Copyright: SiliconGraphics

The Memory Hierarchy in modern RISC based SMP systems

Ruud van der Pas

SiliconGraphics

European High Performance Computing Team

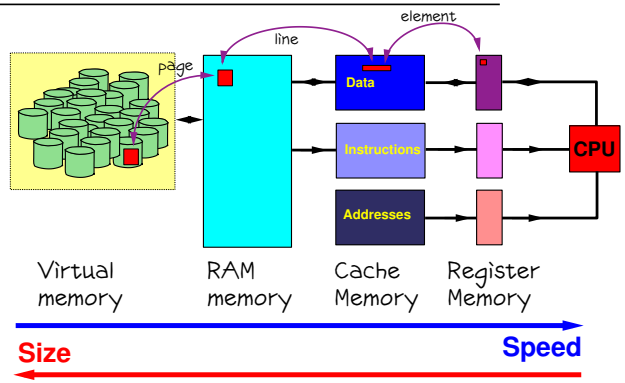
Copyright: SiliconGraphics

Introduction

- *In modern computer systems, a complete hierarchy in memory is used*
- *The reason for using this hierarchy is:* ***cost***
- *When programming for such a hierarchy it is good to know about its existence and main characteristics*
- *This will be the topic of this talk*

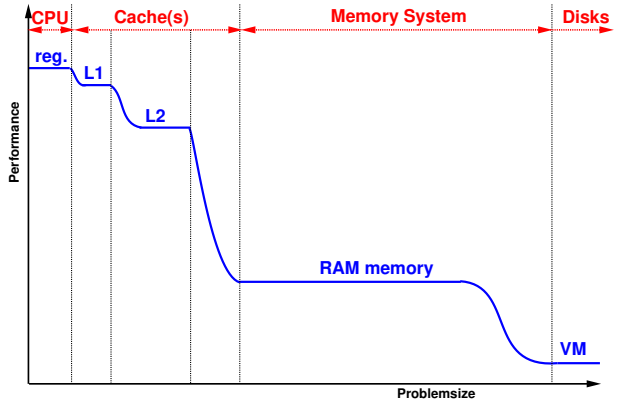
Copyright: SiliconGraphics

The Memory Hierarchy



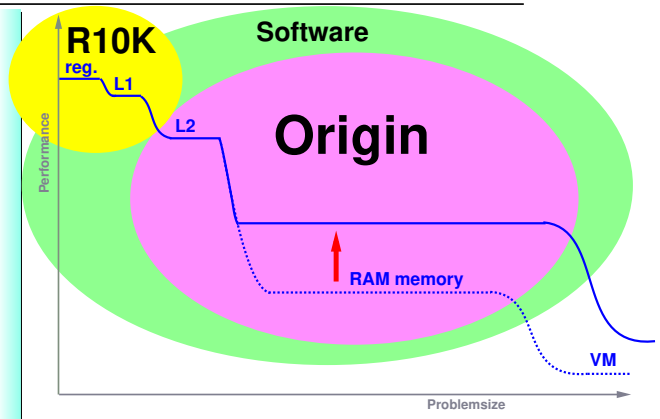
Copyright: SiliconGraphics

Memory Hierarchy Performance



Copyright: SiliconGraphics

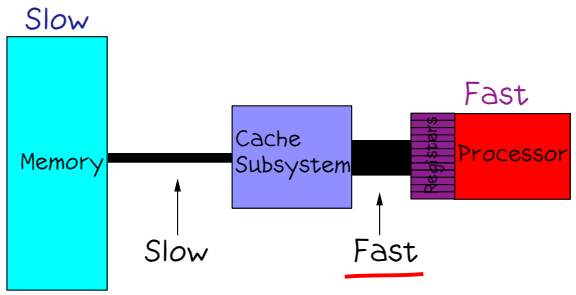
Memory Hierarchy Performance



Copyright: SiliconGraphics

Caches

Caches play an important role in performance



Note:
The R10000 processor has non-blocking caches to overlap computation and memory access

Copyright: SiliconGraphics

About Caches

Typical Problem:

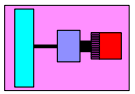
A cache is not big enough to store all data

Question to be answered:

How do we replace data?

