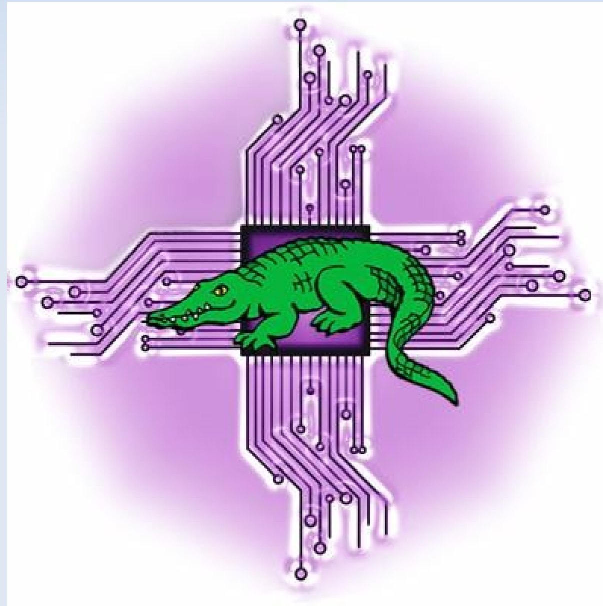
Parallel Programming

Presentation to Linux Users of Victoria, Inc.



November 4th, 2015

http://levlafayette.com

1.0 What Is Parallel Programming?

1.1 Historically, software has been written for serial computation (discrete instructions, sequential execution, single processor)

1.2 Parallel programming is the procedures used for simultaneous computation (discrete parts, concurrent executions, multiple processing).

1.3 Flynn's Taxonomy of Computer Architecture into streams of data and instructions (SISD, SIMD, MISD, MIMD). Later three architectures allow for parallel programming.

Instruction Streams

Data Streams	one	many
one	SISD traditional von Neumann single CPU computer	MISD maybe pipelined computers
many	SIMD vector processors fine-grained data parallel computers	MIMD multicomputers multiprocessors

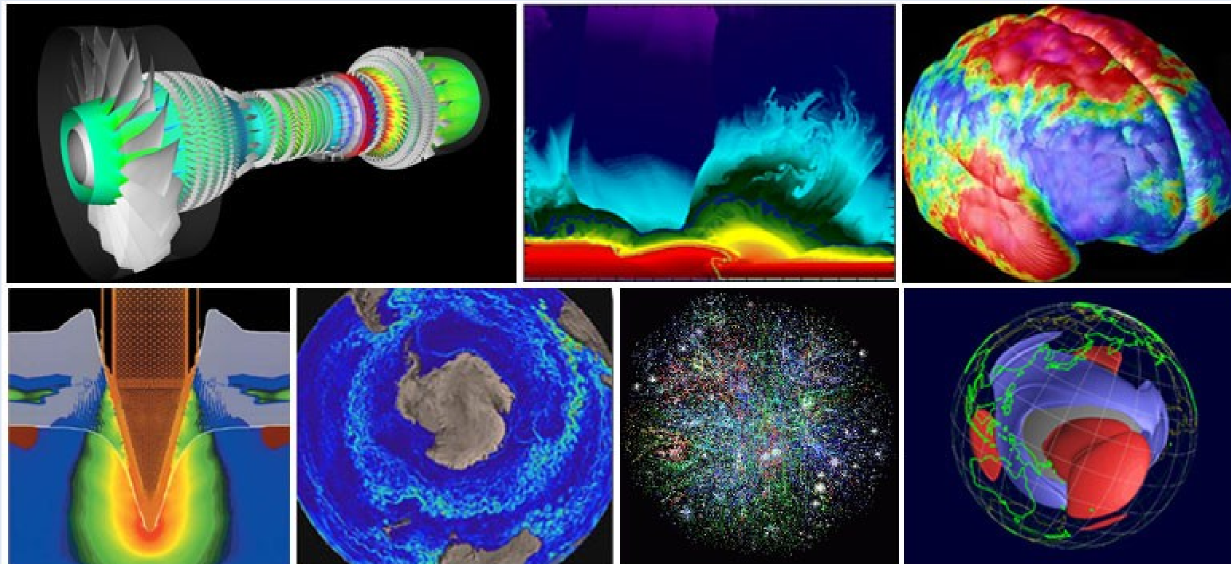
2.0 Why Do We Need Parallel Programming?

