Concertant LLP

Processors Processors Everywhere, But How Do I Actually Use Them?

Dr Russel Winder

Partner, Concertant LLP

russel.winder@concertant.com

Copyright © 2009 Russel Winder

Aims and Objectives of the Session

- Investigate some of the consequences for programming of the "Multicore Revolution".
- Compare and contrast various features for harnessing parallelism offered by some programming languages.
- Show that shared-memory multithreading is too low-level a technique for use in applications programming.

*Have a structured "chin wag" that is (hopefully) both illuminating **and** enlightening.*

Copyright © 2009 Russel Winder

Structure of the Session

- (Very) briefly summarize the world of parallel computing.
- Look at some of the concurrency and parallelism support features in C, C++, Fortran, Java, Scala, Python, Erlang, Haskell, Clojure, possibly even Groovy . . .
- Introduce the (supposedly) next generation languages: X10, Chapel, Fortress.

Gant will not appear in this presentation, but SCons will.

There is an element of dynamic binding to the session so the above is just an initial guide.

Copyright © 2009 Russel Winder

Protocol for the Session

- A sequence of slides, interrupted by looking at various bits of code.
- Example executions of code – with the illuminating presence of a system monitor.
- Questions *and answers* from the audience as and when they crop up.

If an interaction looks like it is getting too involved, we reserve the right to stack it for after the session.

Copyright © 2009 Russel Winder

NB

- The sesssion is not about:
  - Algorithms – but they are crucial.
  - Hardware – but it is essential.
- This is a comparative programming languages session:
  - Looking for “emergent properties” in the directions programming languages and their uses are heading.

Copyright © 2009 Russel Winder

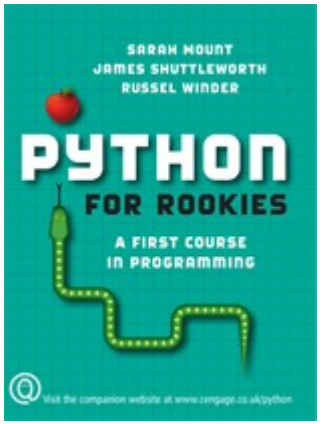
Blatant Advertising

Python for Rookies

Sarah Mount, James Shuttleworth and Russel Winder

Thomson Learning

Now called Cengage Learning.



Learners of Python need this book.

Copyright © 2009 Russel Winder

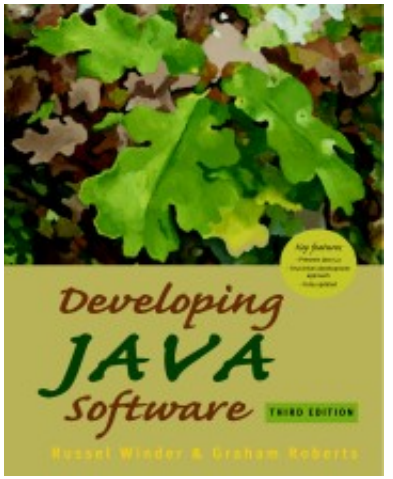
Further Blatant Advertising

Developing Java Software

Third Edition

Russel Winder and Graham Roberts

Wiley



Learners of Java need this book.

Copyright © 2009 Russel Winder

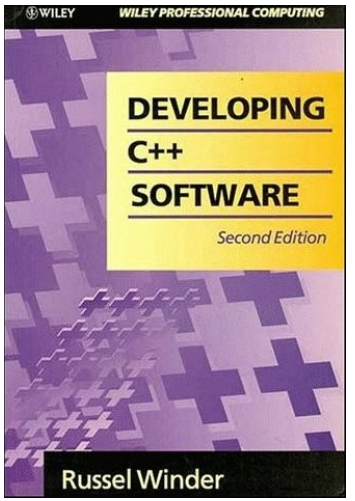
Anti Advertising

Developing C++ Software

Second Edition

Russel Winder

Wiley



Only buy this book if you are studying the history of C++ and how it has been taught.