This is called the cache replacement policy

Different approaches will be discussed next



Copyright: SiliconGraphics

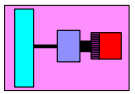
Direct Mapped Caches

Replacement Formula:

cache location = memory address modulo size
(the memory address can be virtual or physical)

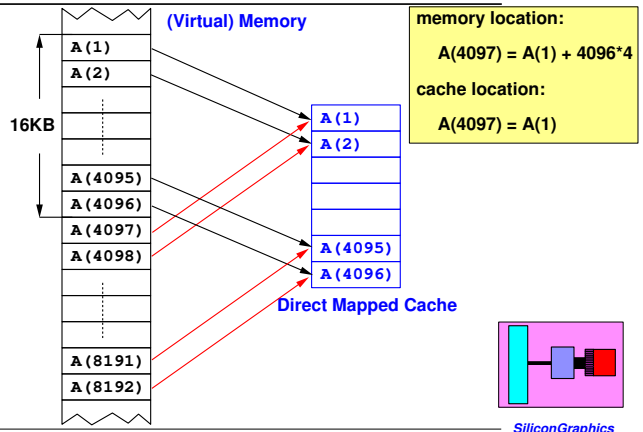
An example:

- ***Cache size = 16 KByte = 16*1024 bytes***
- ***This corresponds to 4*1024 = 4096 32-bit words***
- ***We load a 32-bit array A with 8192 elements***



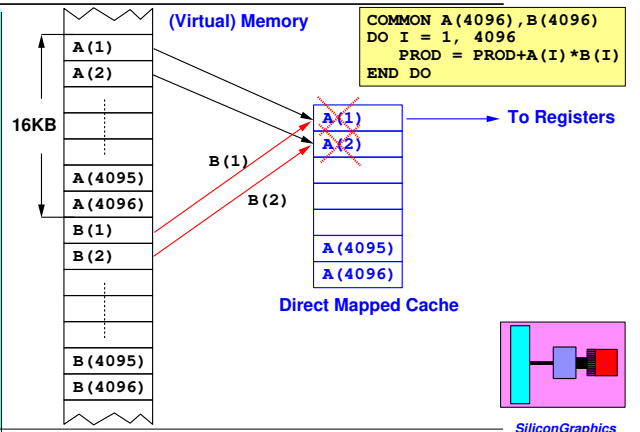
Copyright: SiliconGraphics

Direct Mapped Caches – Trashing



Copyright: SiliconGraphics

Direct Mapped Caches – Two arrays



Copyright: SiliconGraphics

Direct Mapped Caches

- A well–known side–effect of this design: *data elements that are soon needed are overwritten ("trashing")*
- Especially when multiple arrays are involved, direct mapping can become very inefficient
- Often the only remedy is to modify the memory mapping, but this can be non–trivial
- The design discussed next solves this trashing problem

Copyright: SiliconGraphics

Fuly Associative Caches

Fully Associative

- The replacement is based upon a Least–Recently Used (LRU) algorithm:
- Data that is oldest, is removed
- In many cases this makes sense to do
- It greatly helps when working with multiple arrays

Copyright: SiliconGraphics

Fuly Associative Caches

Why are not all caches then Fully Associative?

BECAUSE OF COST

Luckily, a clever alternative exists

Copyright: SiliconGraphics

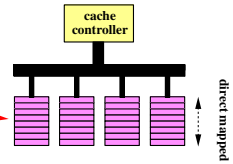
Set Associative Caches

Set Associative

- The cache contains several direct mapped caches
- Data can go into one of these caches (called a "set")
- The choice of a set is often (semi–) LRU

Example:

4–way set associative cache



Trashing can still occur within one set !

Copyright: SiliconGraphics

Set Associative Caches

Set Associative

- Relatively cost–effective solution
- 2, 3 or 4 sets are most common
- Avoids many of the trashing issues found in direct mapped caches
- Trashing can still occur within one set

Copyright: SiliconGraphics

Summary Cache Replacement

Microprocessors often use a mixture of:

- Direct Mapped
- Set Associative
- Fully Associative

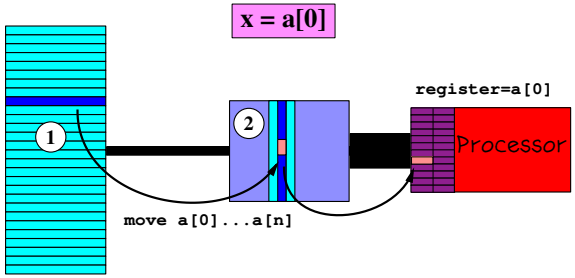
The choice "what is used where?" is determined by:

- Expected usage
- Price/performance

Copyright: SiliconGraphics

Cache Lines

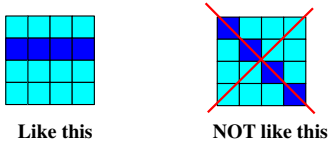
Typically more than one element at once is transferred



