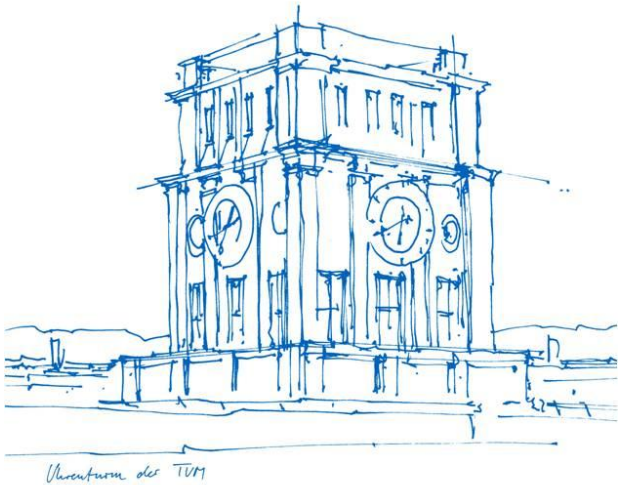
Cache Simulation und Analyse (A12)

Daniel Singh, Nicolas v. Mallinckrodt, Ziang Liu

Technische Universität München

TUM School of Information and Technology

Lehrstuhl für Design Automation



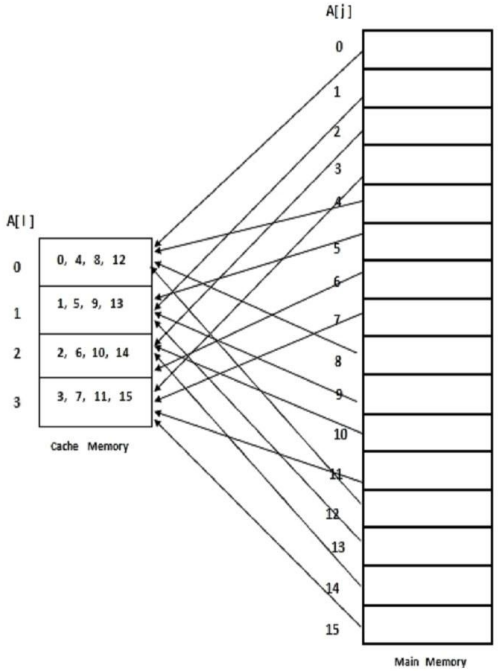
Inhaltsverzeichnis

Aufgabenstellung

- CPU-Cache-Speicher Verhalten recherchieren
- Simulation realisieren
- Simulation analysieren
- Simulationsergebnisse mit recherchierte Ergebnisse vergleichen

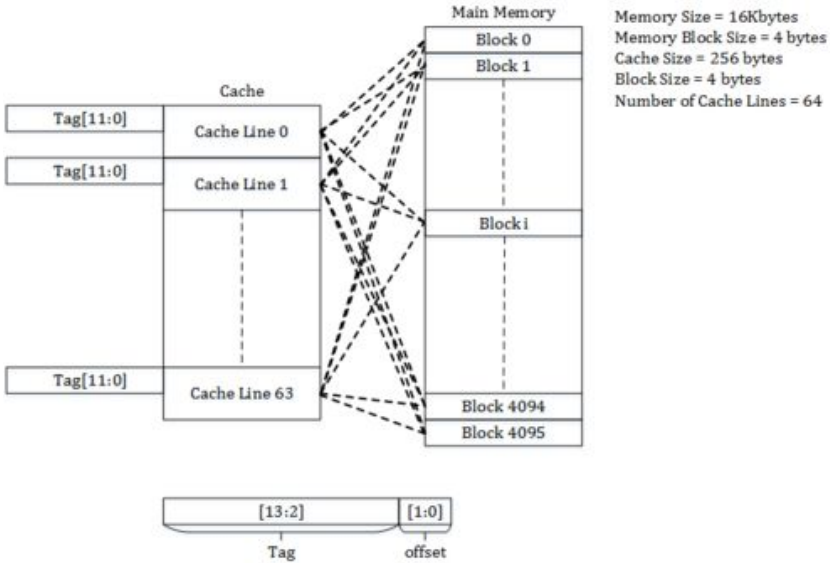
Caches

Direct Mapped



https://www.researchgate.net/figure/Direct-Cache-Mapping-Ai-Aj-mod-Main-memory_fig1_312369247

Vollassoziativ



https://commons.wikimedia.org/wiki/File:Direct-Mapped_Cache_Snehal_Img.png

Hauptspeicher

- Adressen mit gleichem Tag gemeinsam abspeichern:

```
//Tag als Schlüssel, als Wert einen Vektor:
std::unordered_map<uint32_t, std::vector<uint32_t>> memory;
```

- Block an Cache senden:

