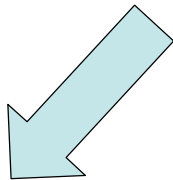


Supercomputing and You

Stephen Cope
Department of Statistics, Auckland



I am Stephen Cope, from the Department of Statistics here at Auckland.

Originally I was hired as the go-between who helped academics get their work onto our website, although lately my role has involved helping academics getting their work onto our computing facilities.

(The aim for this slide is to be standing so the arrow, when it appears, will be pointing at me.)

Look Around You

Outline of this talk

- First, the news
- Second, how to use it
- Finally, questions

I've split this talk into a few sections.

In the first half I've got a brief overview of some news in supercomputing.

In the second half I've got some ways you can apply that, and take better advantage of what we have access to.

In the third half I hope I will have ample time to field questions, briefly discuss any problems you're looking for solve, and we'll try and figure out how this talk manages to have **three** halves.

I bring news from afar

- Attended HPRS: High Performance Research Symposium
- Attended APAN26: **Sustainable Networking**
“New Zealand’s premier digital event for 2008 - showcases first-hand how ICT developments and advanced networks like KAREN are accelerating research and collaboration in the Asia Pacific region and are contributing towards environmental, regional and operational sustainability.”

At the beginning of August I was lucky enough to attend a symposium attached to a conference.

The symposium was the “High Performance Research Symposium”, put on by the University of Canterbury who have the BlueFern supercomputer. During the symposium famous names in Grid and supercomputing talked about trends and what challenges lie ahead. The afternoon of the symposium was spent looking at how HPC was making researchers’ lives easier, or giving them more work to do.

(I should note that “Supercomputing” isn’t quite as scary as it sounds. It’s probably better to use the term “High Performance Computing”.)

The conference was themed, “Sustainable Networking”, and there was quite a lot of buzz about the most recent in trendy things, such as “carbon credits” and “carbon neutral”. Attendees tended to walk to the evenings’ activities, rather than taking the supplied buses.

CPU Increase in \wedge power

- Moore's Law
- Increase in power consumption
- Processing power per watt



Gordon E. Moore, Co-founder, Intel Corporation.
Copyright 2005 Intel Corporation.

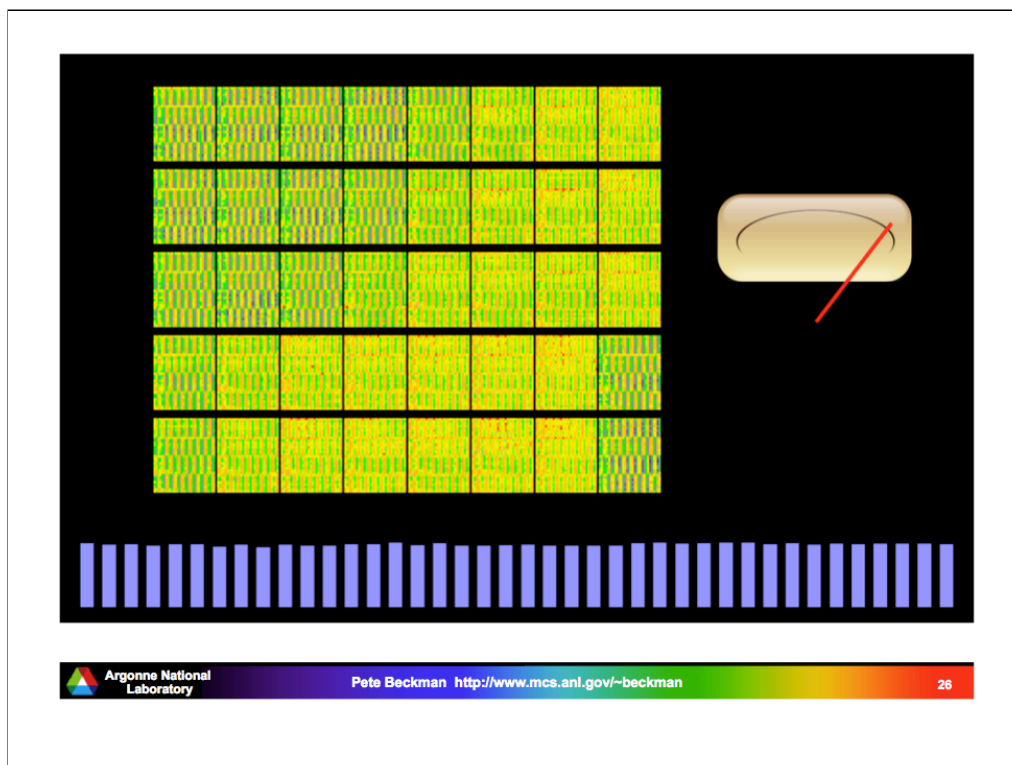
There was quite a lot of talk about the increase in power.

