Trends and Challenges in Multicore Programming

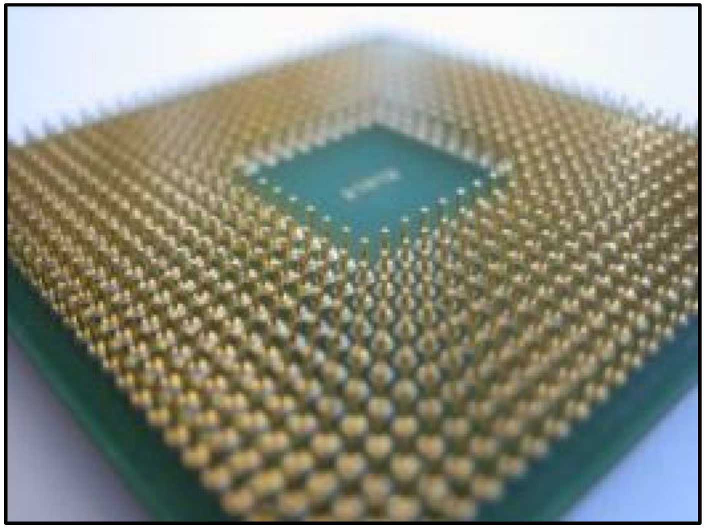
Eva Burrows
Bergen Language Design Laboratory (BLDL)
Department of Informatics, University of Bergen

Bergen, March 17 , 2010

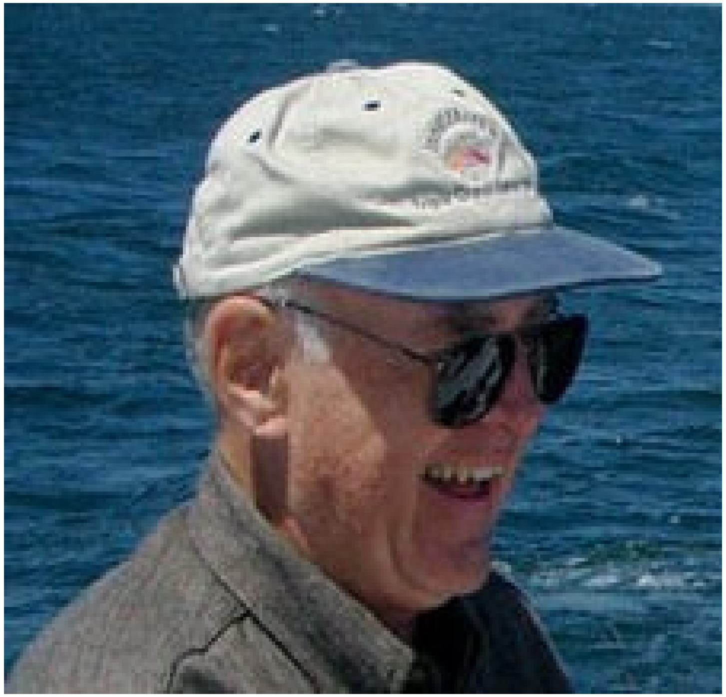
Outline

- The Roadmap of Multicores
- The Challenge of Parallel Thinking
- Multicore Languages and Compilers
- Summary

The Roadmap of Multicores



At the beginnings...



