Prefetching-Techniken

Christoph Berg

cb@cs.uni-sb.de

Compiler Design Lab

17. Februar 2003

Inhalt

- Prefetching, Begriffe
- Historik
- Implementierungen
- Neue Ideen

Fragen!

Prefetching

- Speicher ist langsam
- Caches speichern benutzte Daten aus dem Speicher
  - schneller Zugriff
- erster Zugriff trotzdem langsam
  - Prefetching benutzen
- Prefetching
  - Daten aus Speicher/L2-Cache holen, *bevor* sie gebraucht werden
  - ca. 10-50 Zyklen früher (= ?? Instruktionen)

Klassifikation

- Instruktionen / Daten, Einzel- / Mehrprozessor
- Software: `prefetch`-Befehl (keine Exceptions)
  - wenig Hardware-Unterstützung, evtl. höhere Taktrate
  - Compile-Zeit-Informationen
    - Programm-spezifische Optimierungen möglich
  - Overhead, schwierig für I-Prefetching
- Hardware: Prefetch-Puffer, . . .
  - kostet, mehr Memory-Traffic, I und D
  - keine (?) Compiler/Programmierer-Unterstützung nötig
  - Speedup ohne neu-kompilieren
  - kein Overhead (weniger I-Cache-Pollution!)
  - Laufzeitinformationen

Performance

- Performance
  - Coverage, Accuracy, Memory bandwidth, Cache pollution, MCPI, Speedup, ...
- Prefetch:
  - *possible*: Adresse (genügend) vorher bekannt
  - *unnecessary*: Daten schon im/auf dem Weg zum Cache
  - *effective*: nicht zu früh, nicht zu spät

1982: One-Block-Ahead

[Smith: Cache Memories]

- *prefetch always*: immer nächste Line holen (keine gute Idee)
- *prefetch on miss*: prefetch nur bei cache miss
  - kann Zahl der Misses halbieren
- *tagged prefetch*: Tag bit für jede Line
  - Line prefetchen: auf 0 setzen
  - Line benutzen: auf 1 setzen
  - 0→1-Übergang: nächste Line prefetchen
  - kann zu langsam sein
- n-Block-Ahead

1990: Victim Caches, Stream Buffers

[Jouppi: Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache Prefetch Buffers]

- Miss Cache, Victim Cache: LRU, voll assoziativ
  - 8 Einträge: beseitigt 25-100% D-Misses, 2-30% I-Misses
- Stream Cache: bei Miss füllen mit folgenden Lines
- FIFO: 8 Einträge: 5-35% D-Misses, 60-70% I-Misses weniger
- 4-way FIFO: 17-50% D-Misses, 63-70% I-Misses weniger
- Quasi-sequential 4-way: 23-50% D-Misses, 63-70% I-Misses weniger

1991: Array-Prefetching in Software

[Callahan, Kennedy, Porterfield: Software prefetching]

- Arrays in (inneren) Schleifen für nächste Iteration prefetchen
-
```
for(i = 0; i < 100; i++) {
  prefetch(A[i+1]);
  a += A[i];
}
```
- eine Iteration reicht oft nicht aus
- Kontrollfluß kann in jedem Durchlauf anders sein
- >95% Hit rate für Array-Code
- 0-2% mehr Memory-Traffic, $\sim$*28% Instruktionen*
- noch offene Frage, ob Prefetching zuviel Overhead hat

Overflow iteration

- max. Iterationen, die weniger Speicher benutzen als Cache groß ist
  - Berechnung von Hand
- nur Werte prefetchen, bei denen alle einkommenden Abhängigkeits-Kanten die Overflow iteration überschreiten
- nützliche Prefetches: 33% ohne, 59% mit Overflow iteration
- nur noch 20% Overhead (0,1% weniger Hits)
- Erweiterungen: Register für Adressen reservieren?
- mehrere Iterationen voraus prefetchen

1991: Reference Prediction Table

[Chen, Baer: An effective on-chip preloading scheme to reduce data access penalty]

- Einträge: tag, prev_addr, stride, state
- state: init, transient, steady, no-pred
- LA-PC läuft vor PC (incl. branch prediction)
- 512 Einträge: >90% hits (7 von 10 SPEC)
  - Kombination von Software und Hardware
- $n\times$ stride

