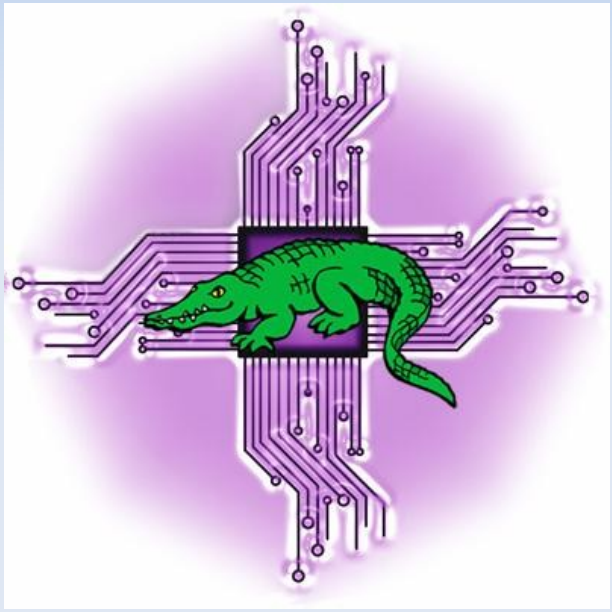
Multicore and Linux

Presentation to Linux Users of Victoria on Multicore and Linux



May 1st, 2012

http://levlafayette.com

The Multicore Revolution

Appropriately we're discussing this on May Day – which started at Melbourne University in 1856.
(cf., http://www.marxists.org/archive/luxemburg/1894/02/may-day.htm)

Despite advances in manufacturing, limits in semiconductor processors lead to a wall where heat and data synchronisation is a problem.

These limits, and the solutions, are relatively new in terms of commercial implementation.

What is a Multicore System?

Flynn's Taxonomy of Computer Systems is a complex between single and multiple data streams and single and multiple instruction streams. Four alternatives; (i) Single Instruction – Single Data (ii) Single Instruction – Multiple Data (iii) Multiple Instruction – Single Data (iv) Multiple Data – Multiple Instruction. All but SISD are multicore.

A multiprocessor has two or more processing units each sharing main memory, system I/O and peripherals. A multi-core processor is a single computing component with two or more independent actual processing cores. A system can be multiprocessor *and* multicore (e.g., dual processor, quad core = eight processing cores per system unit).

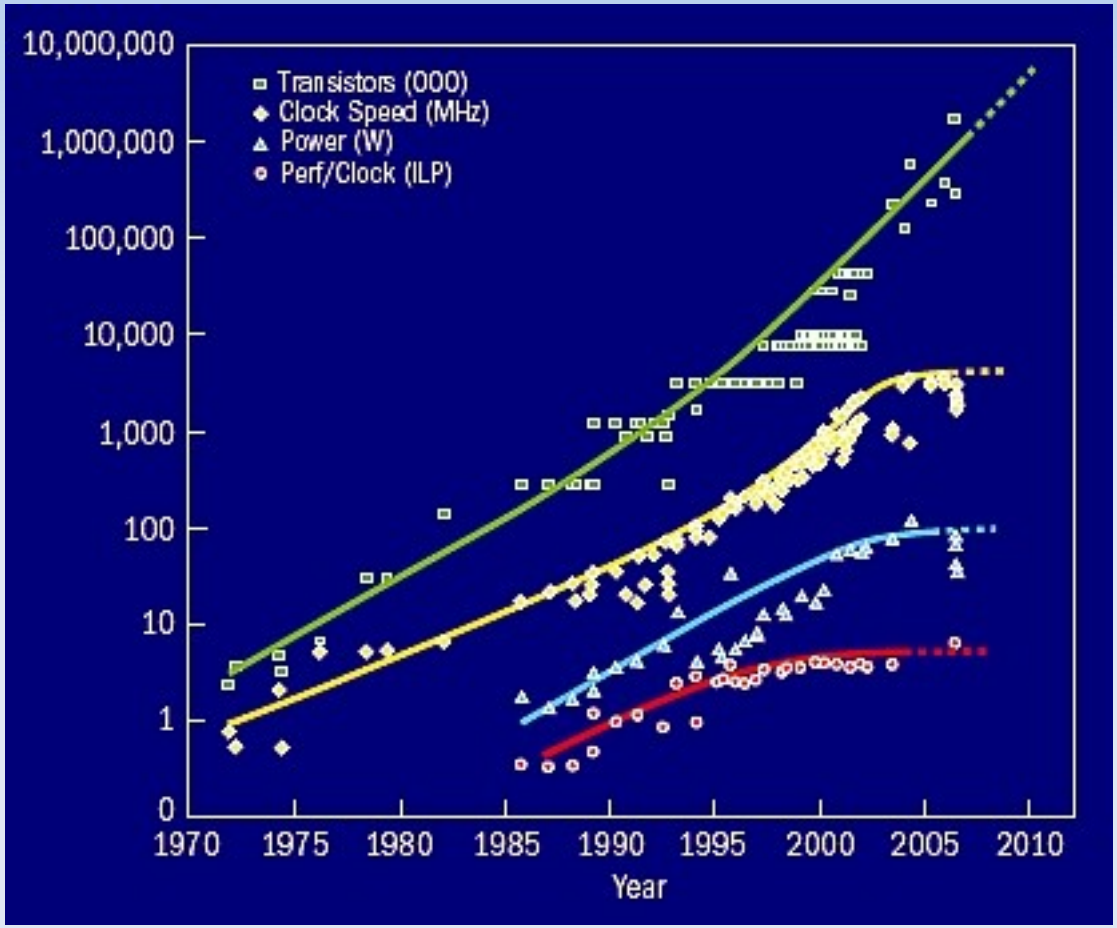
An execution thread is the smallest processing unit in an operating system. A thread is typically contained inside a process. Multiple threads can exist within the same process and share resources. On a uniprocessor, multithreading generally occurs by switching between different threads. On a multi-core system, the threads or tasks will actually run at the same time, with each processor or core running a particular thread or task.

