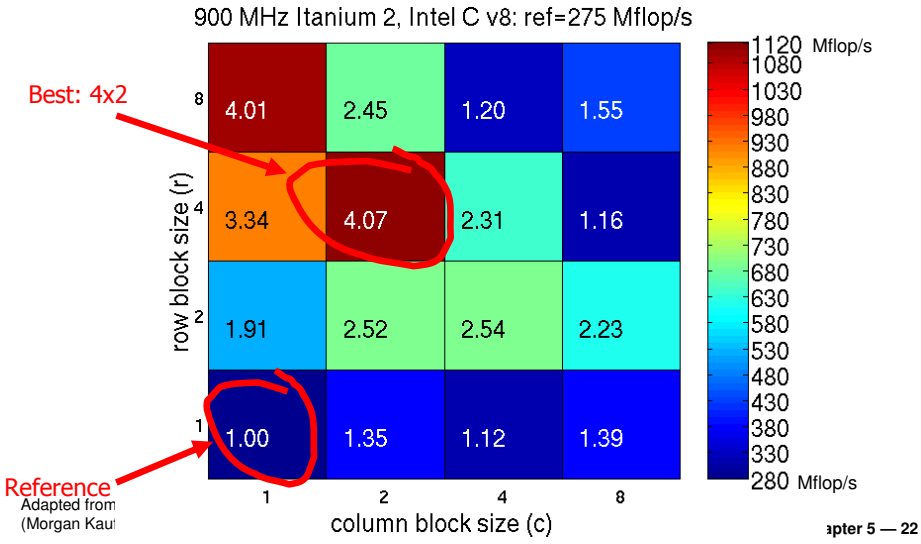
Compiler Optimization vs. Memory Hierarchy Search

- Compiler tries to figure out memory hierarchy optimizations
- New approach: "Auto-tuners" 1st run variations of program on computer to find best combinations of optimizations (blocking, padding, …) and algorithms, then produce C code to be compiled for *that* computer
- "Auto-tuner" targeted to numerical method
  - E.g., PHiPAC (BLAS), Atlas (BLAS), Sparsity (Sparse linear algebra), Spiral (DSP), FFT-W

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

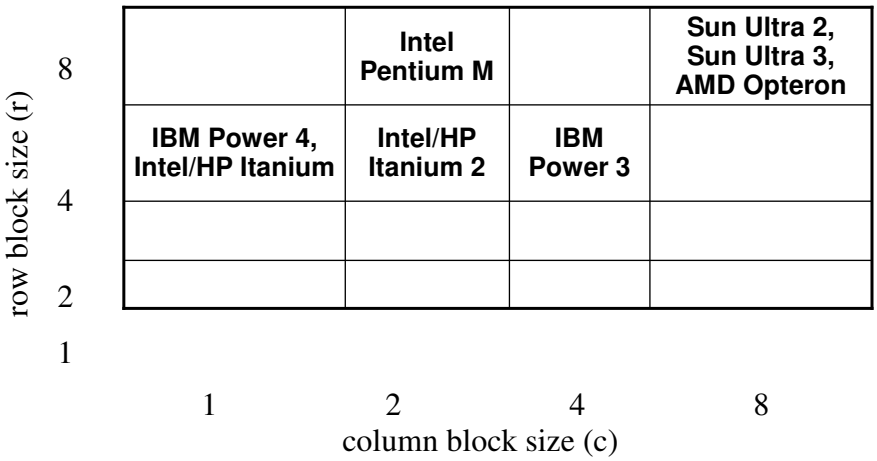
Sparse Matrix – Search for Blocking

for finite element problem [Im, Yelick, Vuduc, 2005]



Adapted from
(Morgan Kau

Best Sparse Blocking for 8 Computers



- All possible column block sizes selected for 8 computers; How could compiler know?