2.1 The size of dataset are growing faster than the processing capability of unicore systems.

2.2 Modelling most real-world phenomena is a parallel task (weather and climate, astronomy, medicine, geological change, economics, robotics).

2.3 Heat-to-performance metrics limits unicore processing speeds, leading to cost issues.

2.4 Parallelisation allows distribution of tasks into a wider network of non-local systems.



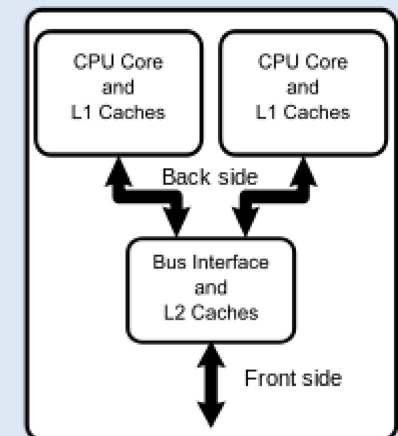
3.0 How Is Parallel Programming Made Possible in Hardware and Architecture?

3.1 In individual systems units, hardware parallelisation is implemented in multiprocessor and multicore architectures.

3.2 In a tightly-coupled distributed system (e.g., cluster computing) the system can be considered as a single unit.

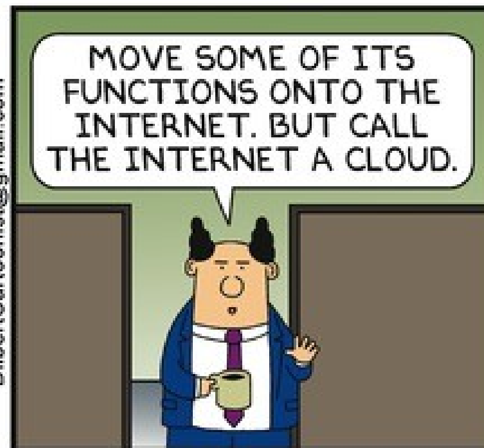
3.3 In a loosely-coupled distributed system (e.g., grid computing) the system is a network of independent units.

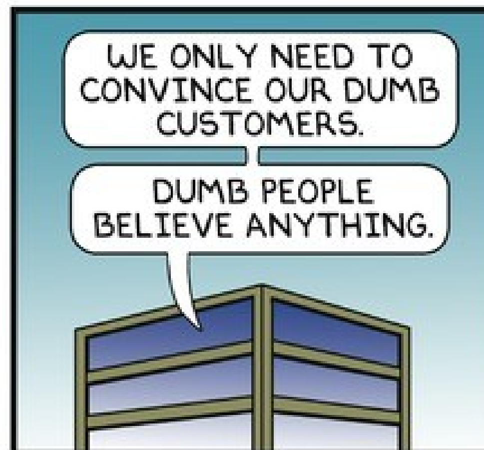
3.4 In a virtualised distributed system (e.g., cloud computing) the system is an on-demand network of independent units.



3.0 How Is Parallel Programming Made Possible in Hardware and Architecture?

DILBERT BY SCOTT ADAMS





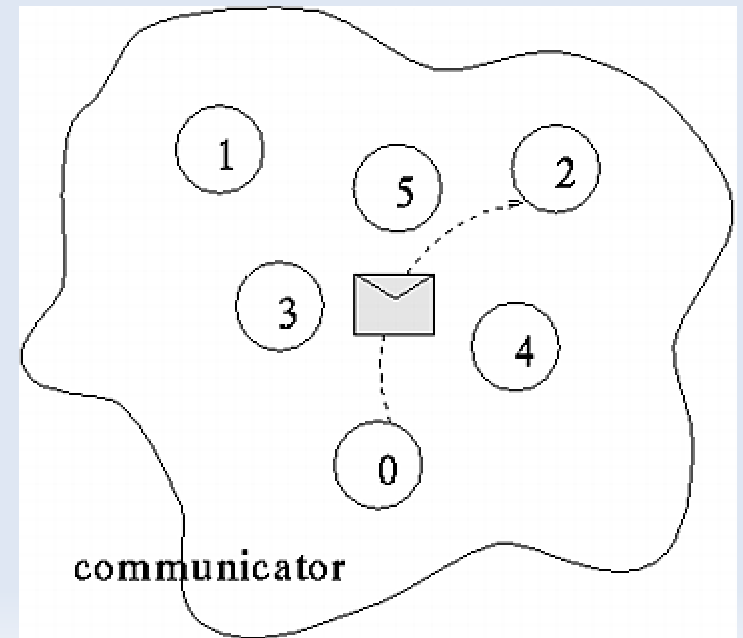
DilbertCartoonist@gmail.com
© 2012 Scott Adams, Inc./Dist. by Universal Uclick
www.dilbert.com
10-21-12