Learners of C++ used to need this book.

Copyright © 2009 Russel Winder

The Keywords

- Multiprogramming
- Multitasking
- Multithreading
- Concurrency
- Multiprocessing
- Parallelism

- SISD
- SIMD
- MISD
- MIMD

- SPMD
- SMMP

- Data parallel
- Systolic array

- Tightly coupled
- Loosely coupled

Flynn's Taxonomy

- Multicore
- Vector processor
- Cluster

- Livelock
- Deadlock

Copyright © 2009 Russel Winder

A Bit of History
In the Beginning

- Computer hardware was expensive and hard to find; maximizing utilization was critical.
- Multitasking (multiple concurrent processes) was crucial for maximizing availability and utilization.

- Virtualization was flirted with, but, except on IBM machines, it did not become mainstream – though recently it has become *de rigueur*, or at least fashionable, mostly as it is an income stream for a number of companies.

Copyright © 2009 Russel Winder

A Bit of History
The Early Middle Period

- Tasks/processes seen as too heavyweight.
- Lightweight processes introduced.
- Lightweight processes became threads.
- Issues:
  - Tasks/processes had hardware and operating system support, threads did not.
  - Tasks/processes have separate memory; threads are a shared memory approach.

Copyright © 2009 Russel Winder

A Bit of History
The Late Middle Period

- Sun LWP
- PThreads
- . . .
- Boost.Thread
- Java (java.lang.Thread, etc.)
- ???

C# just copies Java.

Operating systems now treat both
***processes** and **threads** as core features.*

Copyright © 2009 Russel Winder

A Bit of History
The Multicore Revolution

- Processors cannot be clocked higher than around 4GHz without generating more heat than can be got rid of by conventional airflow techniques:
  - Users don't want water-cooled workstations.
  - Users don't want burning laptops.
- Manufacturer "doublethink":
  - More cores running at nearly the same speed as the old single cores mean more instructions executed per second, which means ***faster processors***.
  - Marketing has switched general expectations from *faster clocks* to *more cores*.

Copyright © 2009 Russel Winder

An Effect of the Multicore Revolution

- A "downside": processor chips are being clocked slower now than they used to be: single processors now run slower than they used to. This means ***single threaded applications run slower***.
- The days of "just wait for the next generation of processor chips" is no longer a viable way of getting improved performance.
- Application performance improvement can only be achieved by harnessing parallelism.

Or a switch to gallium arsenide technology.

Copyright © 2009 Russel Winder

Threads as Parallelism Abstraction

- Operating systems (well Linux and Solaris at least) provide kernel-level threads that give application access to all the cores on a machine.
- Does not deal with clusters; threads are shared memory single (operating system) process parallelism.

Copyright © 2009 Russel Winder

A Problem – π

- We know the value of π exactly, it's π (obviously).
- What is it's value represented as a floating point number?
  - We can only obtain an approximation.
  - A plethora of possible algorithms to choose from, a popular one is to employ the following integral equation.

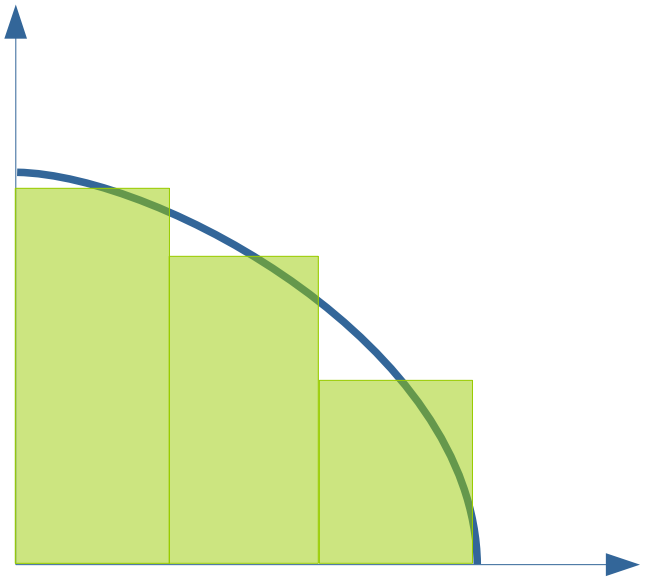
$$\frac{\pi}{4} = \int_0^1 \frac{1}{1+x^2}\,dx$$