  Lösung 1: Alle Adressen als Signal speichern und jede Adresse einzeln senden

  Problem: Bei größeren Cache Zeilen viele Signale zu koordinieren

- Lösung 2:

```
struct Block {
    std::vector<uint32_t> block;

    Block(): block(0, 0) {}

    Block(uint32_t size): block(size, 0) {}

    //Zuweisungsoperator überschrieben
    Block& operator=(const Block& block2) {
        if (this != &block2) {
            block = block2.block;
        }
        return *this;
    }

    //Gleichheitsoperator überschrieben
        bool operator==(const Block& block2) const {
        return block == block2.block;
    }
};
```

```cpp
//Definiert wie ein sc_signal<Block> bei print ausgegeben werden soll
inline std::ostream& operator<<(std::ostream& output, const Block& block) {
    output << "Block: [";
    for (int i = 0; i < block.block.size(); i++) {
        output << block.block[i];
        if (i < block.block.size() - 1) {
            output << " | ";
        }
    }
    output << "]";
    return output;
}

//Definiert wie ein sc_signal<Block> bei trace ausgegeben werden soll
inline void sc_trace(sc_trace_file* file, const Block& block, const std::string& name) {
    for (int i = 0; i < block.block.size(); i++) {
        sc_trace(file, block.block[i], name + ".value=" + std::to_string(i) + "");
    }
}
```

Cache-Speicher-Verbindung

- Problem:

  Signale brauchen Zeit zum Bearbeiten

  -> Cache Logik pausieren und auf Speicher warten

- Lösung:

```
sc_event blockUpdated;
.
.
.
out.write(returnBlock);
blockUpdated.notify(SC_ZERO_TIME);
```

Signale senden

- Lösung 1:

```
sensitive << request << value << write
```

- Problem: Funktion unerwartet aufgerufen

- Lösung 2:

```
struct MEMORY_REQUEST {
   uint32_t addr;
   uint32_t data;
   int we;
   int requestNum;

   MEMORY_REQUEST(): addr(0), data(0), we(0), requestNum(0){}
   MEMORY_REQUEST(uint32_t addr, uint32_t data, int we, int requestNum):
     addr(addr), data(data), we(we), requestNum(requestNum) {}
}
```

Cache Struktur

- Cache Zeilen mit gleichem Tag speichern mehrere Adressen gemeinsam ab:

  Tag: 0x1020 [100, 573,1900, 6102]
  Tag: 0x2001 [4, 3,18193, 2832]

```
struct CacheLine {
   bool occupied = false;
   Tag tag = 0;
   std::vector<uint32_t> line;

   CacheLine(uint32_t size): line(size, 0){}
};
```

Cache Struktur

Direct-mapped:

```
Offset offset = request.read().addr << (32-offsetBitAmount);
offset >>= (32-offsetBitAmount);

Index index = request.read().addr << (32-offsetBitAmount-indexBitAmount);
index >>= (32-indexBitAmount);

Tag tag = request.read().addr >> (offsetBitAmount+indexBitAmount);
```

Vollassoziativ:

```
Offset offset = req.addr << (32 - offsetBitAmount);
offset >>= (32 - offsetBitAmount);

Tag data_tag = req.addr >> offsetBitAmount;

int index = findLine(data_tag);
```

Main.c

Gesucht:
Ein Rahmenprogramm geschrieben in c zur Ausführung der Simulation

Probleme:

1. Sicherstellung, dass nur valide Inputs angenommen werden
2. Bei einem Fehler verstehen was falsch läuft und dies den Nutzer mitteilen

Ansatz:
Input-Phrasing nach den gnu.org standarts für `getopt_long`

Merge-Sort Rahmenprogramm

Gesucht:
Ein Programm welches die read und write Instruktionen eines Merge-Sorts in einem für unseren Cache lesbaren Format generiert.

Probleme:
1 .Wie simulieren wir den Speicher und seine Adressen.
2. Nach welcher Logik werden die Subarrays geschrieben und gelesen.

Merge-Sort Rahmenprogramm

Ansatz:
Ein Zugriffsschema ähnlich zu einem Stack, bei Rekursive Abstieg werden die Zwischenergebnisse nebeneinander geschrieben beim Aufstieg werden nicht mehr benötigte Daten überschrieben und der Pointer entsprechend angepasst.

Merge-Sort Rahmenprogramm

Visualisierung:

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]							

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[2	0]					

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[2	0]	[2]				

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[2	0]	[2]	[0]			

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[0	2]	[2]	[0]			

Merge-Sort Rahmenprogramm

Visualisierung:

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[0	2]	[1	3]			

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[0	2]	[1	3]	[1]		

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[0	2]	[1	3]	[1]	[3]	

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[2	0	1	3]	[0	2]	[1	3]	[1]	[3]	

Adresse	0	1	2	3	4	5	6	7	8	9	10
Datum	[0	1	2	3]	[0	2]	[1	3]	[1]	[3]	

Analyse

Merge-sort small: (33 request)

Direct mapped: (cachelines 128, ansonsten default werte)	Associative: (cachelines 128, ansonsten default werte)
Simulation Results: Cycles: 1165 Cache hits: 23 Cache misses: 10	Simulation Results: Cycles: 1165 Cache hits: 23 Cache misses: 10

Analyse

Merge-sort big: (8127 request)

Direct mapped: (cachelines 128, ansonsten default werte)	Associative: (cachelines 128, ansonsten default werte)
Simulation Results: Cycles: 257235 Cache hits: 5961 Cache misses: 2166	Simulation Results: Cycles: 301635 Cache hits: 5517 Cache misses: 2610

Fazit

Direct-Mapped:

- Geringe Latenz
- Verarbeiten Anfragen schneller
- Billig
- Höhere Miss-quote

Vollassoziativ:

- Höhere Latenz und Verarbeitungszeit
- Teuer und komplex -> viele primitive Gatter
- Höhere Treff-quote