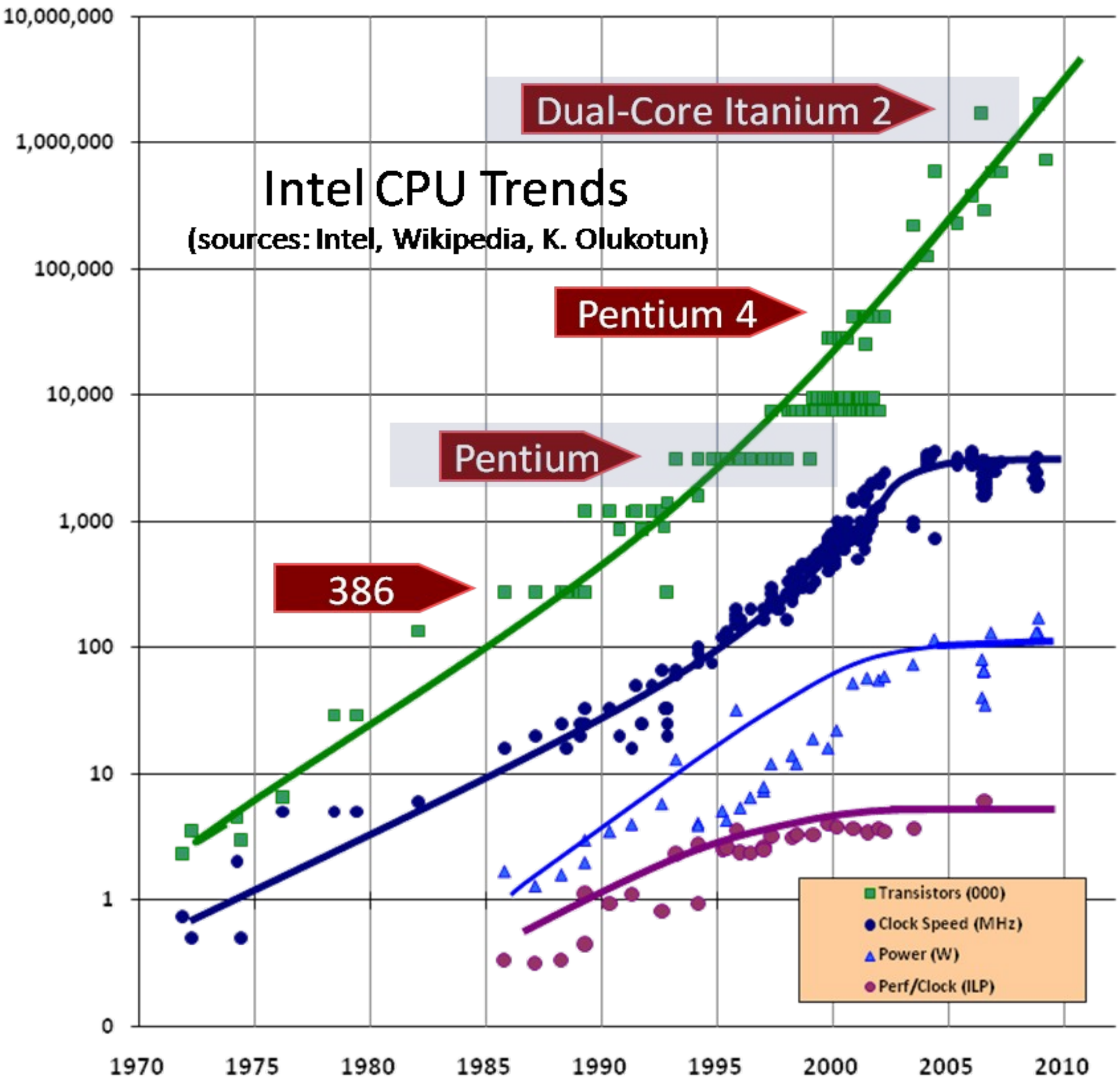
Gordon E. Moore
Co-founder and Chairman Emeritus of Intel

1965:

> *“the number of transistors placed inexpensively on integrated circuit will double approximately every two years “*

- Innocent observation led to an industry goal: *Moore's Law*

Moore's Law illustrated on Intel chips

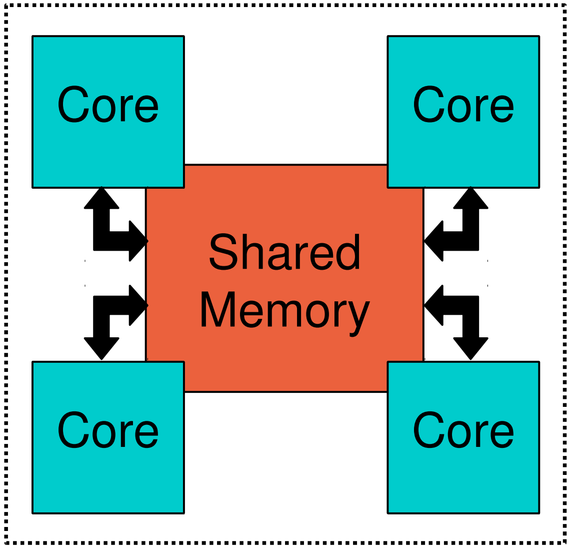


Drawing: Herb Sutter

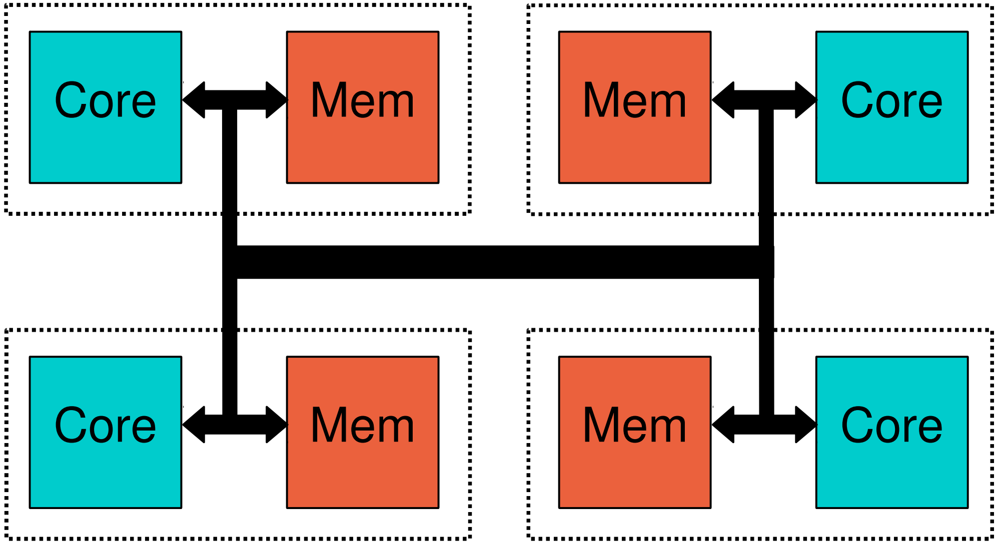
Multicore Architectures

- A processing system with 2 or more independent cores integrated on the same chip
- Number and type of cores:
  - multicore or manycore
  - heterogeneous or homogeneous
- Memory architecture:
  - shared
  - distributed
  - mixture
- Interconnection network