Copyright © 2009 Russel Winder

A Problem Solved

- Use quadrature to estimate the value of the integral – which is the area under the curve.

$$\pi = \frac{4}{n} \sum_{i=1}^{n} \frac{1}{1 + \left(\frac{i - 0.5}{n}\right)^2}$$



With n = 3 not much to do, but potentially lots of error.

Copyright © 2009 Russel Winder

Sequential Implementation

- Can see the sequential implementation in any and/or all of:
  - C, C++, Fortran, D
  - Java, Scala
  - C#
  - **Groovy**, Clojure
  - Python, Ruby
  - Erlang, Haskell

There is a rationale to the clustering.

Copyright © 2009 Russel Winder

Sequential is Not Enough

- The problem is *embarrassingly parallel*.
- Improve performance by throwing more processors at the problem, after all we have plenty of them – post the start of the Multicore Revolution anyway.

*Any summation-like problem where the terms are not interdependent is **embarrassingly parallel**. Nearest neighbour algorithms are far trickier – and therefore far more interesting. This is a topic for a completely separate presentation.*

Copyright © 2009 Russel Winder

Where to Start

- C User Group.
- European C++ User Group (ECUG).
- The above merge to become Association of C and C++ Users.
- The above relabels itself to ACCU in acceptance of the fact that C and C++ are not the future.

Obviously we should start with C since it can only get better from there.

Copyright © 2009 Russel Winder

Doing Things with C

- Use a threading package.
- Usually PThreads, but there are others.
- Involves lots of code, and global variables:
  - Use global variables for parameters and return values.
  - Use parameters for parameters and global variables for return values.
- Global state means synchronization and mutexes.

Copyright © 2009 Russel Winder

C++ Does Thing Better

- Can do things as in C, but why?
- Can use the power of C++:
  - Boost.Thread
  - C++0x

Anthony Williams is ***The Man****.*

cf. following page.

Copyright © 2009 Russel Winder

C++0x

- Anthony Williams (Just Software Solutions) has an implementation of the new standards threads for use with current compilers.
- See *http://www.stdthread.co.uk*

***Futures** are the future – for threads-based programming.*

Copyright © 2009 Russel Winder

Java Gets it Right in the First Place (?)

- Java integrated multithreading from the very outset – which is one better than C and C++.
- Java also introduced checked and unchecked exceptions, but checked exceptions are not obviously a good thing.

Does anyone actually use Java's RMI?

C# just copies Java.

Copyright © 2009 Russel Winder

Parallelism Properly (?) Harnessed

- Supercomputing and HPC dominated by Fortran, with some C, and a little bit of C++ (but increasing).
- Thread and process management the real problems:
  - Process management leads to MPI.
  - Thread management leads to OpenMP.

Real Programmers *use FORTRAN or possibly Fortran.*

GPUs lead to OpenGL and GPGPU leads to OpenCL, for this session we ignore this sort of parallelism.

Copyright © 2009 Russel Winder

Fortran child of FORTRAN

- I begat II begat III begate IV begat 66 begat G begat H begat 77 begat 90 begat 95 begat 2003 . . .
  - In the beginning it was named FORTRAN. From 1990 onwards they admitted the existence of lower case letters and it was called Fortran.
- 90 also begat HPF, but that has basically gone away – in favour of OpenMP.

"I don't know what the programming language of the next century will look like, but it will be called Fortran."

Copyright © 2009 Russel Winder

What have the HPC People Really Done?