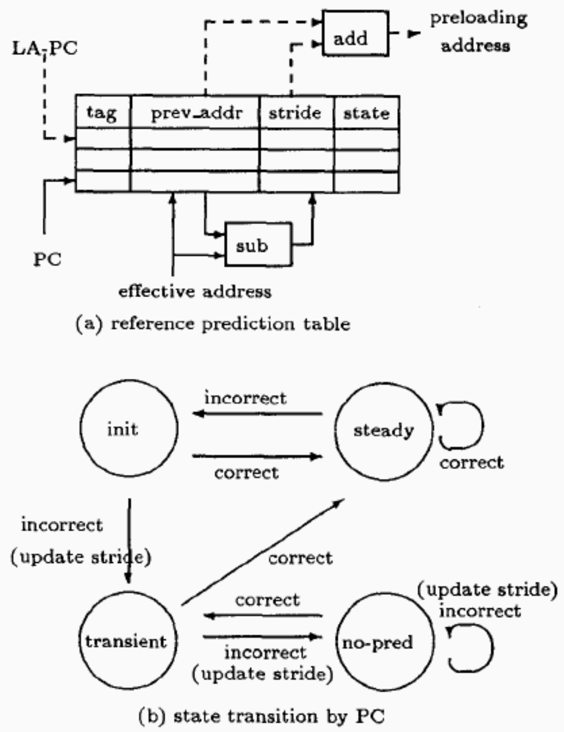


Figure 2: Reference Prediction

1992: Dicht besetzte Matrizen

[Mowry, Lam, Gupta: Design and evaluation of a compiler algorithm for prefetching]

- Software pipelining: loop prolog, epilog
- Prefetche $\lceil \frac{l}{s} \rceil$ Iterationen (l Latenz, s kürzester Pfad)
- Reuse-Analyse über Zugriffs-Vektoren + Heuristiken
- Speedup: 5-100%, bis zu 50% mehr Instruktionen
- 0-60% unnötige Prefetches (60-95% bei vollst. PF)
- Problem: unbekannter Einfluß der Cache-Größe
- "More complex hardware prefetching appears unnecessary."

1994: Wrong Path Prefetching

[Pierce, Mudge: Wrong-path Instruction Prefetching]

- Prefetche *falschen* Pfad bei Branches
- Reduziert I-Misses um ca. $\frac{1}{3}$

1995: PowerPC

[Bernstein, Doron, Freund: Compiler Techniques for Data Prefetching on the PowerPC]

- analog [Mowry 92]
- Problem: unifizierter Cache → kleinere Cache-Größe im Modell, 8-fach → voll assoziativ
- symbolische Ausdrücke für Indizes finden, Differenzen bilden
- für jede innere Schleife
  - pro Gruppe (spatial/temporal) nur eine Referenz
  - self temporal locality entfernen
  - loop unrolling (self spatial locality)
- Speedup: -2,5-22% (ggüber kein Prefetching)

1995: Call Pointer-Prefetching

[Lipasti, Schmidt, Kunkel, Roediger: SPAID: Software Prefetching In Pointer- and Call-Intensive Environments]

- Idee: Prefetching von Pointern an den Callsites
- *ein* Prefetch genügt! (erstes Argument)
  - `this`-Pointer
- 5-7% weniger miss penalty

1996: Jump Pointer-Prefetching

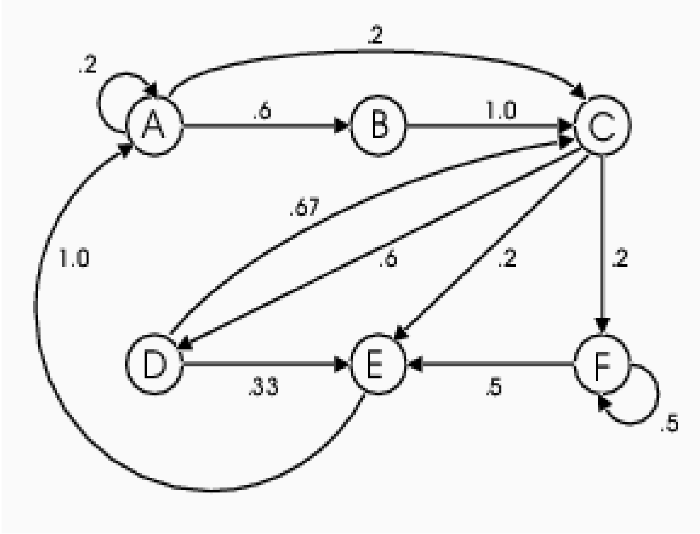
[Luk, Mowry: Compiler-Based Prefetching for Recursive Data Structures]