4.0 How Is Parallel Programming Implemented in Software?

4.1 Automatic parallelisation of a sequential program by a language's internal logic would be very beneficial and is much desired. Alas, it is very elusive.

4.2 There are a very large variety of concurrent programming languages, libraries, API etc. which implement parallelisation in a variety of ways.

4.3 Most common implementations are through multithreaded programming in a shared memory environment (e.g., OpenMP) and message passing in a distributed memory system (e.g., OpenMPI). The former uses a fork-join procedure and the latter by building a communications world.



4.4 A sequential program can be implemented in a batch system to invoke multiple instances simultaneously over different data sets ("data parallelism" versus "task parallelism")

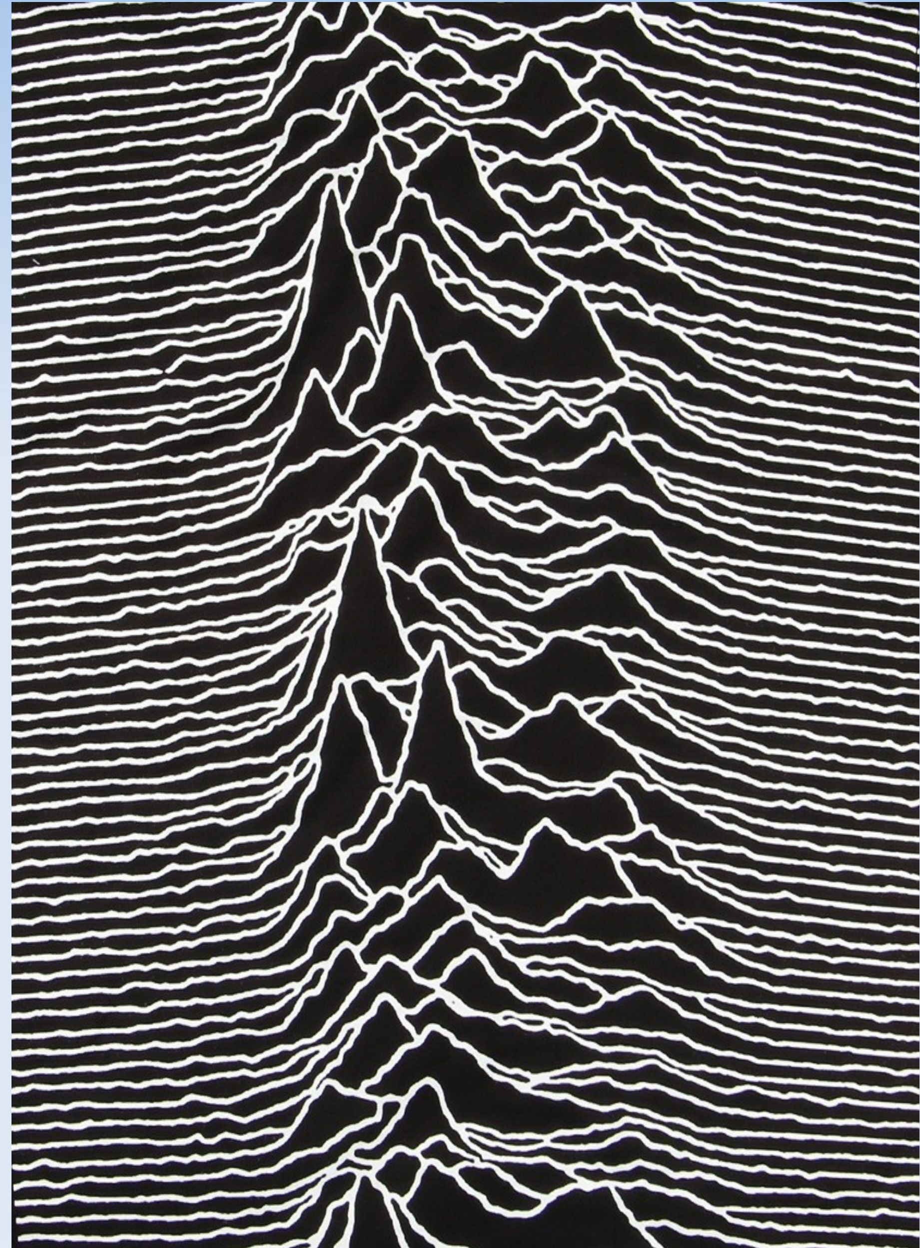
5.0 Embarrassing Pleasures

5.1 A computation problem where it is relatively simple to implement the task into separate parallel problems is "embarrassingly parallel" (or, more optimistically, "pleasingly parallel"). Usually the data or tasks can be carried out independently, simple domain decomposition.

5.2 Example #1: Generation of random numbers in Octave using PBS job arrays.

5.3 Example #2: OpenMP "hello world" thread examples using C and Fortran

5.4 Example #3: OpenMPI "hello world" tasks using C and Fortran.



6.0 Race Conditions, Locks, Profiling

6.1 General rule: programming is hard, parallel programming is really hard. Always start with a working serial program and then determine what parts can become parallel.

6.2 Less pleasing parallel ("more complex") problems require communication between tasks or changes to a shared dataset. This can be lead to race conditions.

6.3 To avoid race conditions barriers and locks can be put into place. These can either slow down the program (barrier) or can lead to deadlocks with a mutual exclusion problem (e.g., apocryphal Kansas railway statute) or livelocks with a circular wait condition (polite people in a corridor problem).