This is not just CPU power, you may have heard Moore's Law, first described over 40 years ago by this dashing boffin, but also electrical power.

IT & communications are the 5th largest user of electrical power in the world. (ref. An Inefficient Truth, Global Action Plan) In the closing plenary it was suggested people with large server rooms should buy their own hydro-electric power plant. The speaker had more sensible solutions too.

Question from the audience (James): What sort of wattage is a typical CPU using these days? No hard figures on hand, however most desktops are shipping with 300 watt power supplies.

With electricity being used in such large volumes, processing power per watt is becoming a more useful metric for comparing CPUs.



Here's a slide by Pete Beckman, Argonne National Laboratory. It was in his presentation "The Path to Exascale Computing".

It shows power consumption, by CPU, in the different racks in BlueGene/P. Different algorithms cause different patterns of power usage.

The difference between idle and busy power usage can be a factor of 3. How power efficient is your algorithm?

UK – JISC study

- Why the future's green for IT at universities
 - <http://education.guardian.co.uk/link/story/0,,2278356,00.html>
- Green IT is best achieved through the collaboration of IT and campus facilities management – power, heat and real estate
 - Most researchers are not aware of true costs of computation such as power, cooling, and specialized buildings.
- Increased energy and computing costs can be offset by technologies such as grid computing and virtualisation.
 - "Eighty to 90% of a computer's capacity is wasted."
- Cardiff University solution to the cost of running super computers for research projects by centralizing departments' IT budgets and transferring byte-hungry number-crunching to clusters of smaller high-performance computers.

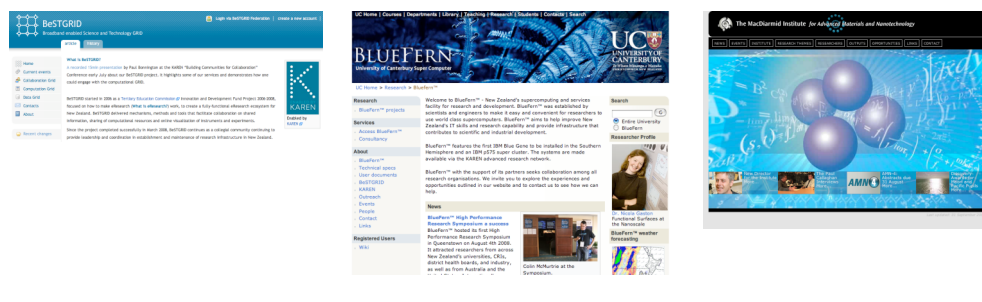
This slide is by Bill St Arnaud, CANARIE Inc. from his presentation "How International R&E networks can help reduce global warming" which he delivered by video conference from Canada.

JISC is "Joint Information System Committee".

Much computer time is wasted, so the best thing to do is to centralise and combine computing resources.

Aggregation of servers

- Instead of distributed CPUs, combine them into compute farms
- Shared between organisations
- CPU time purchased from another org.



Rather than dispersing the processing power around, each researcher having a powerful computer, combine them together into Department-wide compute farms. Why stop at the Department level? Departments combined into Faculties, Faculties combined into organisations, and then ...

... shared between organisations. For example, BeSTGRID. There are computing facilities here in Auckland, down in Canterbury with their BlueFern supercomputer, and elsewhere.

MacDiarmid Institute found it was better to purchase time from UC's BlueFern, and from the NZ Supercomputer Centre (Weta).

Increase in data

- Shifting the data
 - KAREN
- Storing the data
 - BeSTGRID
 - NZ Social Science Data Service
 - BioGRID
- Processing the data
 - BeSTGRID
 - Grid



For whatever reason we're getting more data now. Perhaps we're just better at collecting it?

Fortunately, this University does have access to some fairly good technology when it comes to shifting the data around. We, and most other NZ Universities are attached to KAREN (www.karen.net.nz) which is pretty fast. I was in Queenstown and only 20ms away from my desktop here in Auckland, thanks to KAREN.

When that data arrives somewhere it needs to be stored.

-One of the three aims of BeSTGRID, another thing this University is part of, is to store large volumes of data. Some of us already use that to squirrel away an odd half terabyte or so.

-There are libraries such as the NZSSDA (www.nzssds.org.nz), which one of our alumni Martin von Randow has worked on. It was spoken of quite highly.

-Also BioGrid or the Bio Informatics Grid (www.bioinformatics.org.nz) also sits on top of BeSTGRID. Stephane Guindon has something to do with this.