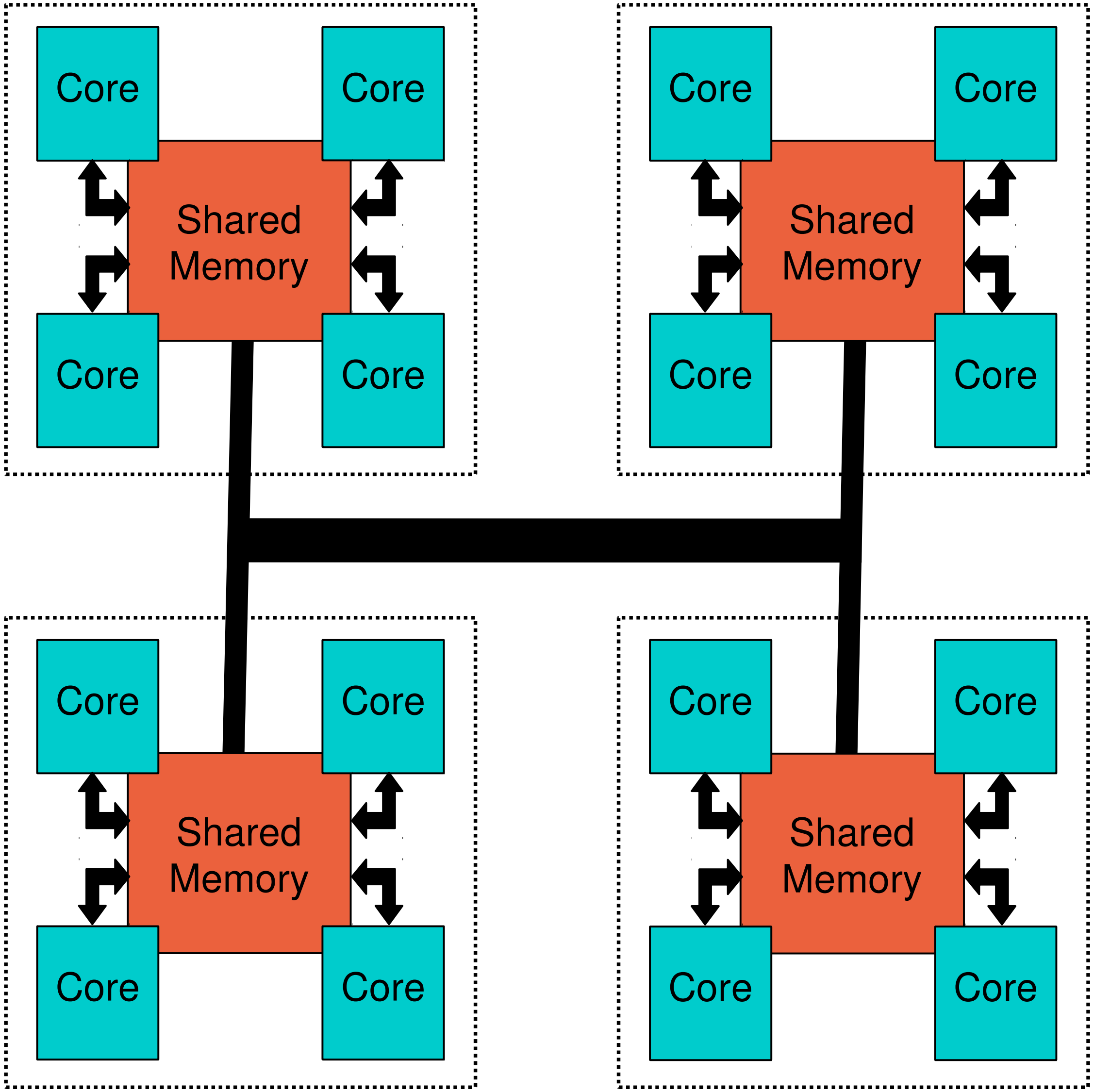
Shared Memory Multicore Architectures



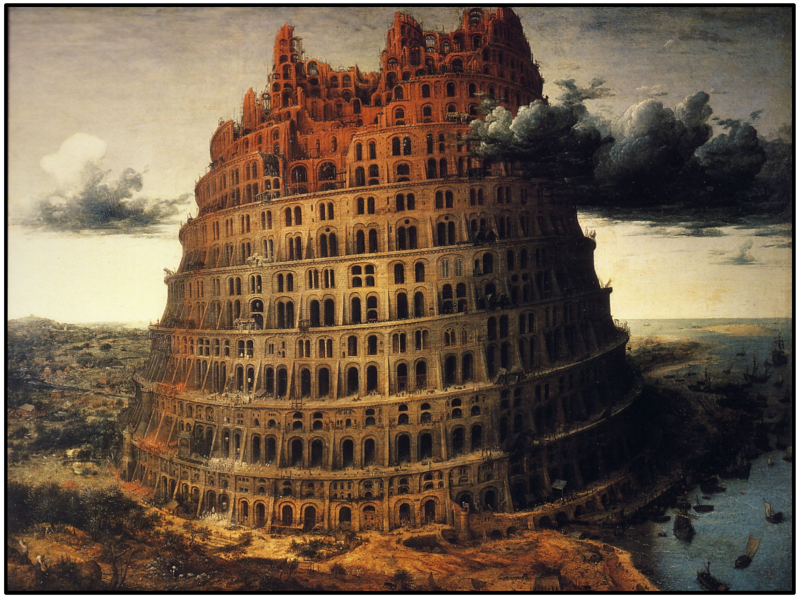
Distributed Memory Multicore Architectures



