Johan Seland

Multi-Core Programming

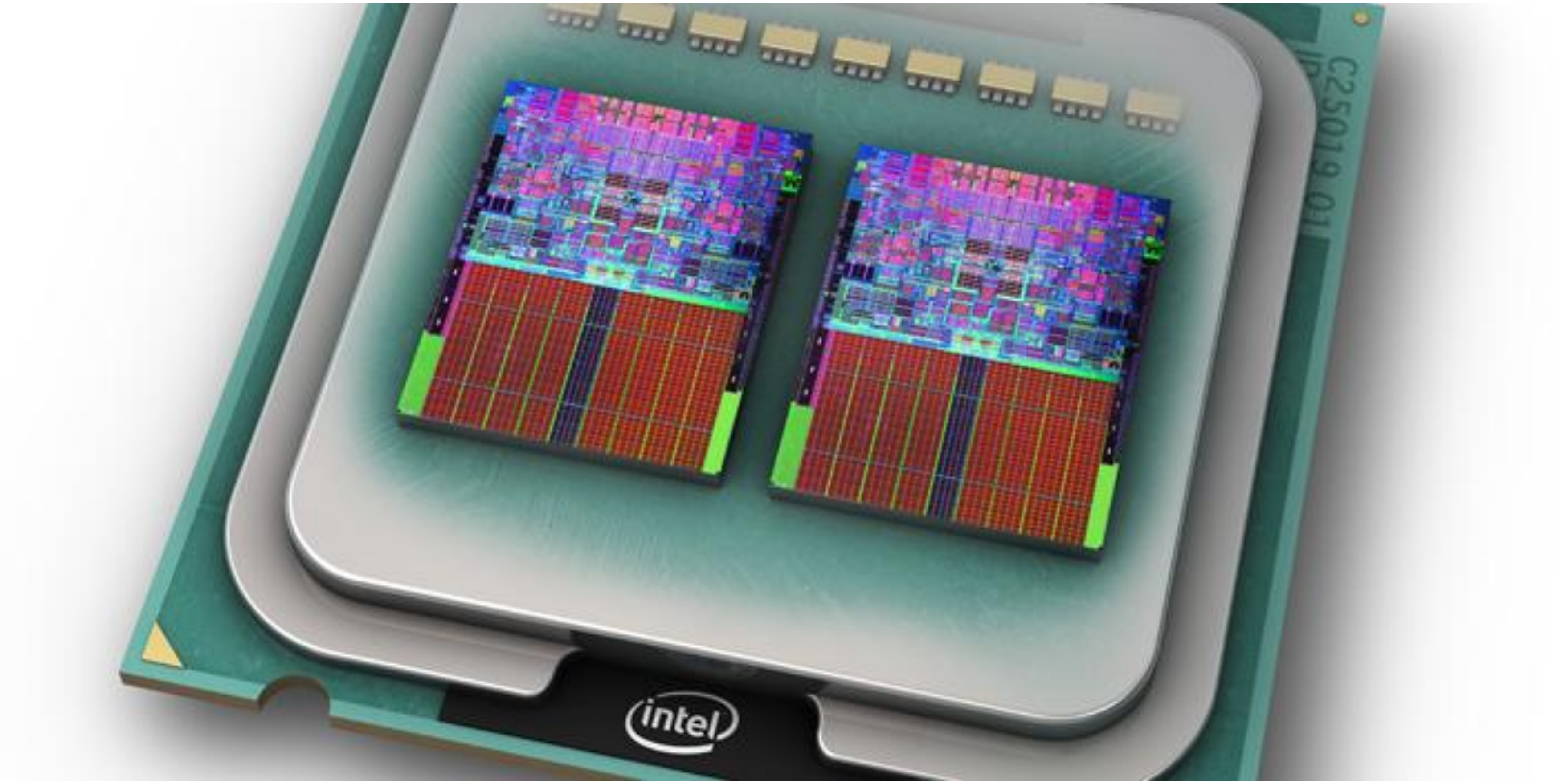
SINTEF Petroleum Development Workshop – Session 3

Trondheim - 9. December 2010

Overview

- Multi-Core Architectures
- Memory Hierarchies
- OpenMP
- Alternatives:
  - C++0X, TBB and Java
- Intel Parallel Studio
- Discussion

Multi-Core Architectures



Multi-Cores

- All desktops, most laptops and some mobile phones are now multi-core
  - Desktops can also-be multi-socket
- Follows Moores law by duplicating functionality on die
- Memory (RAM) is addressable by all cores
  - **Shared memory**
  - NUMA on the horizon
- All cores shares disks, network, GPUs etc.

