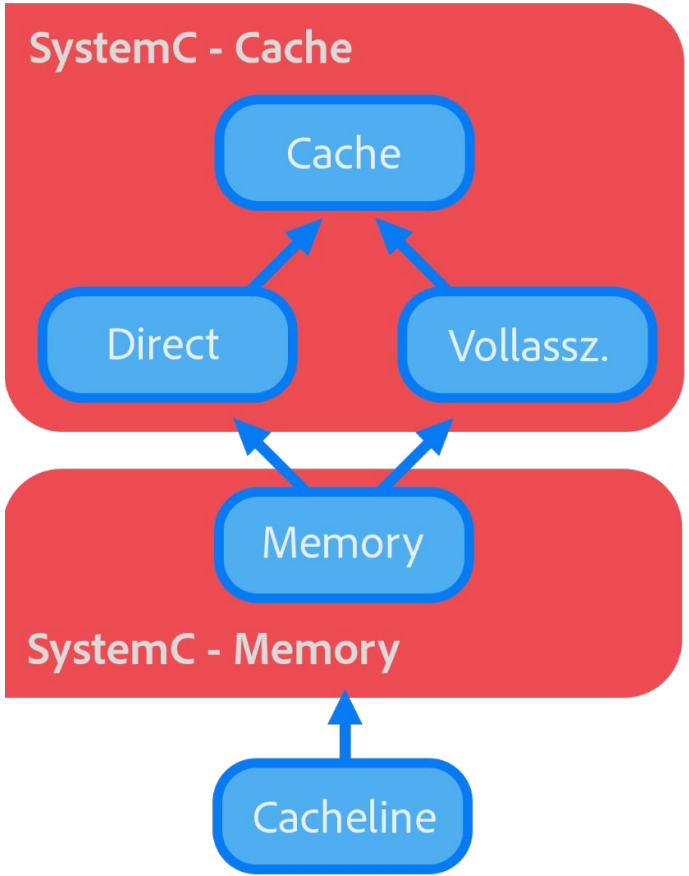
Äußerlich:

- SysC Controller
- Primitive Gate Count

Trennung, um es hardware-näher zu gestalten

Lese- und Schreib Logik in Memory Klasse

Tracefile durch tracking der signale zwischen die klassen



Simulation

Das Programm startet mit der Funktion run_simulation in der Datei simulation.cpp.

Eine Instanz des Controller-Moduls wird erstellt, um die Simulation zu steuern.

Der Controller verwaltet die Interaktionen zwischen Cache und Speicher.

Parameterübergabe und Start der Simulation

Übergabe der Parameter: Der Controller erhält alle Parameter der Anfrage und wandelt diese in SystemC-Signale um

Controller startet den Prozess controller_process

Der Cache prüft, ob die Daten vorhanden sind (Hit) oder nicht (Miss). Falls nötig, werden Anfragen an den Speicher weitergeleitet.

Abschluss

Nach Bearbeitung schreibt der Cache die gelesenen Daten in die entsprechenden Ausgangssignale (sc_out<>)

Der Controller aktualisiert Zähler für Hits, Misses und die Gesamtzahl der Zyklen

Endet wenn alle Anfragen bearbeitet wurden oder die maximale Anzahl an Stimulationszyklen erreicht ist

Das Hauptprogramm (run_simulation in simulation.cpp) empfängt die Daten und beendet die Simulation.

Optimierungen

LRU als Ersetzungs Policy

Laufzeitoptimierung

- Verwendung von constexpr und const
- Berechnungen mit Bitwise-Operationen

Top-Down Abstraktion in Modulen

Nutzung von SystemC Eigenschaften

- System-Event, Dies ermöglicht effiziente Synchronisation zwischen den Prozessen

Analyseergebnisse

Plan

Kosten unseres Caches diskutieren

Leistung in verhalt zu kosten diskutieren

welche Geschwindigkeitssteigerung wir beobachten können, und schließlich bewerten, in welchen Situationen es praktisch ist welche Caches zu verwenden

Zuletzt: Blick auf den State-of-the-art bei Caches werfen und wie Chiphersteller wie Intel und Apple die Technologie angegangen sind

Gatterkosten

Annahmen:

- Wir verwenden SRAM-Zellen mit 6 Transistoren.
- Wir zählen Transistoren als primitive Gatter.
- Wenn ein Gatter mehr als zwei Eingänge hat, zählen wir es trotzdem nur als ein Gatter. (Zum Beispiel zählt ein 4-zu-1 UND-Gatter als ein UND-Gatter)

Berechnungen

Voll Assoziative und Direct Mapped Caches haben andere logik, und damit auch verschiedene kosten

Beide caches haben diesen kosten:

- Speicherzellen = cacheLines * (CacheLineSize * 8 + tagBits + 1) * 6
- wobei "(CacheLineSize * 8 + tagBits + 1) * 6" die Transistoren pro Cachezeile bezeichnet

Direct Mapped Kosten

Comparator_gate_count = tagBits + 1

Multiplexer_Count = tagBits * (indexBits + cacheLines + 1), wobei "(indexBits + cacheLines + 1)" die Multiplexer-Gatterkosten bezeichnet

Damit ist die gesamt formel für direct mapped caches

Direct_mapped_gate_count:

tagBits + 1 + cacheLines * (CacheLineSize * 8 + tagBits + 1) * 6 tagBits * (indexBits + cacheLines + 1)

VollAssz. Cache

Comparator_gate_count = (tagBits + 1) * cacheLines

eEn zusätzliches ODER-Gatter

Damit ist der gesamt gatterkosten:

Fully_associative_gate_count = (tagBits + 1) * cacheLines + cacheLines * (CacheLineSize * 8 + tagBits + 1) * 6 + + 1

Cache Performance - Wann soll ich was benutzen?

X x x

Perspektive zu State-of-the-Art

Cache-Hierarchie: L1, L2 und L3, also mehrere Cache Schichten

- Statt wie wir, bei einem miss, direkt im haupspeicher, können wir erstmal eine cache tiefer suchen

Um die Kohärenz des Caches zwischen mehreren Kernen sicherzustellen, verwenden Intel-Prozessoren Protokolle wie MESI

Moderne Prozessoren fortschrittliche Caching-Techniken wie Hardware-Prefetching

Sources

Wikipedia - CPU Cache

Intel Architecture

PLOS - Prefetching und hierarchy und so

Zugriffszeiten

Zugriffszeiten 2

Cache Größen