- Data linearization hilft, eigenes `malloc()`
- Pointer-Chasing geht nicht (max. 1 Dereferenzierung)
- Compiler identifiziert rekursive Datenstrukturen
- Greedy Prefetching: alle Pointer fetchen
  - $\frac{k-1}{k}$ Knoten in k-Bäumen in den Cache
  - bis zu 45% Speedup (meist viel weniger)
- History-pointer Prefetching: FIFO der Länge d
  - Bei Zugriff auf n_i History-pointer von n_{i-d} auf n_i setzen
  - Beim nächsten Durchlauf diese Pointer prefetchen
  - 40% Speedup über Greedy Prefetching in 1 Fall

1997: Markov Predictors

[Joseph, Grunwald: Prefetching Using Markov Predictors]

- 1-history der Misses (n-history kein Vorteil)



- Tabelle: Miss address, 4 Nachfolger: Adresse, Priorität (LRU)
- Priorisierung der Prefetches
- 1MB Tabelle: 54% weniger Stalls

1998: Reference Prediction Table in Software

[Zucker, Lee, Flynn: An Automated Method for Software Controlled Cache Prefetching]

- RPT simulieren; Performance $\geq$ Hardware-RPT
- RPT-Inhalt (Adresse, Stride) durch Profiling ermitteln
- 15% der nützlichen Prefetch-Befehle erzeugen 90% der nützlichen Prefetches
- diese Befehle als `prefetch-by-stride` einfügen
  - benutzt letzte Load- oder Store-Instruktion
- bis zu 278% Speedup ggüber kein Prefetching

1998: Prefetch Filtering

[Luk, Mowry: Cooperative Prefetching: Compiler and Hardware Support for Effective Instruction Prefetching in Modern Processors]

- Prefetch-Scheduling und -Optimierung auf CFG (Heuristiken)
- I-Prefetcher: generiert Soft- und Hardware-Prefetches (next-n-line)
- Filter: Prefetch bit für jede I-Line: geprefetcht, aber noch nicht benutzt
- 2-bit sat. Zähler: wie oft Line geprefetcht aber nicht benutzt wurde vor Replace
- 6,4% Speedup ggüber anderen Schemata auf nicht-numerischem Code

1998: Correlation Table

[Roth, Moshovos, Sohi: Dependence Based Prefetching for Linked Data Structures]

- Erweiterung der RPT
- Regelmäßigkeit in Code statt in Werten
- CT-Einträge: Producer-Consumer-Paare von Loads, Address generation template
  - (addr, addr, lw 4($))
- Speedup 10% für Pointer-Benchmark

2001: Greedy PF und Jump-Pointer in Java

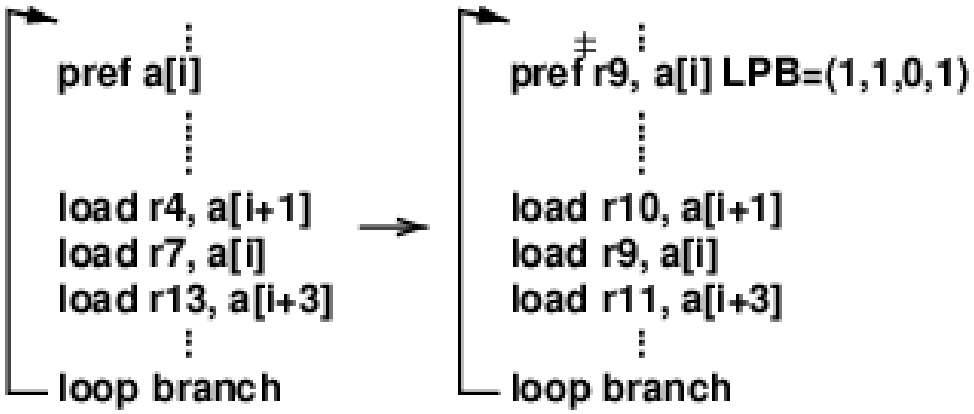
[Cahoon, McKinley: Data Flow Analysis for Software Prefetching Linked Data Structures in Java]

