

Raytracing auf Desktop PCs

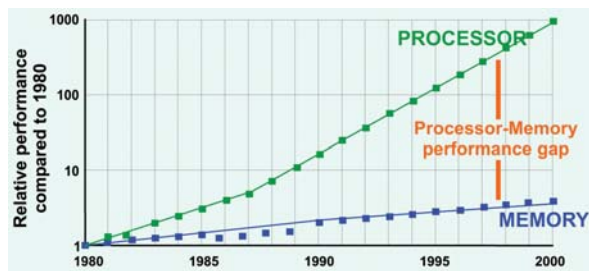
Optimizing Cache Usage

(Intel Corp.)

von
Martin Stöcker

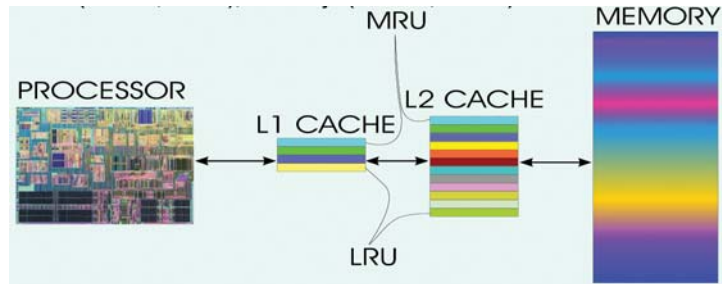
Motivation

- Geschwindigkeit der Prozessoren verdoppelt sich alle 18 Monate (Moore's Law)
- Geschwindigkeit des Speichers wird im Jahr um ca. 7 % erhöht
- Processor-Memory Performance Gap verdoppelt sich alle 21 Monate
- Seit 20 Jahren werden Caches genutzt, um das Geschwindigkeits-Problem zu überbrücken
- Trotzdem entstehen bei vielen Applikationen Wartezeiten zwischen Prozessor und Speicher
- Cache muss effizienter ausgenutzt werden



Cache Prinzip

- Mehrere Cache-Levels sind vorhanden, wobei jedes Level eine bestimmte Größe und Geschwindigkeit hat.
- Beispiel: L1 = 16 KB, 1 ns
L2 = 256 KB, 10 ns
Speicher = 512 MB, 100 ns



Optimierungsstrategie

- Performance-Steigerung durch holen der Daten aus dem Prozessor-Cache anstatt aus dem Hauptspeicher
- Daten müssen in den Prozessor gebracht werden, bevor sie benötigt werden
- Unterstützung durch Streaming SIMD Extensions, die sogenannte Prefetch-Instruktionen bereitstellen
- Durch Prefetch-Operationen soll die Wartezeit des Prozessors auf den Speicher eliminiert werden

Prefetch Instruktionen

- Einfügen der Prefetch Instruktionen von Compiler oder Programmierer
- Zugriff auf ein Minimum von einer Cache-Line von Daten (128 byte im Pentium 4), vor den aktuell benötigten Daten
- Lange, reguläre Datenmuster bearbeitet der automatische Hardware-Prefetcher, Software Prefetching wird nicht benötigt

- Data Reference Patterns:
 - Temporal
 - Spatial
 - Non-Temporal

Software Data Prefetch

- Prinzip : „Wenn du dazu länger brauchst, dann fang früher damit an“
- Bald benötigte Speicherinhalte werden schon früher beim Speichersubsystem angefordert, sodass sie rechtzeitig zur Verfügung stehen, ohne Prozessor zu blockieren
- Angeforderte Werte werden im Cache Speicher abgelegt
- Prefetch Instruktion lädt temporäre oder nicht-temporäre Daten in den spezifizierten Cache-Level
- Datenzugriffstyp und Cache-Level werden von der Instruktion spezifiziert
- Prefetch Instruktion ist implementierungsspezifisch
- Prefetch Instruktion geben lediglich einen Hinweis an die Hardware und generieren keine Exceptions oder Fehler
- Exzessive Implementierung von Prefetches kann Speicherbandbreite vergeuden und führt zu Performanceverlust
- Prefetches vermindern Overhead in Speichertransaktionen und verbergen Speicherlatenzen in den Hintergrund

Software Data Prefetch

Grundvoraussetzungen zur Implementierung:

- Prozessor muss einen speziellen Prefetch-Befehl bereitstellen
- Cache muss nicht-blockierend sein
- Speicherstellen, auf die in Kürze zugegriffen wird, lassen sich im Voraus bestimmen

Weitere Charakteristiken:

- Kann mit irregulären access patterns umgehen
- Kann weniger Bus Bandbreite nutzen
- Muss zu neuem Code hinzugefügt werden