(continues)

(continued)

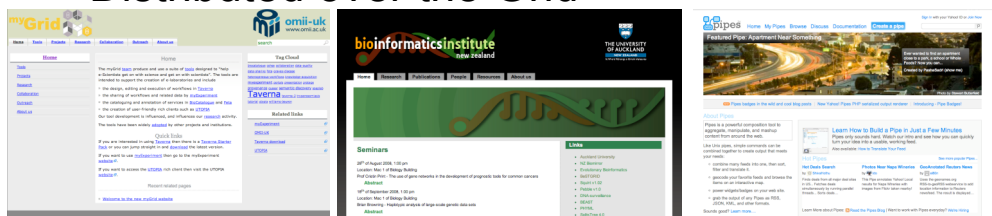
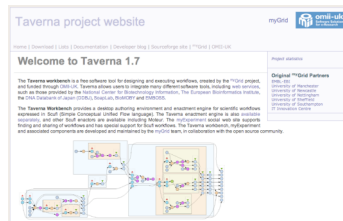
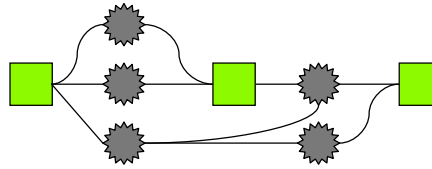
Of course, you also need to process the data, and that's where BeSTGRID is again useful. BlueFern devotes a percentage of their CPU time to BeSTGRID jobs, and within the University, at the Centre for eResearch, Nick Jones and his team are working on getting a SGI Altix attached to BeSTGRID.

There's also our Department's Grid, and Mathematics has an Xgrid setup by Kormal Prasad, and over in Computer Science they have their own sort of grid or cluster.

I like the word "Grid", so I've added it another time on the bottom line here. Hopefully, we can ignore the technology from now on, and just concentrate on what we're doing on top of that.

Workflows

- Sharing workflows
 - Repeated, reused
- Taverna
- myGrid.org.uk
- NZ BioInformatics Grid
- Distributed over the Grid



Workflows are where input data arrives, gets put through a number of processing steps, and a result flies out the far end.

If you've got a great multi-step workflow that provides a useful result, then either YOU are going to reuse it multiple times - in which case if it was easy to reuse then that would be great - or other people are going to use it - in which case it would be great if that was easily shared for them to use as well.

Workflows are also popular with users. Workflows are attractive. The high-level abstraction is convenient. Also it allows for easy distribution over the Grid (each of these processes can run in a different place). (ref. Ian Foster)

Taverna is one example of a piece of software for sharing workflows. myGrid offers a collection of workflows in Taverna which you can use.

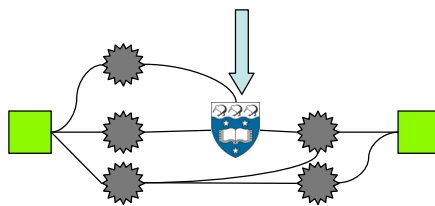
Closer to home, the BioInformatics Institute has a special version of "Geneious" that makes it easy to run your workflows over the Grid.

In a different class, there is also Yahoo! Pipes which is a very simple interface for doing very simple things, but it has the same idea: create a multi-step process for dealing with information and publish the

Published services



- We know about rats
- Why not offer that as a service to those with data?
- Create services, and compose new ones



As an example we know things about, say, rats. Why not offer that as a service to those with the data? Send us your rat DNA, and we'll tell if they're related to the rats in the Bay of Islands. It's a potential service revenue stream.

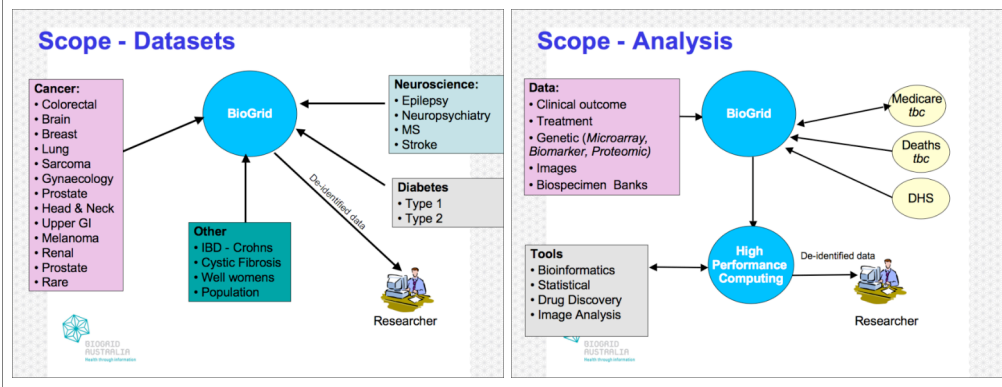
Once our rat DNA comparison service is published and available, other researchers could join it into their own workflow.

Ian Foster talked about “service -oriented science” where services are published so that other researchers can discover them and then plug them all together to make a new service, which can then be published.