- Programs must be explicitly written to use more than one core
  - Equally suited for data or task parallelism

Example: Intel Sandy Bridge

- To be released January 2011
- 32 nm process
- 2-4 Cores at launch. (6 and 8 later)
- 2.2 GHz – 3.4 GHz
- Theoretical performance :128 GFlops (double precision)
- Hyper-threading

- Each core has:
  - 64KiB L1 cache (3 clocks)
  - 256KiB l2 cache

- 8 MiB shared L3 cache

Instruction Level Parallelism (ILP)

- Optimizing compilers reorder instructions to hide latency
  - Compilers can make no assumptions on data values
- Processors know data values and can dynamically reorder instructions at runtime
  - Out of order execution
  - Branch prediction
- Greatly increases the complexity of chip designs

- GPUs and simple mobile CPUs execute their instructions in-order
- Happens automatically

Vector Units

- 3DNow, MMX, SSE{1,2,3,4}, AVX, AltiVec (PowerPC)
- Single Instruction, Multiple Data (SIMD)
- Sandy Bridge: 32 GFlops x87, 128 GFlops AVX

- Compiler tries to detect possibilities
  - Remember to activate this!
  - Breaks backward compatability
- Can also be activated directly (compiler intrinsics)

- "Tiny vector" libraries for small vectors and matrices (up to 4x4)
- Fast Fourier transform

Hyper-Threading

- Registers and Instruction Pointer (IP) require few transistors compared to ALUs
  - Duplicate these
  - Up to 30% performance increase for 5% increase in die area
- One core appears as two "logical cores"
- Only on high-performance CPUs

- Morale: Expose as many threads as possible

Memory Hierarchies

Memory Hierarchies

Definition of CACHE

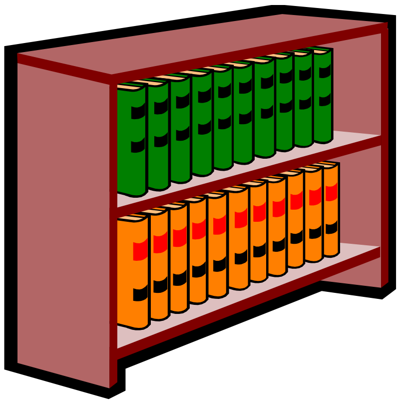
- **1**
  - **a :** a hiding place especially for concealing and preserving provisions or implements
  - **b :** a secure place of storage

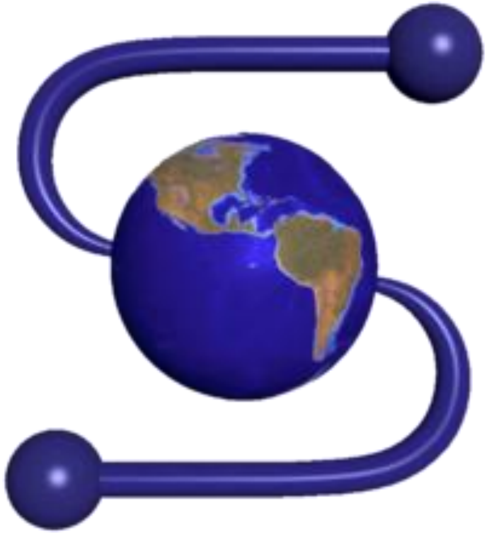
- **2 :** something hidden or stored in a cache

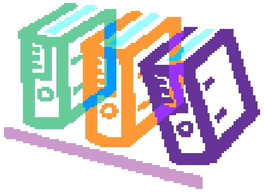
- **3 :** a computer memory with very short access time used for storage of frequently or recently used instructions or data —called also *cache memory*

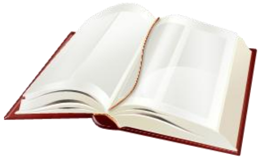
Merriam-Webster

Why caches









Memory caches

- Layers of faster (and more expensive) memory that hides latency
- Highly effective
- Often implements read-ahead policy
- When to write back?
  - Snooping bus

- Works transparently
  - But you should ensure that traversals follow the read-ahead policy (example)
  - Blocking/chunking should fit in cache
  - Autotuning

- Web and disk caches operate similarly

Memory access times and sizes

Cache Level	Access Time	Size
Register	0	16++
L1	3 cycles	32KiB + 32KiB
L2	8 cycles	256KiB
L3	25 cycles	8 MiB
RAM	0.1 ns – 5 ns	2GiB – 1 TiB
Disk	20 ns – 70 ns	128GiB – 10 PiB
Network	10ms - ∞	∞

Demo

- Parallel Studio memory example

OpenMP™

OpenMP

- An implementation of **shared-memory** multithreading

- Version 1.0 released in 1997, version 3.0 in 2008
- Fortran and C/C++
- Open standard (backed by Intel, AMD, Microsoft, Oracle++)
- Broad compiler support