- OpenMP
  - Data parallelism.
  - Fine-grain parallelism.
  - Thread implemented.
- MPI:
  - Message passing.
  - Cluster-level parallelism.
  - Process implemented.

Replacing array element operations with whole array operations in Fortran was a revolution.

Abstraction is what it is all about. Let the library do the work.

Copyright © 2009 Russel Winder

Manipulating Threads is . . .

- . . . low-level manual labour.
- Working with threads directly is for programmers who don't believe in abstraction – so why do C and C++ programmers program threads directly?

Copyright © 2009 Russel Winder

C and C++ Programmers Learn . . .

- . . . from Fortran:
  - OpenMP and MPI available in C.
  - OpenMP and MPI available in C++.
  - Boost.MPI makes MPI more C++esque.

Copyright © 2009 Russel Winder

Thread Safety

- Java really brought thread-based programming to the masses.
- C and C++ have threads but they are add ons (well C++0x brings threads to C++ at least).
- Java brought “thread safety” front and centre:
  - Library classes made thread safe.
  - Programmers taught to think about concurrency and thread safety.
  - Thread safety kills performance of single-threaded applications.

Copyright © 2009 Russel Winder

Dissension

- Concurrency in Java is hard:
  - Threads is the biggest problem in all Java training.
  - java.util.concurrency as in Java 5.0 still doesn't make it easy enough.
- Occam and Erlang never bought into the threads model for programming:
  - Message passing is the only model.
  - No synchronization because there is no shared memory.

Copyright © 2009 Russel Winder

The Problems of Threads

- Shared memory.
- Flow control.
- Shared memory.
- Flow control.
- Shared memory.
- Flow control.
- Shared memory.

- Flow control.
- Shared memory.
- Flow control.
- Shared memory.
- Flow control.
- Share memory.
- Flow control.

Copyright © 2009 Russel Winder

Quick Quiz

- Who said:

  "Multithreaded programming is fraught with many challenges, and can rightly be considered something that the majority of programmers should steer clear of."

Copyright © 2009 Russel Winder

Consequences

- All computers are parallel processors.
- Multicore processors and multiprocessor systems offer threads as the mechanism of control.

- The majority of programmers should steer clear of computers.

- Only write single-threaded programs.

This will bring us to CSP.

Copyright © 2009 Russel Winder

What To Do?

- Refuse to use multiple threads.
- Never use shared mutable state.

$\Longrightarrow$

- Always use message passing.

MPI is not the only solution here . . .

Copyright © 2009 Russel Winder

Intel's Answer
Threading Building Blocks (TBB)

- C++ template library.
- Provide an alternative to OpenMP as a way of managing threads.
- Shared memory and threading but from a high-level perspective.

TBB doesn't just deal with threads, it also manages the core and processor caches and core instruction streams. It is very Intel processor specific.

There be AMD chip, Sun chip, IBM chip versions of the library.

Copyright © 2009 Russel Winder

TBB: Morals and Politics

- The architecture and approach is good:
  - High-level expression of algorithm.
  - Dependence on significant work from the compiler to make things efficient.
- Libraries mean lock in:
  - TBB is a tool to try and make it impossible for system manufacturers to use anything other than Intel chips?

Copyright © 2009 Russel Winder

Java is Ahead of the Game?

- Java programmers have already recognized the deficiencies of explicit thread programming to harness parallelism.
- JSR 166 X (introduced into Java 6.0) introduced data structures and other infrastructure for better implementation of parallel algorithms.
- JSR 166 Y (due to be in Java 7.0) takes things further.

Copyright © 2009 Russel Winder

Java Learns from Erlang

- Erlang quietly proved that functional languages using CSP were wonderfully effective and efficacious.
- CSP – Communicating Sequential Processes – is a mathematically-backed way of properly partitioning problems to avoid the problems of synchronizing in a shared memory multithreading system.

Calculus of Communicating Systems (CCS) appears to have got lost somewhere.

π-Calculus has some future integrated with CSP.

Copyright © 2009 Russel Winder

Functional Is Good . . . Or Is It?