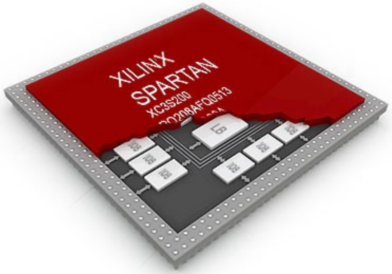
(Future) Manycore Architectures

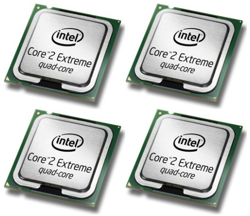


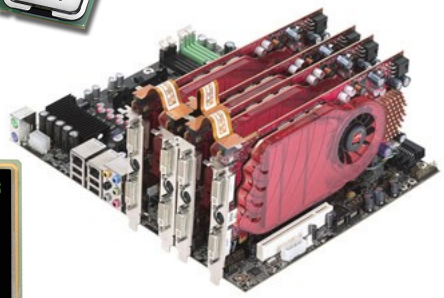
The “Babel” of Multicores

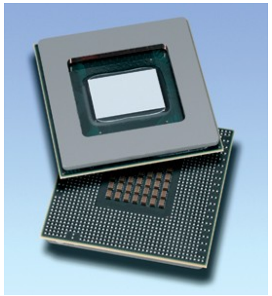


Overwhelming Multicores