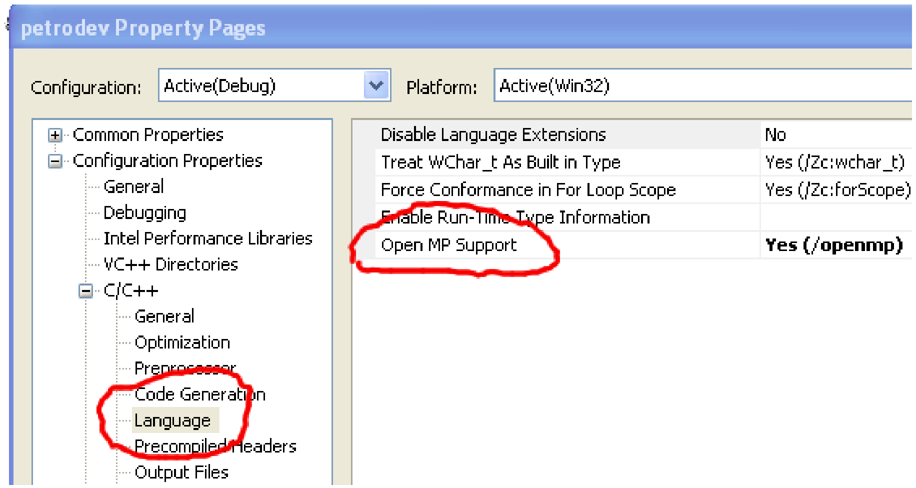
- Implemented as compiler pragmas + small library
  - Can preserve serial program!
- Procedural
- Can gradually be bolted on existing code
- Specification is surprisingly readable (openmp.org)

OpenMP – Whats it not to like?

- Mostly for Data Parallelism
  - Only for regular structures
  - Only for built in datatypes
  - OpenMP 3.0 remedies this a bit
- Extends the language
- Only subset of language allowed in constructs
- Mix algorithm code and threading code
- No datastructures
- No support for objects

Enabling compiler support

- Visual Studio



- GCC (including gfortran)
  - -fopenmp on command line
- Intel Fortran
  - -openmp (Linux)
  - -Qopenmp (Windows)

Example: OpenMP Dot Product

```
double dot_product( vector<double>& a, vector<double>& b ) {
        double sum = 0;
        const int n = a.size();

#pragma omp parallel for reduction(+:sum)
        for ( int i = 0; i < n; ++i ) {
                sum += a[i]*b[i];
        }

        return sum;
}
```

Add compiler pragma

OpenMP Directive Clause

- C/C++
  - **#pragma omp *directive-name [clause,...]***
  - Case-sensitive

- Fortran (free-form)
  - **!$omp *directive-name [clause,...]***
  - Case-insensitive

The parallel construct

- Specify computations that should be executed in parallel"foo"

```
#pragma omp parallel
{
  const int threadId = omp_get_thread_num();
  cout << "I am thread: " << threadId << "\n";

  if ( threadId == 2 ) {
    cout << "Only thread 2 goes here\n";
  }
}
```

Restrictions

- Can not branch into or out of parallel region
- Do not depend on ordering
- Can not throw exceptions out of parallel region
  - Exceptions must be thrown and caught in the same region

Work-sharing constructs

Functionality	C/C Syntax	Fortran Syntax
Distribute iteration over threads	#pragma omp for	!$omp do
Distribute independent work units	#pragma omp sections	!$omp sections
Only one thread executes code block (critical section)	#pragma omp single	!$omp single
Parallelize array-syntex	NA	!$omp workshare

Data-sharing clauses

Clause	Effect	Notes
shared(…)	Variables that will be shared by threads (implies memory fence)	
private(…)	Variables which are replicated to every thread	Loop counter is implied
firstprivate(…)	Variables are pre-initialized by the value before the construct	
lastprivate(…)	Varibable after the construct has the value of the "last" thread.	
schedule(…)	Control how loop iterations are distributed over threads	
Reduction(op:variable)	Identify variable that will hold result of reduction	Order of operations is not guaranteed

Example – data sharing clauses

```
!$OMP PARALLEL SHARED(a,b) PRIVATE(i)
!$OMP DO
do i=1, 1000
  a(i,j) = 3.14*b(j,i)
Enddo
!$OMP END DO
!$OMP END PARALLEL
```