Copyright: SiliconGraphics

Cache Lines

- The unit of transfer is called a cache line
- A cache line consists of consecutive memory locations



- The size of a cache line is architecture dependent

Copyright: SiliconGraphics

The TLB Cache

- In a virtual memory based system, the *virtual address* needs to be translated to a *physical address*
- This address translation is typically a *costly* operation
- Therefore, translations are:
  - Performed on a virtual memory *page* basis
  - Buffered in a cache (with the hope to re–use them)
- This cache is often called Translation Lookaside Buffer or *TLB* for short

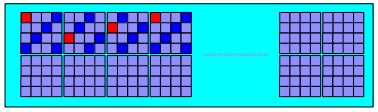
Copyright: SiliconGraphics

The TLB Cache

Bad for the TLB:

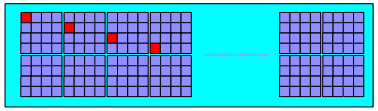
= TLB entry created

Non–unit stride through the data



VERY bad for the TLB

Strides through data which exceed the page size



Copyright: SiliconGraphics

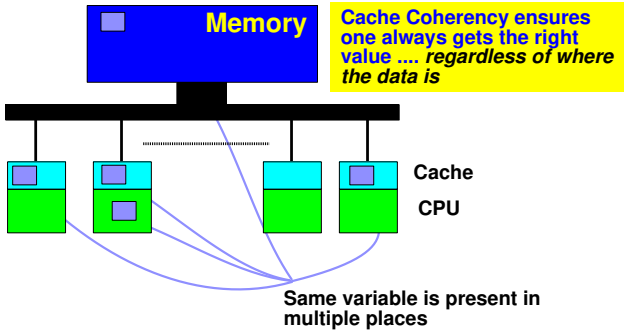
Cache Performance

Rules of thumb for optimal cache performance:

1. Re–use data in the cache
   - This is also called temporal locality
2. Use all the data in one cache line
   - This is also called spatial locality

Copyright: SiliconGraphics

Cache Coherency



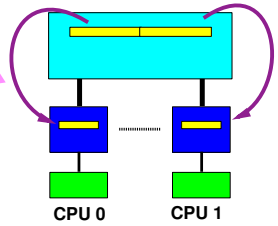
Cache Coherency ensures one always gets the right value .... *regardless of where the data is*

Copyright: SiliconGraphics

Cache Coherency

In a parallel program, data easily can be "anywhere"

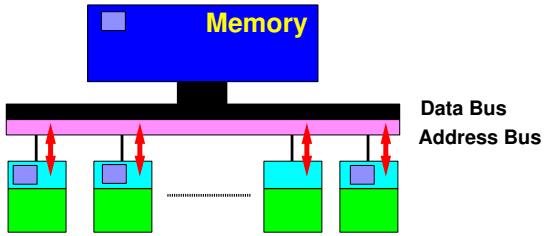
```
for (i=0; i<N; i++)
  a[i] = a[i] + i;

    ......

j = func(m);

    ......

b = a[j];
```



Where in the system is a[j]?

Copyright: SiliconGraphics

Cache Coherency – Snooping



With a snooping protocol, *ALL* address traffic on the bus is monitored by *ALL* processors

This does not scale very well (= quadraticly ...)

Copyright: SiliconGraphics

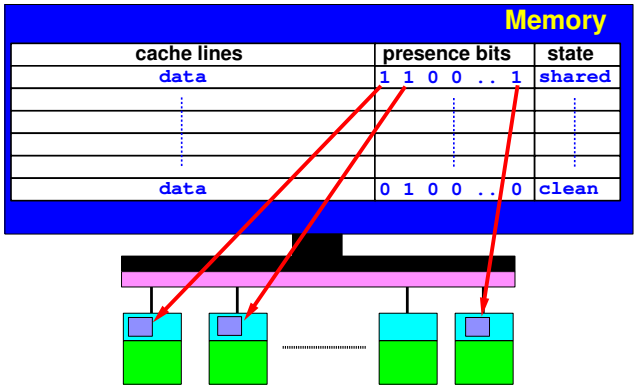
Cache Coherency – Directories

Directory based cache coherency:

- *Keeps track of which CPU(s) the cache line has (have) and what the state is*
- *More scalable coherency architecture*
- *Only processors who access a particular cache line are involved with the coherency for that line*
- *It does cost extra logic and memory though*

