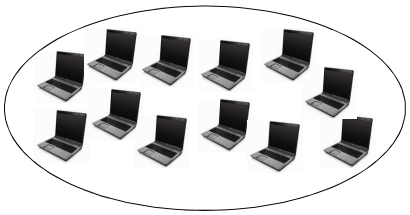
More than 30 Years !

Parallel computing has been an every-day tool for solving computational problems in science and technology since the mid-80´s

- Parallel algorithms
- Tools for parallel programming
- Parallel codes
  - Message passing (MPI) in the 80´s
  - Shared memory (OpenMP) in the 90´s
  - Accelerators, many-core, exascale,…
- Parallel performance analysis
- Parallel computer systems
  - For a long time: 10-1000 processors

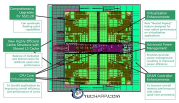
Parallel Computing Everywhere…

Today, all computers are parallel computers !



Multi/manycore Processors

The Multi/manycore Era is here!



The introduction of multicore processors and accelerators has a profound impact on computing

Individual processor [core] costs will drop so quickly that in the near future the world can view processors as a nearly free commodity [IDC presentation on HPC at International Supercomputing Conference, ICS07, Dresden, June 26-29 2007]

Classical microprocessors: W…er it takes to run one progra…

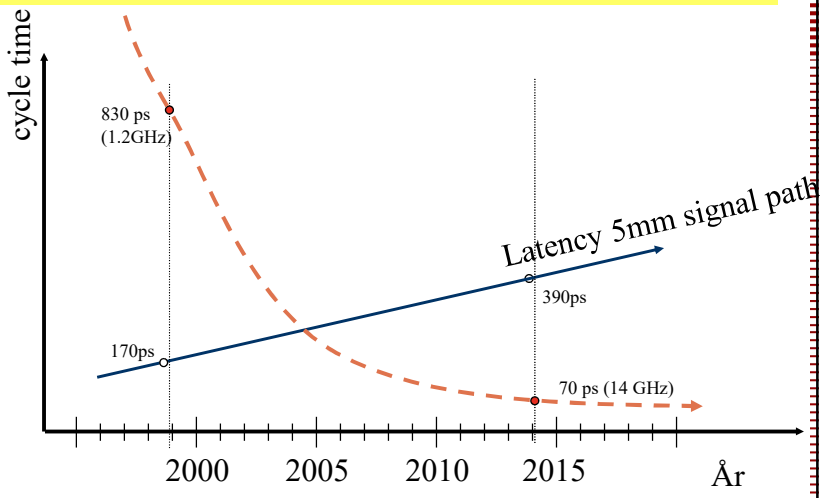
Exploring ILP (instruction-level …

- Faster clocks → Deep pip…
- Superscalar Pipelines
- Branch Prediction
- Out-of-Order Ex…
- Trace Cache
- Speculati…
- Predica…
- Adv… …ess Table
- R… …tack
-

Bad News #1:
We have already explored most ILP
(instruction-level parallelism)

2009-08-17

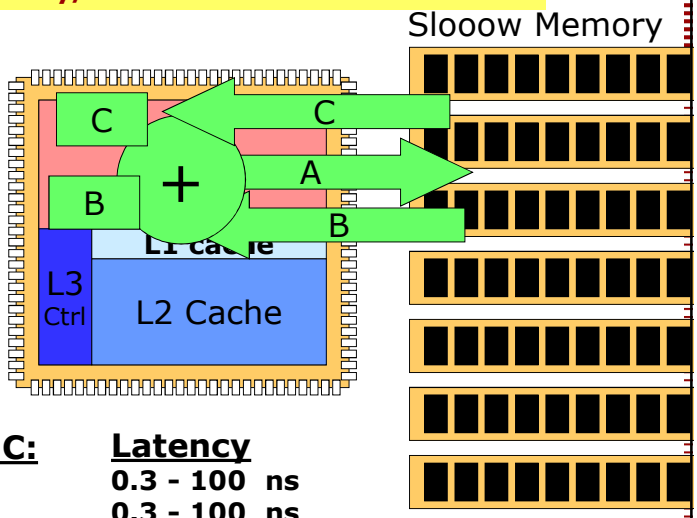
Bad News #2
Looong wire delay ➔slow CPUs