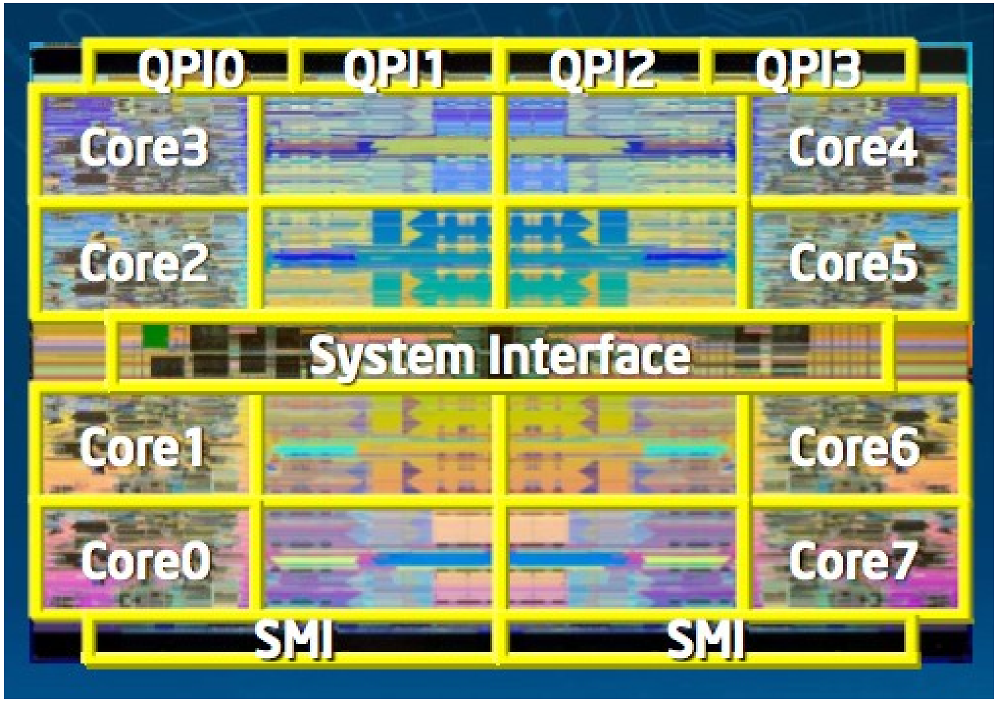






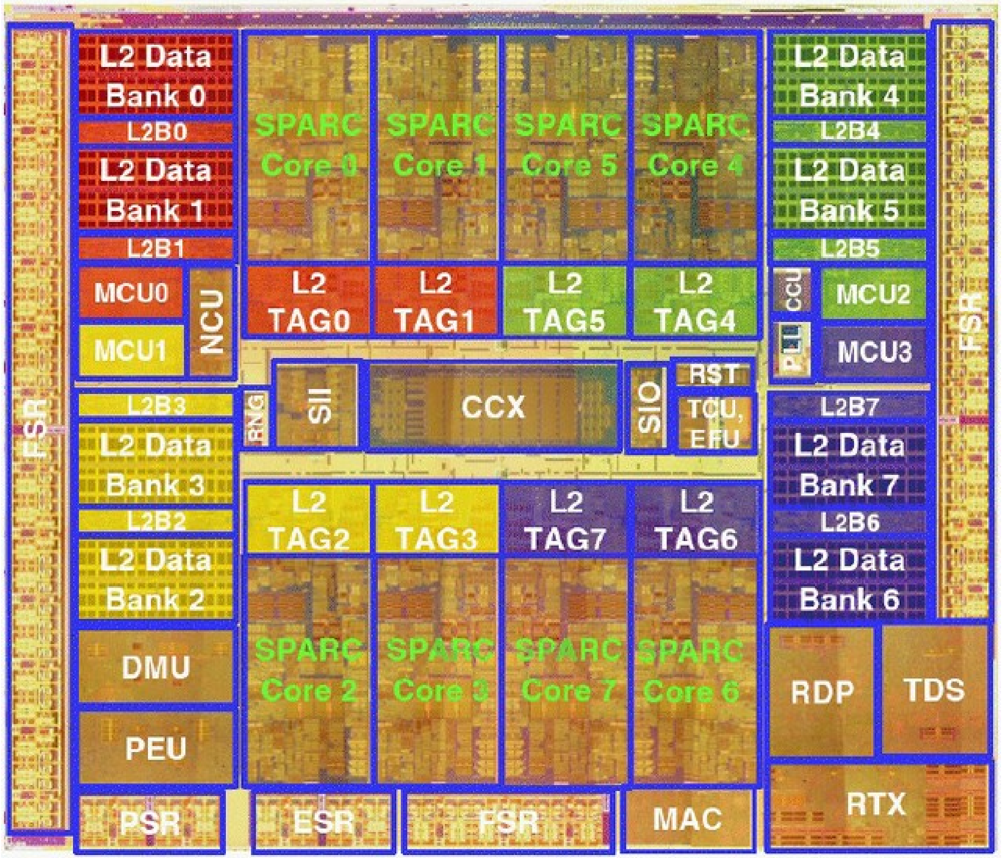


Intel's Nehalem Architecture



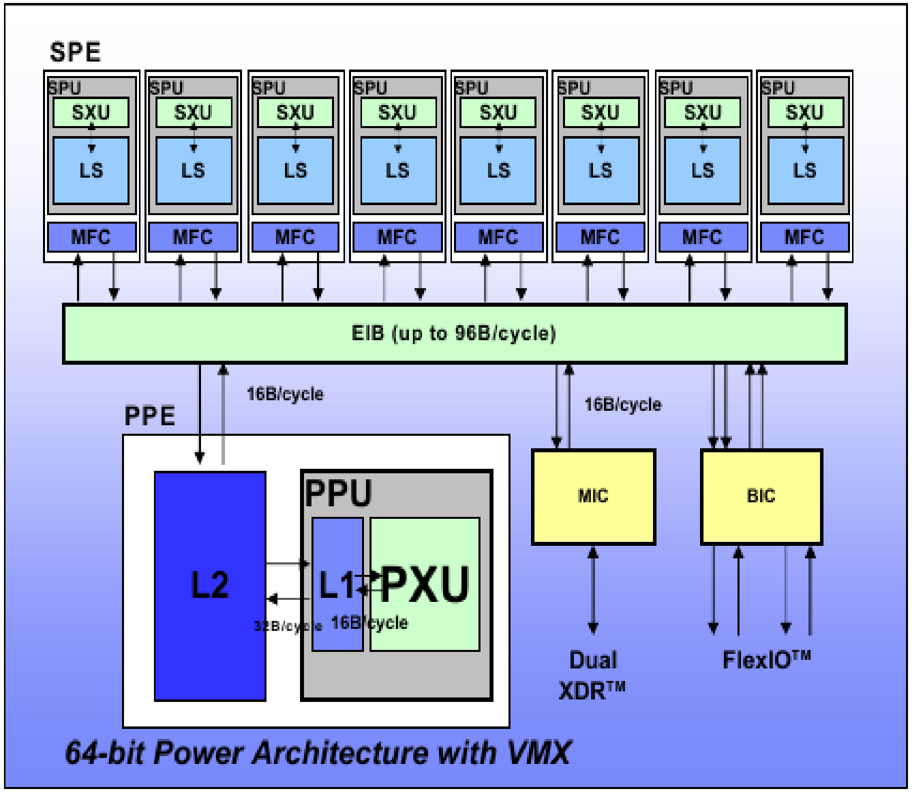
- shared memory
- multi-threading
- up to 8 cores
- 2 threads/core
- private L1 and L2 cache
- 24MB shared L3 cache