Hardware Data Prefetch

Charakteristiken:

- Automatischer Prefetcher überwacht data access patterns und prefetcht Daten automatisch ohne Eingriff des Programmierers
- Hardware Prefetch hilft, wenn Software auf große Speicherblöcke zugreift und diese Blöcke sequentiell und in aufsteigender Reihenfolge durchläuft (Datenlokalität)
- Hardware Prefetcher versucht 256 bytes oberhalb der aktuellen Datenzugriffs-Locations zu bleiben
- Arbeitet mit existierenden Applikationen
- Braucht reguläre access patterns
- Benötigt einige Cache-Misses, bevor er beginnt zu arbeiten

Pentium 4 Processor Implementation

PREFETCHH Instructions:

- PREFETCHNTA
- PREFETCHT0
- PREFETCHT1
- PREFETCHT2

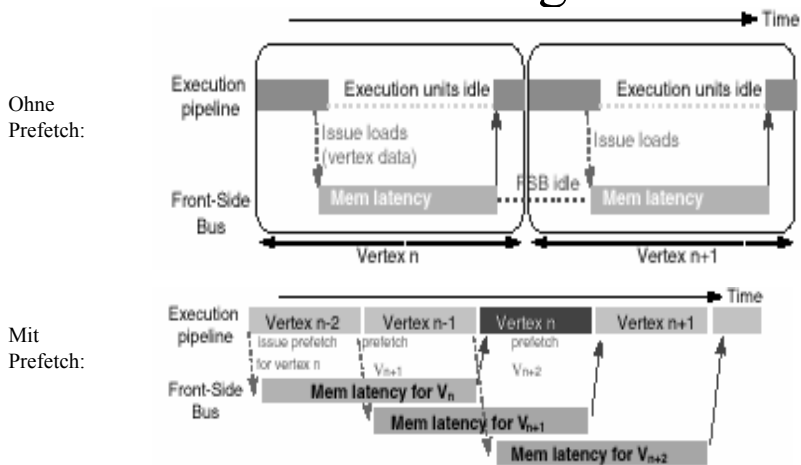
Hints sind:

- T0 (temporal data) - prefetch data into all levels of the cache hierarchy.
- T1 (temporal data with respect to first level cache) - prefetch data into level 2 cache and higher.
- T2 (temporal data with respect to second level cache) - prefetch data into level 2 cache and higher.
- NTA (non-temporal data) – Fetch the data into the second-level cache, minimizing cache pollution.

Prefetch Implementation in PIII and P4

<u>Prefetch type</u>	<u>Pentium III</u>	<u>P4</u>
<u>PrefetchNTA</u>	Fetch 32 bytes Fetch into L1 only	Fetch 128 bytes Fetch only into 1 way of L2
<u>PrefetchT0</u>	Fetch 32 bytes Fetch into L1 and L2	Fetch 128 bytes Fetch into L2
<u>PrefetchT1</u> , <u>PrefetchT2</u>	Fetch 32 bytes Fetch into L2 only	Fetch 128 bytes Fetch into L2 only

Latency Hiding with SW Prefetching



Prefetching Usage Checklist

1. Prefetch Scheduling Distance herausfinden
2. Prefetch Concatenation benutzen
3. Anzahl der Prefetches minimieren
4. Prefetch mit Programmcode mischen
5. Cache blocking Techniken verwenden (z.B. strip mining)

1. Prefetch Scheduling Distance

Eingefügte Prefetch-Befehle könnten im Programmcode überflüssig sein und unnötig Overhead produzieren

Gründe können sein:

- Speicherinhalt wird schon vor seiner Verwendung wieder aus Cache geworfen
- Prefetch-Operation wird zu kurz vor der Verwendung des Datums eingefügt
- Speicherinhalt ist bereits im Cache, Prefetch-Operation ist überflüssig

Ergebnis:

1. Alle Referenzen auf den Hauptspeicher finden
2. Analyse, ob für sie ein Prefetching notwendig oder überflüssig ist
3. Für alle notwendigen Prefetches muss der optimale Zeitpunkt ermittelt werden

1. Prefetch Scheduling Distance

$$psd = \left| \frac{N_{lookup} + N_{xfer} \cdot (N_{pref} + N_{st})}{CPI \cdot N_{inst}} \right|$$

Psd = prefetch scheduling distance

Nlookup = Anzahl der Takte für Lookup-Latenzen

Nxfer = Anzahl der Takte um eine Cache-Line zu übertragen

N_{pref} und N_{st} = Anzahl der Cache Lines, die geprefecht und gespeichert werden sollen

CPI = Anzahl der Takte pro Instruktion

N_{inst} = Anzahl der Instruktionen im Bereich einer Schleifeniteration

1. Prefetch Scheduling Distance

Nachteile der Näherungsformel:

- Gelten nur für einen Prefetch, nicht für mehrere Datenströme gleichzeitig
- Informationen werden benötigt, die in der Regel nicht verfügbar sind

Ergebnis :

- Durch Ausprobieren eine geeignete Prefetchentfernung finden
- Als Faustregel gilt, dass die Prefetchdistanz groß genug sein muss um die Daten vor der Bearbeitung im Cache zu haben und klein genug damit die Daten nicht schon wieder ersetzt wurden wenn sie benötigt werden. Ist dies nicht der Fall sind Prefetches eher kontraproduktiv.