Question from audience (James): What about confidentiality of data? Would researchers want to send their confidential data to us? [Refer to next slide for an example of how this could be solved.]

BioGrid Australia

- Pulled together datasets
- Organised analysis
- www.biogrid.org.au



Marienne Hibbert, from BioGrid Australia spoke about the work she has been doing. Here are two slides I extracted from her presentation. You can find the complete presentation on the BlueFern HPRS website.

BioGrid pulls together a lot of highly confidential patient files, links them to records in other silos, and de-identifies them before a Researcher sees them. Medical records about all sorts of things (as shown on left).

BioGrid also handles the analysis. Not only does it have the data, but it also has the HPC facilities and tools to turn the data into something useful for a researcher. (As shown on the right).

This was mainly done in the Australian state of Victoria, which is approximately the same size as New Zealand [in terms of population, number of Universities, other metrics excluding land area]. Victoria is the place to watch for ideas that we could implement here. Are there any New Zealand organisations you can think of who would benefit from this kind of treatment?

Comment from audience (Thomas Lumley): Most of this data can be de-identified. Some of the data James wants to process includes DNA, which cannot be de-identified.

Collaboration

- PulseNet Asia-Pacific: ESR
 - sharing data
 - catching bacteria outbreaks
- AccessGRID, part of BeSTGRID
 - www.accessgrid.org

Dr Brent Gilpin from ESR Christchurch talked about PulseNet, which coordinates fingerprinting of bacteria by using a centralised database. The New Zealand branch is part of PulseNet Asia-Pacific, which is connected up with the main one in the USA. Individually, a single country or state's database of bacteria fingerprints isn't much use in detecting outbreaks of e. coli or salmonella, but when the data is combined together as PulseNet does, outbreaks can be caught much quicker. This translates into less people dying from the latest batch of infected lettuce, tomatoes, or mince meat.

Another form of collaboration is AccessGRID. You and a remote researcher book a time in one of the online meeting rooms, and away you go. Either head for an already set up conference room, or configure your own webcam and microphone.

Catalysts

- Catalyst Team
- Or “Missionary”
- Facilitators for communities

Last slide from the news!

From Argonne National Laboratory we have the fancy new jargon term of a “Catalyst Team”.

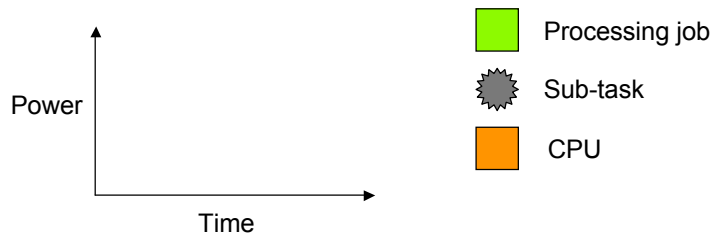
In Victoria they are referred to as “Missionaries”, but the presenter [Bill] admitted that Catalyst was a better sounding title.

These are eager people, such as myself, who will work with a researcher to get their projects working on the HPC facilities. Not only that, but we’ll take an active interest in your program as it runs, and when changes are made to the setup, can check that your programs will still work.

This is what I’ve been for some of you, the “catalyst team” to help get your research run through our HPC facilities.

How to use it

- Conventions for the next section



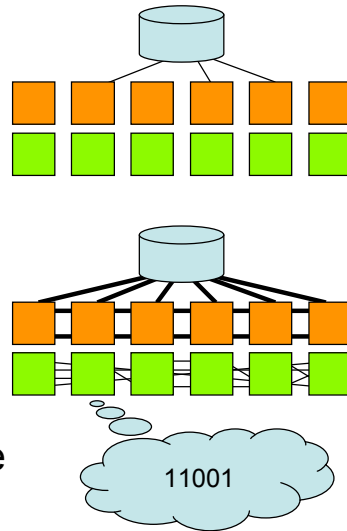
Now I've inspired you to use supercomputing to save the world, let's find out how to do it, and how some people in this Department have already.

There are a few conventions I'm going to use from now on:

- Time runs left to right, processing power goes bottom to top
 - Lime green squares are processing jobs
 - Tasks can contain sub-tasks, which are shown as gears, but most of the time they are synonymous, hence they are not always shown separately
 - Tangerine squares are CPUs, which also map pretty well to tasks, so mostly we'll ignore them
- Let's see how this works ...

Some Jargon

- Capacity computing
 - Loosely coupled
 - Many Linux nodes
- Capability computing
 - Tightly coupled (MPI)
 - Large memory
 - Shared memory
 - SGI Altix, IBM BlueGene



I don't like to burden you with jargon, but there are two different styles of computing that are important to know about.

Capacity computing

This is what we're probably familiar with when we think of the resources here in the Department. There are a bunch of CPUs. There are processes running on those CPUs. There's some shared disk storage in there. Otherwise, they're all on their own, "loosely coupled". A job starts, it runs, it writes its output, it quits.