Necessity of Multicore Systems

Clock rate has (mostly) stalled over the past decade. Apart from the physical reasons, it is uneconomical.

It's simply not worth the cost increasing the frequency of clock rate in terms of the power consumed and the heat dissipitated.

Intel calls the rate/heat trade-off a “fundamental theorem of multicore processors”.



It Is Already A Multicore World

IBM dual-core Power 4 chip in 2001, for use in IBM's RISC servers, in 2001. AMD and Intel announced their respective dual core plans in 2004 and began their first dual-core shipments in 2005 for desktop systems.

In 2006 Intel released quad core processors, with AMD following in 2007. In 2009 Intel released an octa-core processor (Xeon) and in 2011 AMD (Operton Interlagos) hexadeca-core (16) processors

And so on....Is there an end to this?

Maybe not! IBM Power7 processors are available in sixteen-core or thirty two-core (triacontaduo-core) configuration for servers.

As far back as 2007 Tilera (using RISC CPUs) released sixty-four core processors and in 2009, one-hundred core processors.

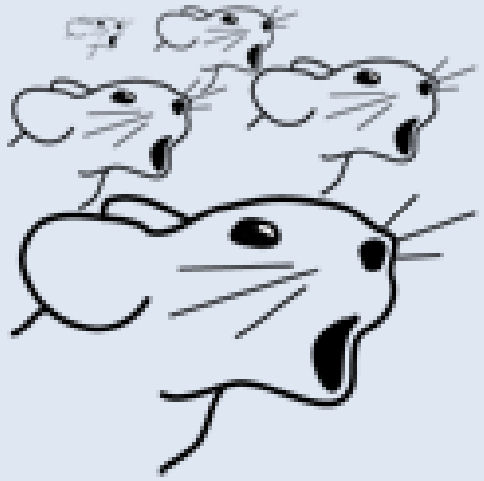
In 2012 Tilera founder, Dr. Agarwal, is leading a new MIT effort dubbed The Angstrom Project. It is one of four DARPA-funded efforts aimed at building exascale supercomputers. The goal is to design a chip with 1,000 cores.

Programming Challenges (I)

Multicore performance depends on the software algorithms. The improvement of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program (Amdahl's Law). Solution? Make the problem bigger! (Gustafson's Law).

Parallel programming is *hard*. More complexity = more bugs. Correctness in parallelisation requires synchronisation (locking). Synchronisation and atomic operations causes loss of performance, communication latency.

At the Renaissance project at IBM, Brussels, and Portland State, there are investigations in "anti-lock", "race-and-repair", or "end-to-end nondeterministic" computing. (http://soft.vub.ac.be/~smarr/renaissance/)



Programming Challenges (II)

Can parallel programming for multicore systems become automated?