Synchronization Constructs

Clause	**Syntax**	**Notes**
Barrier	#pragma omp barrier	Threads may no proceed barrier until all threads are at this point
Ordered	#pragma omp ordered	Enforce order within parallel construct
Critical	#pragma omp critical	Only one thread may enter critical region at a time
Atomic	#prgama omp atomic ***statement***	One-line critical section. Useful for assignments. Effective on some HW.
Locks	omp_*func*_lock(*lck)	Get, free, test lock.
Master	#pragma omp master	Executed by the master thread only

OpenMP discussion

- Easy to get started on existing codes
- Small syntax – easy to learn
- Don't try to be too fancy

C++0X

- Next version of C++ standard
- Draft expected to be completed in March 2011
- Visual Studio 2010 and GCC 4.x has support for much of it

- Lots of nice stuff:
  - Auto variables
  - Lambda functions
  - Initializer lists
  - Smart pointers
  - Hash tables
  - Tuples
  - Regular Expressions
  - **THREADS**
  - ++

Brief overview of C++0X threads

- New std::thread class

```
void do_work() {...};
std::thread t(do_work);
// do other stuff
t.join() // wait for t to finish
```

- std::mutex and friends

```
std::mutex m;
void foo() {
  std::lock_guard<std::mutex> lock( m );
  process( data );
} // mutex unlocked in d'tor (RAII)
```

- Condition variables
- thread_local keyword

Threading Building Blocks

- STL inspired thread library
- Originally developed by Intel
  - Commercial and Open Source (GPL)
  - http://www.threadingbuildingblocks.org/
  - At version 3.0

- Parallel Algorithhms
  - for, while, reduce, scan, pipeline
- Concurrent containers
  - queue, vector, hashmap
- Mutexes and atomic operations
- Advanced task scheduling

Java

- Java classes can implement the "Runnable" interface
  - One method: run()

```
class PrimeRun implements Runnable {
        void run() {…}
}

PrimeRun p = new PrimeRun();
new Thread(p).start();
```

- Concurrent containers
- Class monitors: `synchronized` keyword
- Java 7 (mid 2011): Fork-join
- Java 8 (late 2012): Lambdas and closures

Other languages/libraries

- MPI
- Posix Threads (Pthreads)
  - C-style API, verbose
- QThread - thread abstraction in Qt
- Cilk
  - Intel owned, C extension (spawn, sync, inlet, abort)
- OpenCL
- CUDA
- Haskell

Conclusion

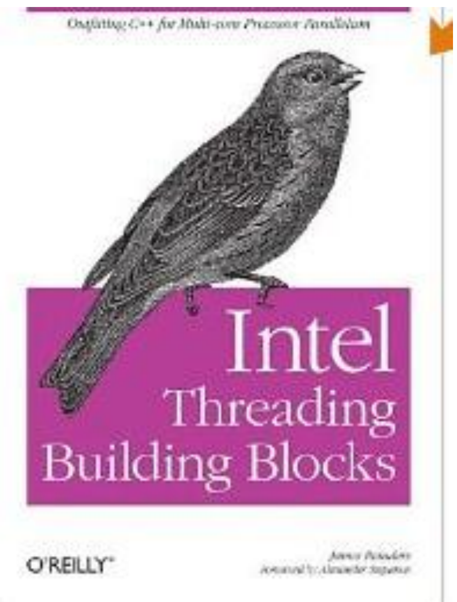
The age of multi-core is NOW!

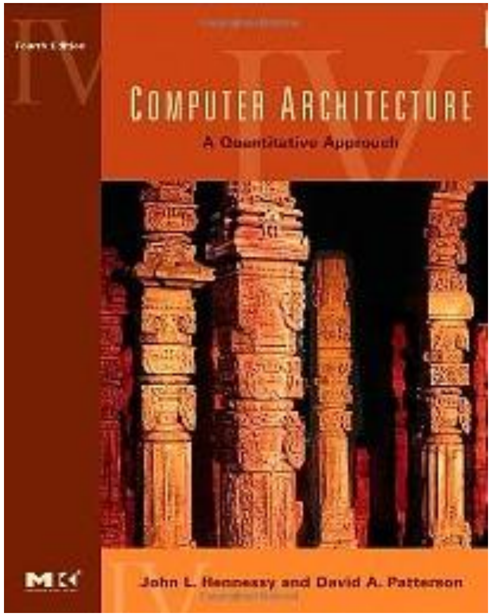
(It has been here for 5+ years)

- For HPC you can not afford to ignore this
  - Much easier for business/web developers
- Pick your abstraction level
- Existing codes can "easily" be extended with some OpenMP

- For new projects:
  - Thoroughly evaluate performance, people and business value before choosing tool

Reading list





What Every Programmer Should Know About Memory

Ulrich Drepper
Red Hat, Inc.
drepper@redhat.com

November 21, 2007