Capability computing

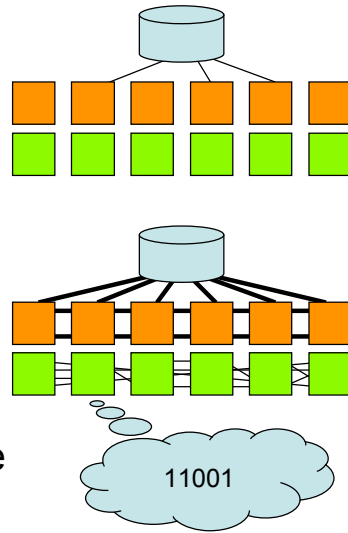
This is a more advanced method of solving problems, and being more advanced it requires more advanced hardware, and more complicated programming methods. Sorry!

Jobs are "tightly coupled". They're communicating with each other using something like MPI. In the system there are large amounts of shared memory (within limitations!). The hardware is probably specially designed for this.

(continues)

Some Jargon

- Capacity computing
 - Loosely coupled
 - Many Linux nodes
- Capability computing
 - Tightly coupled (MPI)
 - Large memory
 - Shared memory
 - SGI Altix, IBM BlueGene



(continued)

Most of our resources are in “capacity” hardware, and we have about 1GB of memory per CPU. We do have some “capability” type hardware. For example, paieka and the new hydra both have fairly large amounts of memory and 8 and 16 CPUs respectively, so it would be possible to run a fairly large task on them with shared memory and message passing.

How have we changed?

- Have we changed?
- Are we still running in serial?

I suppose the audience is best placed to answer this question.

How have we changed the way we run simulations?

From what I can tell, not much.

Take advantage of more CPUs

- More interpolation
- More complex code
- Rewriting your code

We have more CPUs available, so how have people been taking advantage of them?


If you previously used all four of hydra's CPUs to calculate four points, on the new 16-cpu hydra you can calculate 16 points, or even more. Those calculated points on the graph need not be so far apart.

Your code can be more complex. You can calculate to an extra decimal place. Maybe give the Monte Carlo simulation an extra run just to be sure.

You could even rewrite your code to incorporate that feature you've had to leave estimated due to time constraints.

This is a good start. We can do more.

Rewriting your code

- Some code lends itself to parallelisation
 - Permutations of inputs:
 $p(x), x \in \{1, 2, \dots\}$
 - Independent functions:
 $p(x) = f(x) \text{ R } g(x)$
 - Repeated random walks


To take advantage of this new way of computing things, we do have to have a new way of coding things. It isn't much of a difference, depending on how your code is structured. It's also highly dependant on your problem.

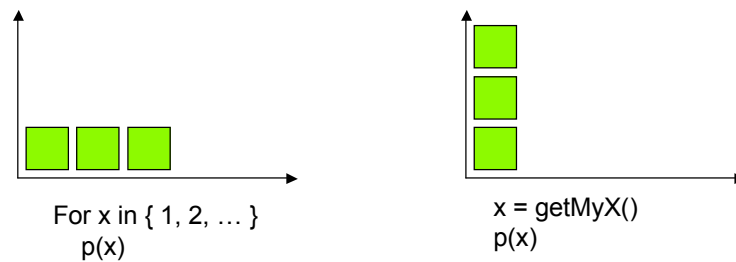
Let's take a look at some really easy ones first.

(list)

Let's look at these better on the next few slides.

Permutations of inputs

- Permutations of inputs:
 $p(x)$, $x \in \{ 1, 2, \dots \}$



Here we have a function that takes a range of inputs. The function does not depend on previous values of itself. The only change is that different values are being fed to it.

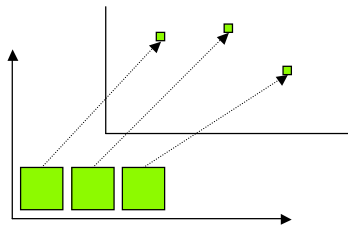
Each input can thus be computing separately.

Instead of having a for loop which runs each one in sequence, how about submitting each value in a separate process? (I have Java and R code for doing this.)

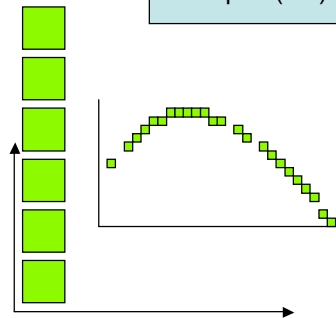
A bogus example illustrating the benefit of parallelism

- More points on the graph in less time
- How's our function behaving?

```
for (x in c(1:4)) {  
  out <- c(x, p(x))  
  print(out) }
```



```
x <- getMyX()  
out <- c(x, p(x))  
print(out)
```



Here's a rather contrived example to illustrate an advantage of running your code in parallel.