- 15-95% Memory Stall für Olden Benchmarks
- Recurrent analysis: `o = o.next`
  - reaching definitions + definition-use chains für Felder
- Greedy prefetching: 4% Speedup
- Jump-Pointer bei Objekt-Erzeugung einfügen: 10% Speedup
  - Jump queue wie in [Luk96]
  - funktioniert nur bei gleicher Objekt-Traversierung
- noch keine Garbage collection

2001: Register Renaming

[Ortega, Valero, Ayguade: A novel renaming mechanism that boosts software prefetching]

- erweitert vorhandenes Register Renaming / Prefetching
- Load = *binding* prefetch



- arbeitet auf optimiertem Code mit Prefetching
- orthogonale Verbesserung von Prefetching
- numerische Applikationen: 5-22% Speedup

2002: Pre-Execution

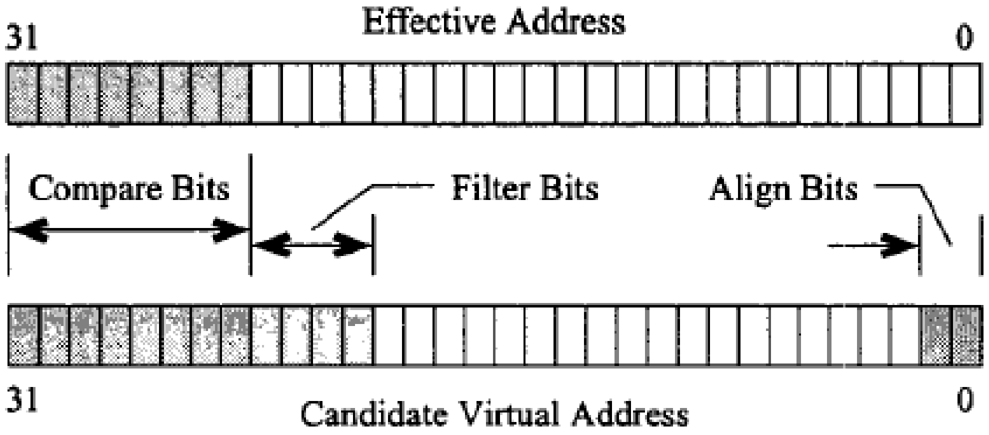
[Kim, Yeung: Design and evaluation of compiler algorithms for pre-execution]

- eigener HW-Thread führt Pre-execution Code spekulativ aus
  - Profiling: Slices finden
- 17% Speedup
- funktioniert für bel. Zugriffsmuster
- benötigt Threading-HW
- bisher nur für C, keine rekursiven Baum-Durchläufe

2002: Cache-Snooping

[Cooksey, Jourdan, Grunwald: A stateless, content-directed data prefetching mechanism]

- Finde Daten, die Pointer sein könnten



- Rekursiv mit Tiefen-Beschränkung: 11% Speedup ggüber Pentium 4 mit Prefetching
- Prefetch-Tiefe im Cache ($\frac{1}{2}$% Overhead): 13% Speedup
- Daten sind meist rechtzeitig da

Prozessoren

- alle: Cache-Lines
- HP PA7200: tagged prefetching in hardware
- HP PA8000: software
- UltraSparc III: Prefetch pipeline neu, buggy (4/2001)
- AMD: Hardware pre-fetch neu im Palomino (10/2001)
- Itanium: Software hints: fetch “few” or “many” lines
- GCC: manche Targets, `-fprefetch-loop-arrays`

2003: Abstrakte Interpretation?

- benutze unsere Cache-Analyse für PowerPC um Cache-Misses zu finden
- Prefetches an geeigneten Stellen einfügen
- arbeitet auf CRL-Ebene (gcc, exec2crl, crl2asm)
- Ergebnisse . . . demnächst

Prefetching: Schlußwort

[Oren 2000]:

- I-Prefetching $\longrightarrow$ Branch prediction
- D-Prefetching: noch offen
  - “an understudied but potentially very beneficial area of research”