Locks are manually inserted; without locks programs can be put in an inconsistent state. Multiple locks in different places and orders can lead to deadlocks. Manual lock inserts is error-prone, tedious and difficult to maintain. Does the programmer know what parts of a program will benefit from parallelisation?

To ensure that parallel execution is safe, a task's effects must not interfere with the execution of another task. An automated system should be able to determine whether parallelisation is safe and efficient and translate sequential to parallel code.

These checks are very simple in pure declarative languages; in imperative languages, these checks are much more complex (require aliasing analysis).

A paralleliser would profile a program, determine inefficient parts that can be run in parallel.

Multicore World New Zealand (Part I)

Held at the Wellington Town Hall, 27 - 28th March. Opened by the Mayor, Celia Wade-Brown - a former a former APL programmer.

First keynote by James Reinders on parallel C/C++ programming, parallel race conditions, flat clock rates since 2004 vs multicore. Followed by Dr. Tim Cornwell on the challenge of data processing algorithms as astronomical data requirements increase; data in the range of 10 petabytes/s and compute power in the petaflop range, a topic that was also taken up by Mahmoud S. Mahmoud. Dr. Mark Moir of Oracle gave an *excellent* introduction to concurrency and synchronisation issues.

Dr. Martin McKendry and Stephan Friedl spoke on the contribution of multicore to switching and in particular the need to move to NoSQL databases. Dr. Mark Moir, argued that concurrent programming is essential and forced by multicore necessity. Paul Bone noted argued that automatic parallelisation is possible with declarative languages (e.g., Mercury).

Dr. Zhiyi Huang spoke on task scheduling in asymetric multicore. John Goodacre elaborated on the role of of ARM architecture in heterogeneous computing. Artur Laksberg spoke on Asynchronous Programming in C++.

Multicore World New Zealand (Part II)

Dr. David Eyers, gave a technical talk but with serious business implications, with studies in multicore power versus energy metrics. Jim Peek spoke on data centre innovation.

Claire Curran, Opposition spokesperson for Communications and Information Technology attended the second day of the conference.

Sebastian Sylwan of Weta presented on art, technology, an amazing render wall and incredible detail. New Zealand has a very significant proportion of the world's Top500 computers, which are nearly all owned by Weta. Dr. Tim Mattson (Intel/Khronos) gave an entertaining presentation on future of multicore, but also on funding. "We have a truckload of money" (Phil McCaw, "We have a mini"). Very keen on public research centres in universities. Scott Houston and Dave Fellows (GreenButton) discuss how software vendors can survive and benefit with cloud technologies. John Houlker gave a sober presentation from the perspective of NZ Trade and Enterprise.

Lenz Gschwendtner, spoke on the enthusiasm of technology of parallel programming (compared to steam engines).

Two major panel discussions: Simple Evolution or Cambrian Explosion?, The Road Ahead from Today

Developments in the Linux Kernel For Multicore Systems

GNU Linux is a symmetric multiprocessing (SMP) operating system - it know how to manage a multiple processor with a single shared main memory.

Linux used to use a "big kernel lock" to provide the concurrency control required by symmetric multiprocessing (SMP) systems, used whenever a thread entered kernel space, and is released when the thread returns to user space (a system call is the archetypal example). BKL threads in user space can run concurrently on any available processors or processor cores, but no more than one thread can run in kernel space. The Linux kernel had a big kernel lock until it was finally replaced by more fine-grained locking mechanisms by Arnd Bergmann in 2011.

By 2.2 big kernel lock for most of kernel, interrupts own lock. 2.4 more fine grained locking, still several common global locks. 2.6 further BLK reductions, new subsystems (multi queue scheduler, multi flow networking) 2.6.35 included (thank you Google) Receive Packet Steering (RPS) and Receive Flow Steering (RFS) to spread the load of network handling across the CPUs available in the system. Network cards have improved the bandwidth to the point where it's hard for a single modern CPU to keep up. In 2.6.36 Tilera architecture support (Tile CPUs) was added. In 2.6.38 automatic process grouping was added, allowing processes with the same sessionID to be grouped on the same processor as a single scheduling entity.

Questions and Thanks

Questions?

Thanks to Nicolás Erdödy for organising Multicore World 2012 and Paul Bone for many discussions during the conference itself which contributed to this talk (*especially* on automatic parallelisation).