Parallelism allows us to get more points on the graph in less time. More points gives you a better idea of how your function is behaving.

A summer research student has given his supervisor the result of his summer of hard labour: a hallowed R function which the researcher will now investigate. Oddly enough it always takes 30 minutes to complete, even on the fastest processor available. Our researcher has a stub of R code, similar to this, which loops through the values being investigated, plugs it into the formula, and waits for the result.

```
R: for (x in c(1:4)) { out <- c(x, p(x)); print(out) }
```

After half an hour the first result arrives and is duly plotted. Can't tell anything from the graph yet with a single point.

Another half hour later, so a total of one hour from the start of the simulation, and the second result arrives. Looks like a positive linear increase.

(continued)

(continued)

For good measure a third point was plotted. Suddenly, surprise, the value has dropped dramatically. What is it showing? Perhaps some sort of quadratic relationship here? A fault in the function? Even more importantly it's an hour and a half since the simulation was started and it's time for a cuppa!

However, a second researcher taking advantage of the power of parallelism has slightly different R code. Each run of the R code causes it to load a previously uncomputed value of x before running it through the student-supplied function. (The R code for doing this is available in the handout notes.)






Half an hour later the results are back and the researcher finds that he has been had. The student merely wrapped up $\sin(x)$ and a call to `sleep()`.

```
task <- as.numeric(Sys.getenv("SGE_TASK_ID"))
x.df <- read.csv("x-px.csv", header=T)
x <- x.df[task,]$x
```

```
out <- c(x, f(x)); print(out)
```

getMyX

- `for (x in c(1:10)) {`
 - `out <- c(x, p(x))`
 - `print(out) }`
- `R --vanilla < serial.R`
- `task <-`
 - `as.numeric(Sys.getenv("SGE_TASK_ID"))`
- `x.df <- read.csv("x-px.csv", header=T)`
- `x <- x.df[task,]$x`
- `out <- c(x, p(x)) print(out)`
- `qsub -t 1-10 run-R parallel.R`

x	p(x)
0.5	
1.0	
1.5	
2.0	
...	

Let's just take a closer look at this function, which I have named "getMyX()", and compare it to a for() loop:

On the left a traditional for() loop. A value of x is retrieved, p(x) is calculated, then output.

To run this through R you would typically do something like this. R vanilla pipe in serial.R.

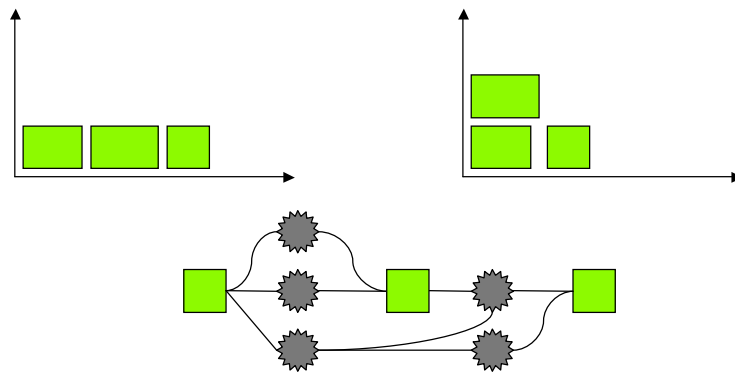
You end up building a table like this, with your input of x, and the value of p(x).

On the right our new way of doing things. Each task when it is run has an environment variable telling it where it is. SGE_TASK_ID. We grab this, convert it to a number, and that tells us which row of this input file we are to be processing. I could have also done task / 2 and got the same answer.

How does R get run with the right tasks numbers? When I submitted the job I used the -t (task) operator and requested task numbers from 1 to 10. The Grid handles when each item gets run.

Independent functions

- Independent sub-functions:
 $p(x) = f(x) \text{ R } g(x)$

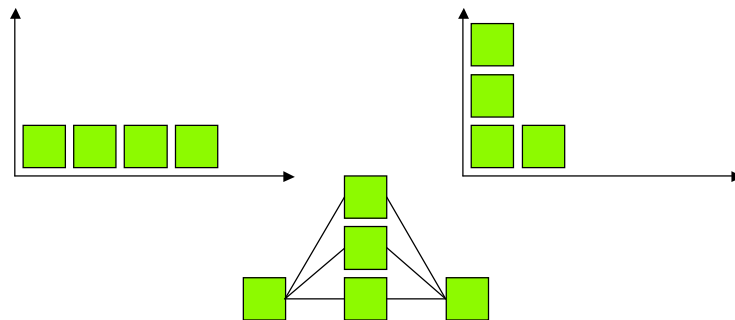


Two fairly involved yet independent [of each other] functions, f and g , whose results are later combined. Rather than running them in series, spin each off to a separate thread and have them run simultaneously. Once they're both calculated, continue on and combine them.