6.4 Example #4: Game theory example in C and Fortran, with MPI_Wait routines.

6.5 Profiling parallel programs needs to consider load balancing between concurrent tasks (but increases fine-grain complexity), reducing I/O (much worse that memory operations), using profilers and debuggers (e.g., PDT/TAU, Valgrind, GDB).

7.0 Parallel Computation Limits and Solutions

7.1 The benefit from parallelisation can be computed as a ratio: Speedup (p) = Time (serial)/ Time (parallel)

7.2 However parallel compution must include some serial component. This component sets a limit on the advantage gained from parallelisation. i.e., S(N) = 1 / (1-P) + (P/N), Amdahl's Law

Cores	Mallacoota	Sydney	Total Time
1	8 hours	+8 hours	16 hours
2	8 hours	+4 hours	12 hours
4	8 hours	+2 hours	10 hours
8	8 hours	+1 hour	9 hours
..	..	..	..
Inf	8 hours	+nil	8 hours

7.3 The Gustafon-Barsis solution to Amdahl's Law is to effectively to reduce the serial proportion by making the parallel tasks larger. Why stop at Sydney?

8.0 Further Examples of Parallel Computation

8.1 Example #5: R with SNOW library

R is usually a single-core application. However using the Rmpi and SNOW libraries, MPI functionality can be provided. The R script creates a random data set and samples, fits the model to the samples, and eventually the mean squared difference – with a test to ensure that the results are equal.

8.2 Example #6: Octave with Parallel and MPI examples

Like R, Octave by default is a unicore application but can be extended with packages to provide multicore functionality, specifically the parallel and MPI packages. These two examples create a vector from basic function and make a calculation of Pi.

8.3 Example #7: Python with Multiprocessor Package Examples

Three examples (courtesy of Praetorian) generate N random integers, and calculate the sum of the generated integers. The timed first example is without any multicore functionality. The second and third approach uses subprocesses from the Python, with the latter making use of a pool.

THANKS FOR WATCHING
& LISTENING PATIENTLY
makeameme.org