Sun's Niagara 2 Architecture



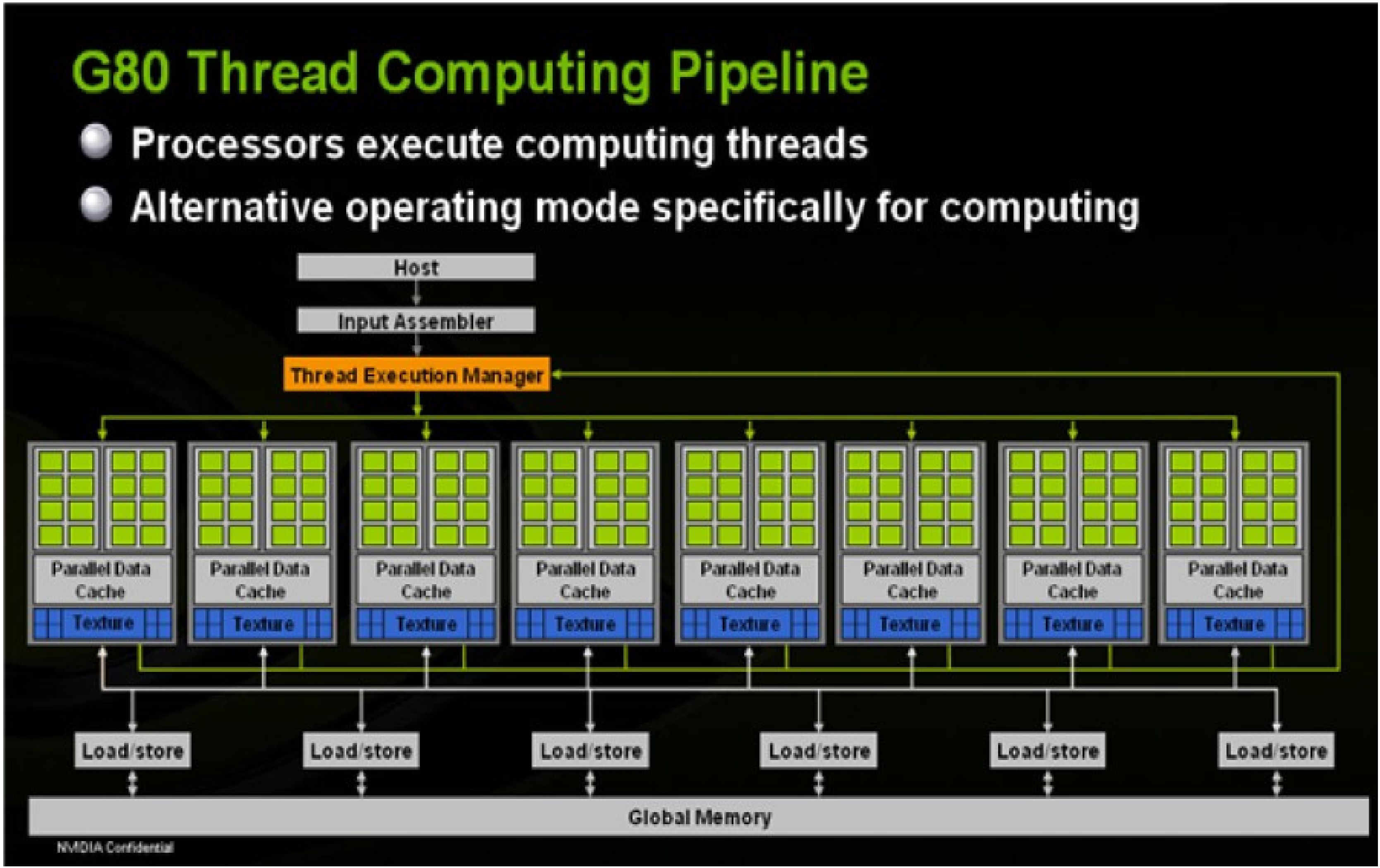
- shared memory
- multi-threading
- up to 8 cores
- 4-8 threads/core
- 4MB shared L2 cache

Cell BE – Heterogeneous Architecture



- 1 PPU for OS and PC
- 8 SPE capable of vector processing with local store memory (4GB)
- high-performance interconnection bus