The advantage and potential increase here is a factor of how many independent functions you have to calculate.

Splitting input

- Where
 - $A = \{x_1, x_2, \dots\}$
 - $p(A) = f(x_1) R f(x_2) R \dots$
- Gather-Scatter / Map-Reduce



Here we have a function, p , operating on an input set, A , where an operator R is applied to the results of processing, f . Whether $f(x_1)$ is processed before or after $f(x_2)$ the result is still the same, so it follows that each function $f()$ can be run separately and the results then collected later, with the same final result.

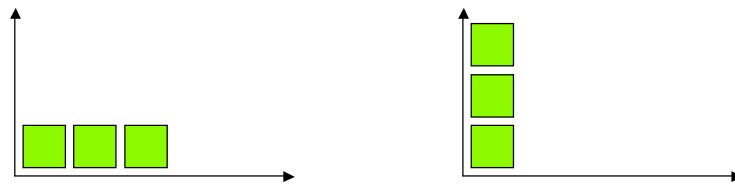
Rather than processing each line sequentially in serial, chunks of the input file can be processed by individual threads. The results are then collected, and the final function applied.

Gather-Scatter is one methodology that has been around for a while. Using a message passing interface (MPI) the input is distributed, scattered, to processors and then the results are gathered back in.

Google reinvented this wheel and called it “Map-Reduce”. They use this to process log files and calculate PageRank. Given the I/O overheads we may not see much of an advantage with this in our environment.

Repeated random walks

– Repeated random walks



It's repeated, so no coding change needed.

What's not to understand about it?

Question from the audience: What about the same seed for random number generators?

Possible solutions: use the JOB_ID and TASK_ID (which will be unique for each run) and include them as part of the seed. Use of a file containing random data is not feasible as it will require a lot of I/O to read it, and a pseudo-random number generator (PRNG) is preferred as it can be reproduced later, given the same seed.

Comment from audience (Ivan Kojadinovich): The snow package in R includes a function called "clusterSetupRNG" which can be used to make sure seeds are different.

Case Study: Jiang & Wild

- Permutations of input
- Two parameters: Alpha, Beta
- Written in R
- Before: `for (alpha) for (beta)`
- Using a template, substitute in the values for Alpha and Beta, then submit
- After: `alpha <- ALPHA; beta <- BETA`

Chris Wild and Yannan Jiang were two of the first users of the Grid.

They were investigating what would happen to a function based on different inputs of two parameters: alpha and beta. It was written in R and originally used a nested for loop to iterate through all possible combinations.

When I was asked to look at it I removed the nested loops and shifted that out to a separate script which would generate R files, based on the values of Alpha and Beta, then submit them all for processing.

Now, a single given job has the values of Alpha and Beta, here substituted in place of “ALPHA” and “BETA” in all-capitals, and calculates only a single result. These are then collected afterwards.

for each - nested for loops

- `for(alpha in c(1:20)/20){`
 - `for(beta in c(1:20)/20){`
 - `p(alpha, beta) }`
- Assuming `p()` does not modify global variables (ie, independent)
 - $\alpha \in \{\alpha_1, \alpha_2, \dots, \alpha_i\}$
 - $\beta \in \{\beta_1, \beta_2, \dots, \beta_j\}$
- $|\alpha| \cdot |\beta|$ combinations

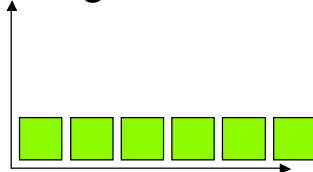
Does this bit of R code look at all similar to anything you've worked on?

Then you have the perfect candidate for parallelism ... assuming the value of the function is independent of previous runs. Ie, it does not modify any global variables.

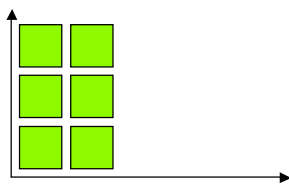
The increase in speed is huge for a nested loop, although a single loop can still be optimised. You're looking at the size of alpha multiplied by the size of beta combinations, which can quickly grow into the thousands.

Spreading out permutations

- This is the easiest way ever
- Using a nested for loop:



- Parallelised via Grid:



	α_1	α_2	α_3	...	α_i
β_1					
β_2					
β_3					
...					
β_j					

This is “embarrassingly” simple to parallelise.

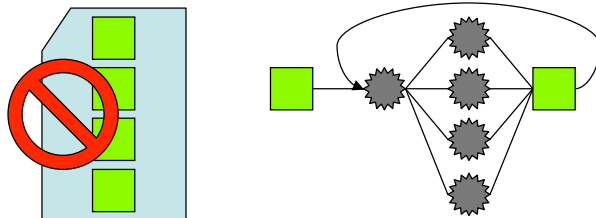
Originally you have a nested loop. One set of input is evaluated at a time.