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Technique	Hit Time	Band-width	Miss penalty	Miss rate	HW cost/ complexity	Comment
Small and simple caches	+			–	0	Trivial; widely used
Way-predicting caches	+				1	Used in Pentium 4
Trace caches	+				3	Used in Pentium 4
Pipelined cache access	–	+			1	Widely used
Nonblocking caches		+	+		3	Widely used
Banked caches		+			1	Used in L2 of Opteron and Niagara
Critical word first and early restart			+		2	Widely used
Merging write buffer			+		1	Widely used with write through
Compiler techniques to reduce cache misses				+	0	Software is a challenge; some computers have compiler option
Hardware prefetching of instructions and data			+	+	2 instr., 3 data	Many prefetch instructions; AMD Opteron prefetches data
Compiler-controlled prefetching			+	+	3	Needs nonblocking cache; in many CPUs

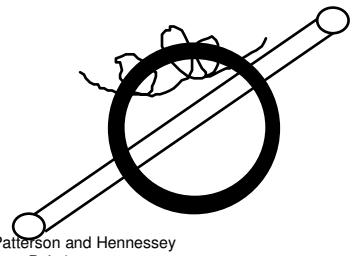
Main Memory Background

- Performance of Main Memory:
  - Latency: Cache Miss Penalty
    - *Access Time*: time between request and word arrives
    - *Cycle Time*: time between requests
  - Bandwidth: I/O & Large Block Miss Penalty (L2)
- Main Memory is *DRAM*: Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms, 1% time)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - *RAS* or *Row Access Strobe*
    - *CAS* or *Column Access Strobe*
- Cache uses *SRAM*: Static Random Access Memory
  - No refresh (6 transistors/bit vs. 1 transistor
    Size: DRAM/SRAM - *4-8*,
    Cost/Cycle time: SRAM/DRAM - *8-16*

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

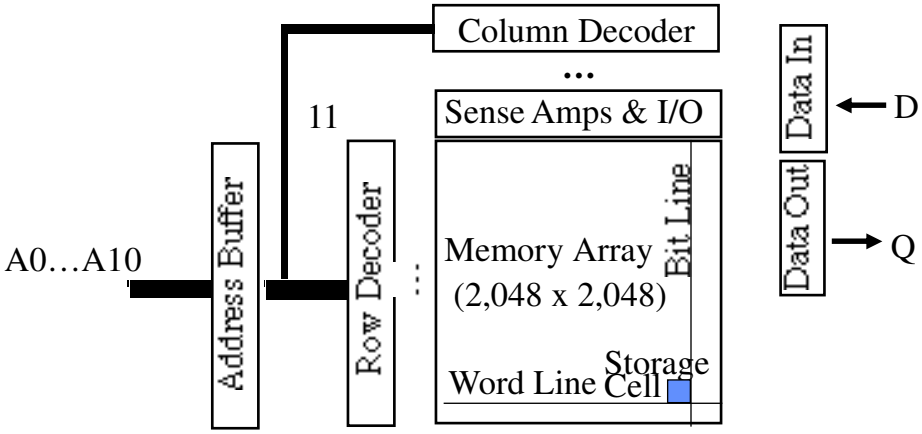
Main Memory Deep Background

- "Out-of-Core", "In-Core," "Core Dump"?
- "Core memory"?
- Non-volatile, magnetic
- Lost to 4 Kbit DRAM (today using 512Mbit DRAM)
- Access time 750 ns, cycle time 1500-3000 ns



Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

DRAM logical organization (4 Mbit)



- **Square root of bits per RAS/CAS**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Quest for DRAM Performance

1. **Fast Page mode**
   - **Add timing signals that allow repeated accesses to row buffer without another row access time**
   - **Such a buffer comes naturally, as each array will buffer 1024 to 2048 bits for each access**
2. **Synchronous DRAM (SDRAM)**
   - **Add a clock signal to DRAM interface, so that the repeated transfers would not bear overhead to synchronize with DRAM controller**
3. **Double Data Rate (DDR SDRAM)**
   - **Transfer data on both the rising edge and falling edge of the DRAM clock signal ⇒ doubling the peak data rate**
   - **DDR2 lowers power by dropping the voltage from 2.5 to 1.8 volts + offers higher clock rates: up to 400 MHz**
   - **DDR3 drops to 1.5 volts + higher clock rates: up to 800 MHz**