Copyright: SiliconGraphics

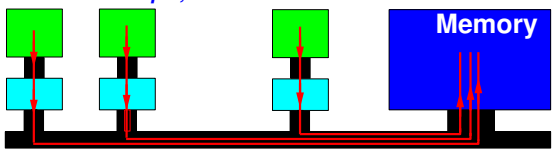
Cache Coherency – Directories



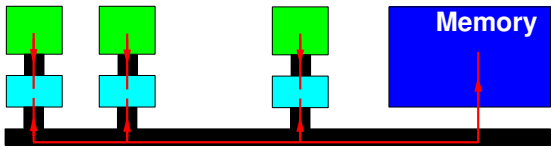
Copyright: SiliconGraphics

Cache Coherency

Write–Through
Simple, but wastes bandwidth



Write–Back
Minimizes bandwidth, takes extra logic



Copyright: SiliconGraphics

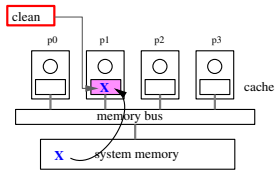
MP Cache Coherency

A cache line has 4 possible states in the write–back protocol:

1. *clean*
   It is the only cached copy
2. *shared*
   It resides in multiple caches
3. *dirty*
   It has been modified and should be written to main memory before it is overwritten.
4. *invalid*
   No longer valid. The most recent copy of the cache line is elsewhere.

Copyright: SiliconGraphics

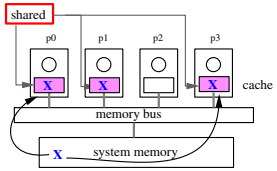
Cache Line States



- Processor P1 has loaded X into it's cache
- Nobody else has a copy, so it is "clean"

Copyright: SiliconGraphics

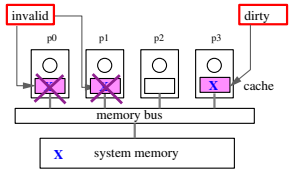
Cache Line States



- Also Processors P0 and P1 have loaded X into their cache
- The cache line containing X is now in the state "shared"

Copyright: SiliconGraphics

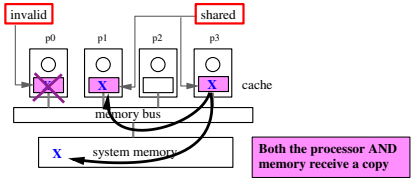
Cache Line States



- Suppose that Processor P3 now modifies X
- The cache line in P3 is now in the state "dirty"
- The cache lines in P0 and P1 are marked "invalid" first

Copyright: SiliconGraphics

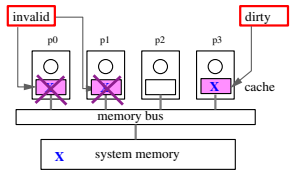
Cache Line States



- A modified ("dirty") cache line will not be written back to memory unless:
  - It's location is needed for other data
  - Another processor needs the same data (like P1 here)

Copyright: SiliconGraphics

False Sharing of Cache Lines

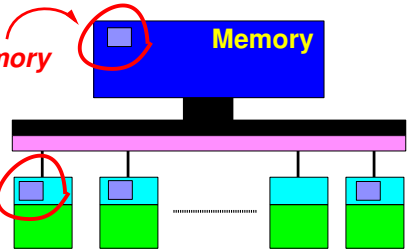


- A modified cache line, is marked "dirty"
- All processors that share this line, must invalidate their copy
- Even if they don't use the variables in the line ...
- This is called "false sharing"

Copyright: SiliconGraphics

NUMA=Non–Uniform Memory Access

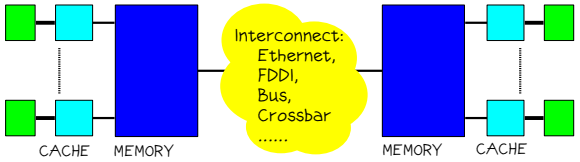
Data in the memory is further away from the CPU than data in the cache



This means the Memory Access time is ..... Non–Uniform

Copyright: SiliconGraphics

NUMA=Non–Uniform Memory Access



- *NUMA extends to the memory level*
- *The real differentiators are ...*
  - *Latency and bandwidth*
  - *Programming model*

Copyright: SiliconGraphics

Memory – Summary

- *The memory hierarchy plays a crucial role in the performance of any system*
- *Especially the size and structure of the cache(s) is an important factor*
- *RISC optimisation is often about tuning for the memory hierarchy*