- Functional programming separates program from execution – uses some form of graph reduction as operational semantics.
- This separation and the reliance on referentially transparent, side-effect free code is supposed to make functional languages good for parallelism.
- Erlang ✔
- Haskell ?
- Clojure ?

Not having side effects is always good for parallelism even in C, C++, Fortran, Java, etc.

Copyright © 2009 Russel Winder

Scala Gets it Right?

- Scala allows shared memory multithreading just as Java does.
- Scala also provides infrastructure for using the Actor Model.

Actors are processes that can send messages to each other.
They can also create new actors.

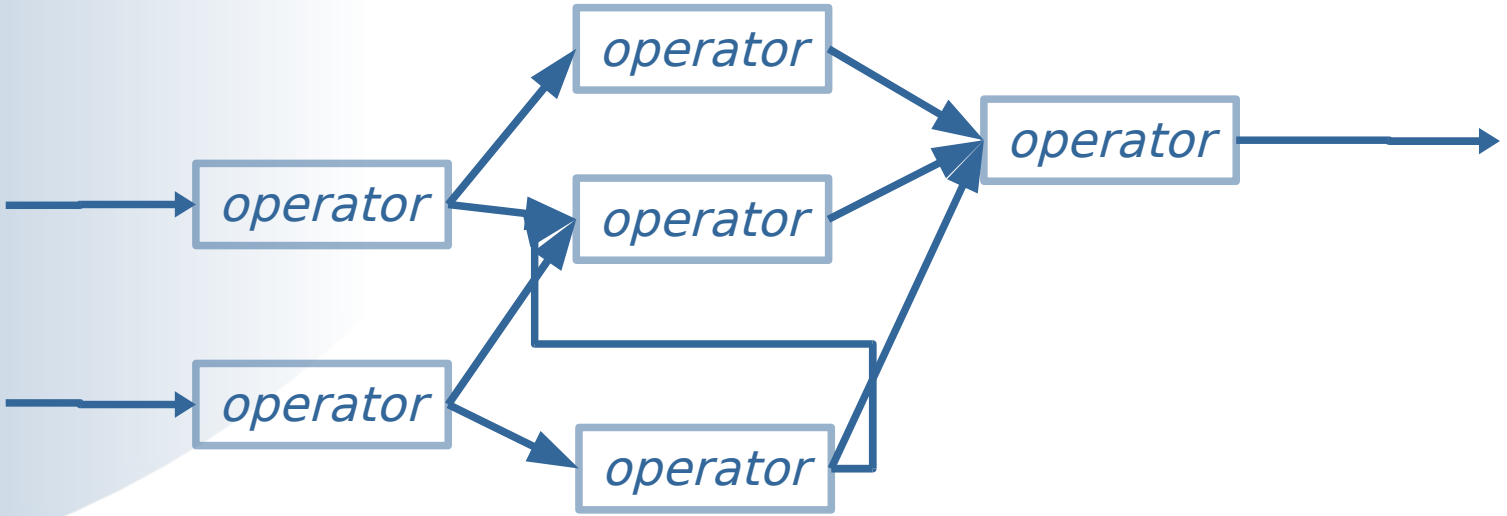
Copyright © 2009 Russel Winder

Flowing the Data

- Dataflow computers were tried but never really broke through.
- Dataflow architectures do not need dataflow hardware.
- Dataflow architectures allow for rampant parallelism.

Copyright © 2009 Russel Winder

The Dataflow Model



Operators operate only when all their inputs are ready, and then output values on their outputs.

*Pervasive Software have created **DataRush**.*

Copyright © 2009 Russel Winder

Dataflow: The Lessons

- Eschewing shared memory and threads in favour of process-based message passing appears to be an overhead for small problems and/or for few processors.
- Process based, message passing appears to scale to large numbers of processes and processors.
- Message passing architectures lead to better scalability.
- Integrating control flow with data readiness makes things even easier and more scalable.
- The paradigm shift is not easy.