Quantitative data and trends according to V. Agarwal et al., ISCA 2000
***Based on SIA* (Semiconductor Industry Association) prediction, 1999**

2009-08-17

Bad News #3:
Memory latency/bandwidth is the bottleneck…



A = B + C:	**Latency**
Read B	**0.3 - 100 ns**
Read C	**0.3 - 100 ns**
Add B & C	**0.3 ns**
WriteA	**0.3 - 100 ns**

2009-08-17

Bad News #4:
Power is the limit

- Power consumption is the bottleneck
  - Cooling servers is hard
  - Battery lifetime for mobile computers
  - Energy is money
- Dynamic effect is proportional to
  - ~ Frequency
  - ~ Voltage^2

2009-08-17

Why multiple cores?

- Instruction level parallelism is running out
  - Feed the CPU with instructions from many threads (Simultaneous MultiThreading, SMT)
- Wire delay is hurting now
  - Use the chip for multiple smaller CPU cores (Chip MultiProcessors, CMP)
- Power is a limit now
  - Lower the frequency (which enables lower voltage)
- Memory latency/bandwidth bottleneck
  - Access the memory from many threads

2009-08-17

Parallel ComputingTerminology

Node / CPU / Socket / Processor / Core
Task
Pipelining
Shared Memory (architecture and programming model)
SMP / UMA / NUMA (architecture and programming model)
Distributed Memory (architecture and programming model)
Communication
Synchronization
Granularity
Parallel Overhead
Massively Parallel
Embarrassingly Parallel

Terminology/Concepts cont.

Observed Speedup: wall-clock time of serial execution/wall-clock time of parallel execution

Amdahl's Law: max speedup = 1/(1 – P)

Complexity (Design/Coding/Debugging/Tuning/Maintenance)

Resource Requirements

Scalability (observed, strong, weak)

Parallel Computer Architechtures

Shared Memory

Ability for all processors to access all memory as global address space. Multiple processors can operate independently but share the same memory resources. Changes in a memory location effected by one processor are visible to all other processors (caching systems!)

Uniform Memory Access (UMA): Equal access and access times to memory. CC-UMA - Cache Coherent UMA.

Non-Uniform Memory Access (NUMA): Not all processors have equal access time to all memories. If cache coherency is maintained: CC-NUMA - Cache Coherent NUMA

Virtual Shared Memory Access: Use caching techniques and software to create a VSM (Hagersten et al.)

Parallel Computer Architechtures

Distributed Memory

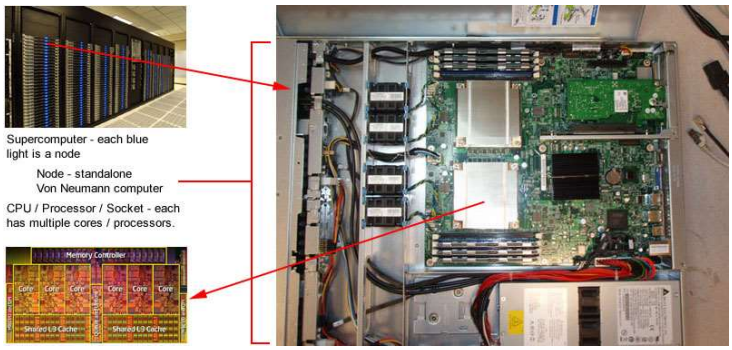
Processors have their own local memory. Memory addresses in one processor do not map to another processor. There is no concept of global address space across all processors. Distributed memory systems require a communication network to connect inter-processor memory.

When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.

The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet and even “the internet”

Parallel Computer Architechtures

The “standard supercomputer” today: Hybrid!