- **Improved Bandwidth, not Latency**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

DRAM name based on Peak Chip Transfers / Sec
DIMM name based on Peak DIMM MBytes / Sec

Fastest for sale 4/06 ($125/GB)

Stan-dard	Clock Rate (MHz)	M transfers / second	DRAM Name	Mbytes/s/ DIMM	DIMM Name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800

x 2 x 8

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Need for Error Correction!

- **Motivation:**
  - **Failures/time *proportional* to number of bits!**
  - **As DRAM cells shrink, more vulnerable**
- **Went through period in which failure rate was low enough without error correction that people didn't do correction**
  - **DRAM banks too large now**
  - **Servers always corrected memory systems**
- **Basic idea: add redundancy through parity bits**
  - **Common configuration: Random error correction**
    - » **SEC-DED (single error correct, double error detect)**
    - » **One example: 64 data bits + 8 parity bits (11% overhead)**
  - **Really want to handle failures of physical components as well**
    - » **Organization is multiple DRAMs/DIMM, multiple DIMMs**
    - » **Want to recover from failed DRAM and failed DIMM!**
    - » **"Chip kill" handle failures width of single DRAM chip**

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Introduction to Virtual Machines

- VMs developed in late 1960s
  - Remained important in mainframe computing over the years
  - Largely ignored in single user computers of 1980s and 1990s
- Recently regained popularity due to
  - increasing importance of isolation and security in modern systems,
  - failures in security and reliability of standard operating systems,
  - sharing of a single computer among many unrelated users,
  - and the dramatic increases in raw speed of processors, which makes the overhead of VMs more acceptable

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

What is a Virtual Machine (VM)?

- Broadest definition includes all emulation methods that provide a standard software interface, such as the Java VM
- "(Operating) System Virtual Machines" provide a complete system level environment at binary ISA
  - Here assume ISAs always match the native hardware ISA
  - E.g., IBM VM/370, VMware ESX Server, and Xen
- Present illusion that VM users have entire computer to themselves, including a copy of OS
- Single computer runs multiple VMs, and can support a multiple, different OSes
  - On conventional platform, single OS "owns" all HW resources
  - With a VM, multiple OSes all share HW resources
- Underlying HW platform is called the host, and its resources are shared among the guest VMs

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Virtual Machine Monitors (VMMs)

- Virtual machine monitor (VMM) or hypervisor is software that supports VMs
- VMM determines how to map virtual resources to physical resources
- Physical resource may be time-shared, partitioned, or emulated in software
- VMM is much smaller than a traditional OS;
  - isolation portion of a VMM is ≈ 10,000 lines of code

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

VMM Overhead?

- Depends on the workload
- User-level processor-bound programs (e.g., SPEC) have zero-virtualization overhead
  - Runs at native speeds since OS rarely invoked
- I/O-intensive workloads ⇒ OS-intensive
  ⇒ execute many system calls and privileged instructions
  ⇒ can result in high virtualization overhead
  - For System VMs, goal of architecture and VMM is to run almost all instructions directly on native hardware
- If I/O-intensive workload is also I/O-bound
  ⇒ low processor utilization since waiting for I/O
  ⇒ processor virtualization can be hidden
  ⇒ low virtualization overhead

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Other Uses of VMs

- Focus here on protection
- 2 Other commercially important uses of VMs

1. Managing Software
   - VMs provide an abstraction that can run the complete SW stack, even including old OSes like DOS
   - Typical deployment: some VMs running legacy OSes, many running current stable OS release, few testing next OS release
2. Managing Hardware
   - VMs allow separate SW stacks to run independently yet share HW, thereby consolidating number of servers
     - » Some run each application with compatible version of OS on separate computers, as separation helps dependability
   - Migrate running VM to a different computer
     - » Either to balance load or to evacuate from failing HW