2. Prefetch Concatenation

Ohne Prefetch Concatenation

```
for (ii = 0; ii < 100; ii++) {  
  for (jj = 0; jj < 32; jj+=8) {  
    prefetch a[ii][jj+8];  
    computation a[ii][jj];  
  }  
}
```

Mit Prefetch Concatenation und Loop Unrolling

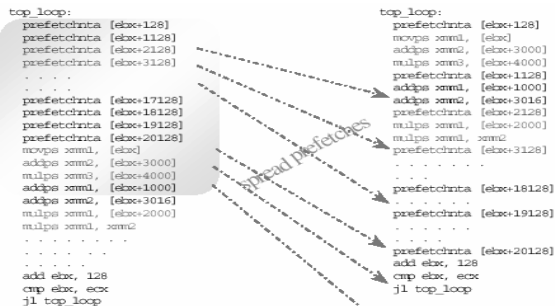
```
for (ii = 0; ii < 100; ii++) {  
  for (jj = 0; jj < 24; jj+=8) {  
    prefetch a[ii][jj+8];  
    computation a[ii][jj];  
  }  
  prefetch a[ii+1][0];  
  computation a[ii][jj];  
}
```

3. Minimize Number of Prefetches

- Prefetch Instruktionen sind nicht komplett frei bezogen auf Buszyklen, Maschinentakten und Ressourcen
- Performanceverlust bei exzessiver Benutzung der Prefetch Operation z.B. in kleinen Schleifen
- Besser: Loop Unrolling und/oder Software-Pipelining, um Anzahl der Prefetches gering zu halten

4. Mix Prefetch with Computation Instructions

- Prefetch Instruktionen sollten nicht vor einer Schleife oder am Anfang eines Schleifenrumpfes angehauft werden
- Bestmogliche Performance, wenn Prefetch Instruktionen mit Programmcode vermischt werden
- Nutzliche Heuristik beim Pentium 4 ist, Prefetch Instruktionen auf alle 20-25 Takte zu setzen



5. Prefetch and Cache Blocking Techniques

- Cache Blocking Techniken verbessern die Cache Hit Rate
- Strip Mining: Datenmengen, die größer sind als der Cache werden in kleinen Gruppen verarbeitet, die in den Cache passen. Dies erlaubt temporären Daten, länger im Cache zu bleiben um somit den Datenverkehr im Bus zu entlasten

5. Prefetch and Cache Blocking Techniques

Ohne Strip Mining:

```
while (nvtx < MAX_NUM_VTX) {
  prefetchnta vertexi data // v=[x,y,z,nx,ny,nz,tu,tv]
  prefetchnta vertexi,1 data
  prefetchnta vertexi,2 data
  prefetchnta vertexi,3 data
  TRANSFORMATION code // use only x,y,z,tu,tv of a vertex
  nvtx+=4
}
while (nvtx < MAX_NUM_VTX) {
  prefetchnta vertexi data // v=[x,y,z,nx,ny,nz,tu,tv]
  // x,y,z fetched again
  prefetchnta vertexi,1 data
  prefetchnta vertexi,2 data
  prefetchnta vertexi,3 data
  compute the light vectors // use only x,y,z
  LOCAL LIGHTING code // use only nx,ny,nz
  nvtx+=4
}
```

Mit Strip Mining:

```
while (nstrip < NUM_STRIP) {
  /* Strip-mine the loop to fit data into one way of the second-level
  cache */
  while (nvtx < MAX_NUM_VTX_PER_STRIP) {
    prefetchnta vertexi data // v=[x,y,z,nx,ny,nz,tu,tv]
    prefetchnta vertexi,1 data
    prefetchnta vertexi,2 data
    prefetchnta vertexi,3 data
    TRANSFORMATION code
    nvtx+=4
  }
  while (nvtx < MAX_NUM_VTX_PER_STRIP) {
    /* x y z coordinates are in the second-level cache, no prefetch is
    required */
    compute the light vectors
    POINT LIGHTING code
    nvtx+=4
  }
}
```