Copyright © 2009 Russel Winder

Parallelism can be Pythonic

- Python is compiled to PVM bytecode.
- CPython implementation of the PVM enforces execution of one Python thread at a time – there is the Global Interpreter Lock (GIL).
- Stackless Python doesn't have this.
- But is it a problem?
- Microprocessing package (new in Python 2.6, previously it was the separate Processing package).
- Parallel Python.

The PVM is not PVM.

OCaml also uses a "master lock" to enforce single threading at a time.

Copyright © 2009 Russel Winder

PLNG
Programming Languages, the Next Generation

- DARPA HPCS programme
- First funding round:
  - X10
  - Chapel
  - Fortress
- Second funding round:
  - X10
  - Chapel

Defense Advanced Research Programmes Agency

High Productivity Computing Systems

Unified Parallel C (UPC)
Co-Array Fortran
Titanium

Copyright © 2009 Russel Winder

Programming Models

- Shared Memory
  - Threads
  - Processes
- Distributed Memory
  - Processes
- Partitioned Global Address Space
  - Threads with affinity.

Global view systems have become the standard approach – it avoids manual partitioning.

Copyright © 2009 Russel Winder

PLNG – X10

http://x10.codehaus.org

- IBM and various unnamed academics.
- JVM and Java based – well X10 1.5 was.
- X10 1.7 looks more like Scala and there is now a C++-based as well as a Java-based code generation system.
- Introduces a partitioned global address space:
  - Activity is bound to a processor but can move.
  - Affinity of activity and processor is controllable.

X10 is so named because its goal is to create an order of magnitude improvement on performance compared to Java.

Copyright © 2009 Russel Winder

PLNG – Chapel

http://chapel.cs.washington.org

- Cray and Washington University.
- New language – but with a nod to Python, Scala, C++ and Java.
- Focused on being a language that Fortran, C and C++ programmers can migrate to.

Copyright © 2009 Russel Winder

PLNG – Fortress

http://projectfortress.sun.com/Projects/Community

- Sun and ???.
- JVM based.
- The goal is to replace Fortran:
  - Definitely aimed at the more mathematical end of things.
  - Remove the reliance on ASCII-based representation of algorithm.

Copyright © 2009 Russel Winder

Side-step the Shared Memory Problem: Hardware

- Sun is putting support for transactional memory in hardware – the Rock processor.
- PowerPC, ARM, etc. have support for hardware-supported software transactional memory.

Copyright © 2009 Russel Winder

Side-step the Shared Memory Problem: Software Transactional Memory (STM)

- Haskell supports STM.
- Clojure supports STM.
- Intel compiler offers STM to C++ users.

The primary example of STM in the Clojure documentation is to implement pmap. Why not just treat STM as an infrastructure implementation tool and use pmap for implementing the algorithm?

Copyright © 2009 Russel Winder

Transactional Memory – Why Bother?

- Given that lightweight process and message passing based approaches work, why hassle with application use of shared memory in a multithreaded context?

The primary example of STM in the Clojure documentation is to implement pmap. Why not just treat STM as an infrastructure implementation tool and use pmap for implementing the algorithm?

Copyright © 2009 Russel Winder

Summary – A

- Threads are useful but they are not an application programmer tool.
- Why bother with an explicit threads API when OpenMP is available?
- Why not use a language that supports parallelism directly:
  - Erlang, occam, Scala
  - C++ + TBB, Java + JSR 166
  - Chapel, Fortress

Copyright © 2009 Russel Winder

Summary – B

- ***Focus on the data and its transformation not on the sequence of actions to achieve the transformation.***

Intel C++, Java, Erlang and Scala all go this way and point the direction.

If computing is about abstraction let's do it.

Copyright © 2009 Russel Winder

Postscript – An Interesting Moral

- ***RAII*** (resource acquisition is initialization) is C++'s Big Win in the abstraction development stakes.

Copyright © 2009 Russel Winder