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Requirements of a Virtual Machine Monitor

- A VM Monitor
  - Presents a SW interface to guest software,
  - Isolates state of guests from each other, and
  - Protects itself from guest software (including guest OSes)
- Guest software should behave on a VM exactly as if running on the native HW
  - Except for performance-related behavior or limitations of fixed resources shared by multiple VMs
- Guest software should not be able to change allocation of real system resources directly
- Hence, VMM must control ≈ everything even though guest VM and OS currently running is temporarily using them
  - Access to privileged state, Address translation, I/O, Exceptions and Interrupts, …

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Requirements of a Virtual Machine Monitor

- VMM must be at higher privilege level than guest VM, which generally run in user mode
  - ⇒ Execution of privileged instructions handled by VMM
- E.g., Timer interrupt: VMM suspends currently running guest VM, saves its state, handles interrupt, determine which guest VM to run next, and then load its state
  - Guest VMs that rely on timer interrupt provided with virtual timer and an emulated timer interrupt by VMM
- Requirements of system virtual machines are ≈ same as paged-virtual memory:

1. At least 2 processor modes, system and user
2. Privileged subset of instructions available only in system mode, trap if executed in user mode
   - All system resources controllable only via these instructions

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

ISA Support for Virtual Machines

- If VMs are planned for during design of ISA, easy to reduce instructions that must be executed by a VMM and how long it takes to emulate them
  - Since VMs have been considered for desktop/PC server apps only recently, most ISAs were created without virtualization in mind, including 80x86 and most RISC architectures
- VMM must ensure that guest system only interacts with virtual resources ⇒ conventional guest OS runs as user mode program on top of VMM
  - If guest OS attempts to access or modify information related to HW resources via a privileged instruction--for example, reading or writing the page table pointer--it will trap to the VMM
- If not, VMM must intercept instruction and support a virtual version of the sensitive information as the guest OS expects (examples soon)

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Impact of VMs on Virtual Memory

- Virtualization of virtual memory if each guest OS in every VM manages its own set of page tables?
- VMM separates real and physical memory
  - Makes real memory a separate, intermediate level between virtual memory and physical memory
  - Some use the terms virtual memory, physical memory, and machine memory to name the 3 levels
  - Guest OS maps virtual memory to real memory via its page tables, and VMM page tables map real memory to physical memory
- VMM maintains a shadow page table that maps directly from the guest virtual address space to the physical address space of HW
  - Rather than pay extra level of indirection on every memory access
  - VMM must trap any attempt by guest OS to change its page table or to access the page table pointer

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

ISA Support for VMs & Virtual Memory

- IBM 370 architecture added additional level of indirection that is managed by the VMM
  - Guest OS keeps its page tables as before, so the shadow pages are unnecessary
- To virtualize software TLB, VMM manages the real TLB and has a copy of the contents of the TLB of each guest VM
  - Any instruction that accesses the TLB must trap
  - TLBs with Process ID tags support a mix of entries from different VMs and the VMM, thereby avoiding flushing of the TLB on a VM switch

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Impact of I/O on Virtual Memory

- **Most difficult part of virtualization**
  - **Increasing number of I/O devices attached to the computer**
  - **Increasing diversity of I/O device types**
  - **Sharing of a real device among multiple VMs,**
  - **Supporting the myriad of device drivers that are required, especially if different guest OSes are supported on the same VM system**
- **Give each VM generic versions of each type of I/O device driver, and let VMM to handle real I/O**
- **Method for mapping virtual to physical I/O device depends on the type of device:**
  - **Disks partitioned by VMM to create virtual disks for guest VMs**
  - **Network interfaces shared between VMs in short time slices, and VMM tracks messages for virtual network addresses to ensure that guest VMs only receive their messages**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)