2009-08-17

Parallel Computer Architechtures

Shared Memory

Global address space provides a user-friendly programming perspective to memory. Potentially lacks of scalability between memory and CPUs. Programmer is responsible for synchronization constructs that ensure "correct" access of global memory.

Distributed Memory

Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately. Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency.

The programmer is responsible for many of the details associated with data communication between processors. It may be difficult to map existing data structures, based on global memory, to this memory organization.

2009-08-17

Parallel Programming Models

Examples

- Shared Memory (with or without threads)
- Data Parallel (“PGAS” or “SIMD”)
- Distributed Memory / Message Passing
- Task Parallel (Function Parallelism) – “tightly coupled” (Chunks and Tasks) or “loosely coupled” (Hadoop, MapReduce)
- Hybrids …

Abstraction above hardware and memory architectures!

Models are NOT specific to a particular type of machine or memory architecture!

Virtual Shared Memory – SM model on DM architecture

Message Passing on UMA/NUMA – DM model on SM arch.

No "best" model, although there certainly are better implementations of some models over others.

2009-08-17

Thread Programming Models

A type of shared memory programming. A single "heavy weight" process can have multiple "light weight", concurrent execution paths.

The main program a.out is scheduled to run by the native operating system. a.out loads and acquires all of the necessary system and user resources to run. This is the "heavy weight" process.

a.out performs some serial work, and then creates a number of tasks (threads) that can be scheduled and run by the operating system concurrently.

Each thread has local data, but also, shares the entire resources of a.out. This saves the overhead associated with replicating a program's resources for each thread ("light weight"). Each thread also benefits from a global memory view because it shares the memory space of a.out.

2009-08-17

Thread Programming Models

Threads communicate with each other through global memory (updating address locations). This requires synchronization constructs to ensure that more than one thread is not updating the same global address at any time.

Threads can come and go, but a.out remains present to provide the necessary shared resources until the application has completed.

Implementations:

A library of subroutines

A set of compiler directives imbedded in the source code

The programmer is responsible for determining the parallelism (although compilers can sometimes help).

Thread Programming Models

- POSIX Threads. Part of Unix/Linux operating systems
- OpenMP. Industry standard, jointly defined and endorsed by a group of major computer hardware and software vendors, organizations and individuals. Compiler directive based. Can be very easy and simple to use - provides for "incremental parallelism". Can begin with serial code.
- Microsoft threads
- Java, Python threads
- CUDA threads for GPUs

Data Parallel Models (PGAS)

Address space is treated globally. Most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.

On shared memory architectures, all tasks may have access to the data structure through global memory.

On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks.

- Unified Parallel C (UPC): an extension to the C programming language for SPMD parallel programming.
- Global Arrays: provides a shared memory style programming environment in the context of distributed array data structures.
- Chapel: Open source parallel programming language project.

2009-08-17

Distributed Memory Models

Distributed Memory / Message Passing Model

A set of tasks that use their own local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines. Tasks exchange data through communications by sending and receiving messages.

Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.

Implementations:

A library of subroutines. Calls to these subroutines are imbedded in source code. The programmer is responsible for determining all parallelism.

MPI is the "de facto" industry standard for message passing. Not all implementations include everything in MPI-1, MPI-2 or MPI-3.

2009-08-17

Task Parallelism

Each processor executes a different thread (or process) on the same or different data. The threads may execute the same or different code. In the general case, different execution threads communicate with one another as they work. Communication usually takes place by passing data from one thread to the next as part of a workflow.

Task parallelism emphasizes the distributed (parallelized) nature of the processing (i.e. threads), as opposed to the data (data parallelism).

- Ada: Tasks (built-in)
- C++ (Intel): Threading Building Blocks
- C++ (Intel): Cilk Plus
- Java: Java concurrency

2009-08-17

Task Parallelism

Hadoop - Framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. Designed to detect and handle failures at the application layer. Can be used on top of a cluster of computers, each of which may be prone to failures.

Hadoop MapReduce - Software framework for processing of (large) data sets in-parallel on clusters of computers.

Splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

2009-08-17