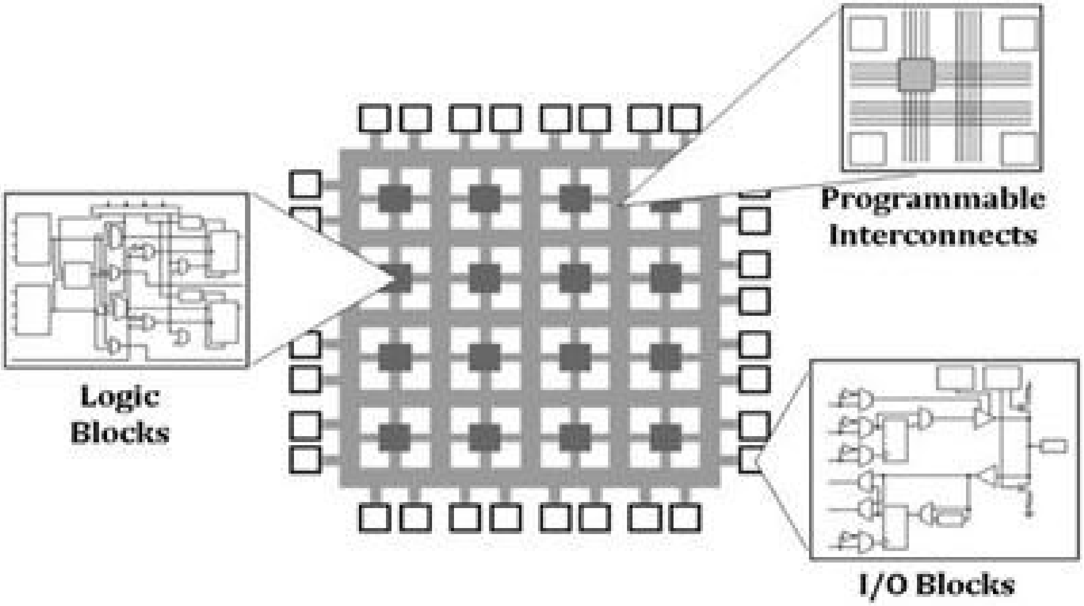
NVIDIA GPUs with CUDA



G80 Thread Computing Pipeline

- Processors execute computing threads
- Alternative operating mode specifically for computing

Field Programmable Gate Arrays (FPGAs)



- device with a matrix of reconfigurable gate array logic circuitry
- when configured works as a hardware implementation of a software application
- user can create task-specific cores that all run like parallel circuits inside one FPGA chip

The Challenge of Parallel Thinking

“Think Parallel or Perish”

James Reinders
Chief Evangelist –
Intel’s Software Products Division

2009:

“... the 'not parallel' era will appear to be a very primitive time in the history of computers when people look back in a hundred years...”

“... in less than a decade, a programmer who does not 'Think parallel' first will not be a programmer”

Multicore is a Challenge

- Primarily in software development
- Performance speed up depends on how good is the multi-threading of the parallel source code
- Parallel code ought to be:
  - correct
  - efficient
  - scalable
  - future-proof
- Portable code across platforms – major issue

Start to Think Parallel

- what hardware do we have?
  - multithreaded system architecture
  - GPU
  - heterogeneous multicore (ex. Cell BE)
  - FPGA
  - etc...
- language: what data structures and operations are supported?
- identify parallelism
  - embarrassingly parallel?
  - functional decomposition: task parallelism
  - data decomposition: data parallelism

Start to Think Parallel

- what hardware do we have?
  - *multithreaded system architecture*
  - GPU
  - heterogeneous multicore (ex. Cell BE)
  - FPGA
  - etc...
- language: what data structures and operations are supported?
- identify parallelism
  - embarrassingly parallel?
  - functional decomposition: task parallelism
  - data decomposition: data parallelism

Threading Methods

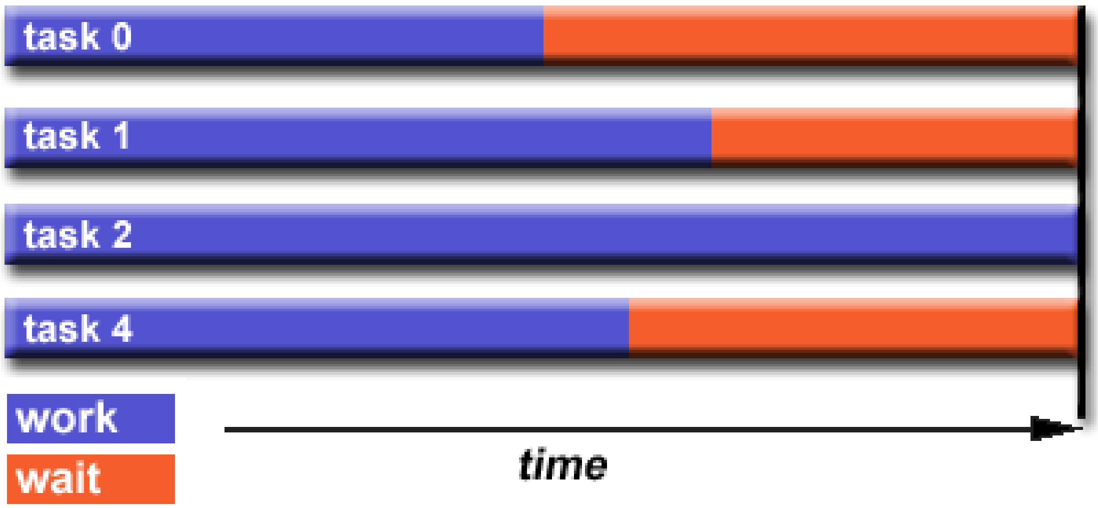
- Explicit threading (rather low level)
  - manually write all code responsible for managing threads that interface to a specific library
- Library-based – best for functional decomposition
  - user creates and synchronize threads explicitly
  - Ex. Pthreads
- Compiler-based – best for data parallelization
  - user annotates code with pragmas
  - Ex. OpenMP, TBB

Golden True: Use Abstraction Where Possible

- Future-proof applications
- Express parallelism, without thinking much about threads/core management
  - Libraries, OpenMP, Intel TBB – good examples for this
- Best to avoid raw native threads, like Pthreads
  - Native threads and MPI are like the assembly language of parallelism
- Think in *tasks*, not threads

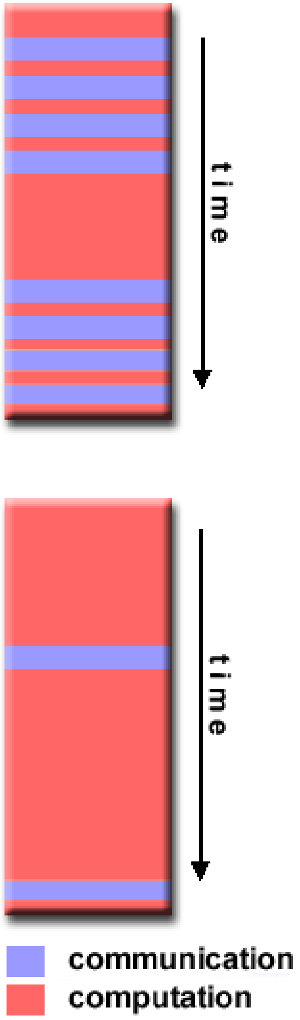
Load balancing

- keep all threads busy all the time



Fine- or Coarse-grain – Which is Best?

- Depends on algorithm and hardware
- Fine-grain: good for load-balancing, but too much communication overhead
- Coarse-grain: more opportunity to increase performance, but not so good for load-balancing



Lock-based Synchronization

- is error-prone
- they may cause blocking:
  - deadlocks: threads are waiting for each other to release a resource
  - livelocks: threads continuously change their state but not doing any useful work.
- Lock-free programming: e.g. transactional memory

Exercise: Parallelizing A Baking Process

- We are making a birthday cake:
  - Mix ingredients: 20 minutes
  - Bake: 30 minutes

- Can we parallelize it?
  - How many cooks?
  - Each cook has his own spoon?
  - How if I make cup-cakes?

A Note on Performance Gain

- Amdahl's Law: *the pessimistic*
  - A program's serial portion is a practical upper bound on the performance of its parallel portion
  - Baking a cake:

Number of cooks	Time	Speedup
1	30 + 20 = 50	1.0x
2	30 + 10 = 40	1.2x
4	30 + 5 = 35	1.4x
infinite	30 + 0 = 30	1.6x

  - Overall parallel performance is still limited by the baking time
- So are massively parallel systems hopeless... ?

A Second Note on Performance Gain

- Gustafson's Law: scaled speed-up measurement – *the optimistic*
  - What if we want to bake *100* cakes?

Number of cooks	Time	Speedup
1	30 + 20*100 = 2030	1.0x
2	30 + 20*50 = 1030	1.9x
4	30 + 20*25 = 530	3.8x
infinite	30 + 20*0 = 30	67x

  - certain problems have increased performance by increasing the problem size
  - the problem size scales with the number of processors
  - speed-up should be measured by scaling the problem size to the number of processors, not fixing the problem size

Parallelizing is Difficult

- Writing correct, efficient parallel programs has always been challenging (e.g. HPC)
- This applies to multicore programming too
- Higher abstraction levels help
- *“We cannot start from scratch whenever a new multicore hardware turns up”*

Ongoing Research

- Goal: to lift the abstraction level even higher
  - To free the user from low level hardware details
  - To bridge the gap between programming different types of multicores and/or HPC facilities
- Many high-level programming models have been proposed:
  - Functional approaches: Haskell, SAC, Crystal, etc
  - Data parallel languages: NESL, DPH, SAC, Fortran95, Sisal etc.
  - Implicit parallelism: HPF, Id, NESL, Sisal, ZPL
  - PGAS model: UPC, Co-Array Fortran, Fortress, Chapel, X10
  - etc.

Languages and Compilers for Multicores

Intel's for Multicore CPUs

- Intel Compilers support OpenMP
- Intel launched its own MPI library
- Performance analysis tools, debuggers
- Intel TBB – adding parallelism to C++
- Intel's Ct technology – nested data parallelism for C++
- Intel Parallel Studio – an all in all support toolbox
- Higher-level models:
  - Intel Concurrent Collections for C++
  - Intel Cilk++ Software Development Kit

Intel's Parallel Studio

- Microsoft Visual Studio C/C++ developers toolbox
- interoperable with OpenMP and Intel's TBB libraries
- helps the programmer throughout the parallelization process (to identify, create, debug and tune)

Others

- Java: Java threads, java.util.concurrent package
- Microsoft .NET: Task Parallel Library (TPL)
- Haskell: thread programming and data parallelism
- etc...

Programming GPGPU

- NVIDIA's CUDA model:
  - Gives access to the enormous computing power of NVIDIA GPUs via standards like OpenCL, C/C++, Fortran, Python, .NET
- OpenCL – generally adopted by other GPU vendors (e.g. AMD)

OpenCL (Open Computing Language)

- a new open standard for programming heterogenous systems suported by most hardware vendors
- uniform programming environment to write efficient, portable code for both multicore CPUs and GPUs
- http://www.khronos.org/opencl/

Summary

- Multicores (hardware)
  - are reality
  - are overwhelming
  - many, more complex, more heterogeneous to appear

- Multicores (software)
  - writing parallel code is challenging (always has been)
  - programming models are versatile and confusing
  - portability across various platforms major issue

- Unified high-level parallel programming model is still open research

Staying Tuned?

- http://software.intel.com/en-us/parallel/
- http://www.upcrc.illinois.edu/
- http://www.multicoreinfo.com/
- http://www.drdobbs.com/go-parallel