Parallelise this, and even on this simple graph the speed advantages are already obvious.

Alpha by Beta, that grows pretty quickly. Now what if you’re using three parameters? The gains are even better!

Case Study: Lyndon Walker

- 30 days and still no results
- Could not split the input file
- Multi-threading the independent function
- Optimising another function
- Results arrive within a day



This was one of the trickier ones, but also the most satisfying. The answer was not simply, “THE GRID”.

An input data file is read, then a two-step simulation is run. Due to the nature of the simulation it was not possible to just split the input file so each process worked on a sub-set of the input. The results would have come out incorrect. (Java code written to work on a subset of an input file is available on my website as a result of our first investigation into that.)

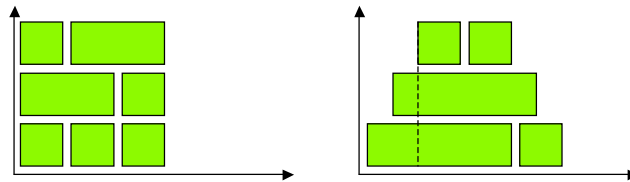
The first step of the loop involves a function that is applied to a single row of the dataset. It has to be applied to each row in the dataset, and is independent. Perfect for multithreading. Using Java’s threading abilities I created a new thread containing the function, and would spawn off a collection of four or five that would process a row each.

The second step of the loop was also very slow. However, it was not able to be broken down into sub-threads. There was an array search that was written with $O(n)$ complexity. A small reshuffle of the data structure and that particular search was reduced to $O(1)$ complexity.

Now results are returned within a day.

An ideal job

- As small as possible
 - Unless it is checkpointed
- Needs to finish before:
 - Mean time between (power) failures / upgrade
 - Your paper is due
- Shorter = more agile, better scheduling



An ideal job for running on the grid is one that is as small (in time) as possible ... unless it is checkpointed and can resume. The Grid is very polite and if it is going to kill or suspend your thread it will give it advance notice of at least a minute. Sometimes it isn't the Grid which causes your job to end prematurely, though.

Your job needs to be short because it needs to finish in less time than the MTBF. Also it helps to have results when you submit a paper, so have your processing finished by then.

Shorter jobs also help with scheduling. As computing resources vary - we have an overnight-only queue - and machines are brought online or taken offline for maintenance, your job needs to be able to move with them.

This graph on the left with the fairly short running jobs is fine. The jobs run, then quit, then another job is scheduled.

This graph on the right has a number of long running jobs. When these short jobs are submitted (dotted line) they have to wait around much longer until the other jobs complete. Maybe those long running jobs could have been split into smaller chunks?

(continued)

What if a server requires maintenance? How much work are you prepared to lose?

What if there is a power failure? (As the monthly power faults for the last three months have demonstrated.)

Can your job be rerun? [Is it writing output to a file which it would then clobber?] How much would you lose?

Output

- Print to the screen
- Allow the Grid to collect your output
- Avoid file contention
- A final step that collects output?
- *"A supercomputer is a device for turning compute-bound problems into I/O-bound problems."* Ken Batcher

The best way to do output is simply to spit your results out to the screen.

The Grid will collect any output that is written to the screen and will put that in a file for you. As the Grid knows what is going on, there will not be locking or contention with other threads attempting to write to the same file.

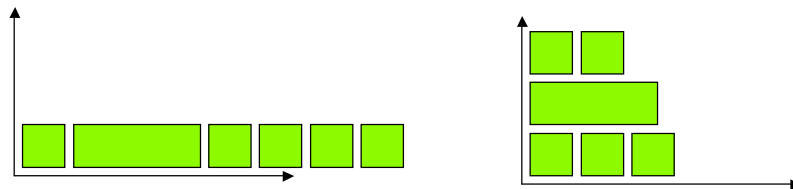
Perhaps you can include a final step that gathers up the input? Can't be too hard.

A rather serious joke from Ken Batcher, a pioneer in the field.

Our Department is not crash hot when it comes to I/O. There is plenty of scratch space on the local disk - check environment variable TMPDIR.

Case Study: Mark Holmes

- A week and a half and still running
- Inspection showed nested for loops
- Mixture of templating and variable retrieval
- After: `alpha <- ALPHA; beta <- getMyX()`
- Results arrived within a half hour



Mark Holmes had a number of simulations running. I saw them running and after a week and a half one died unexpectedly.

Mark came to me then and on inspection I found nested `for()` loops giving different values to a function, whose value was independent of previous runs.

I then wrote some scripts to put together a mixture of templating (where the value of one variable is inserted into a template and used to make the new file) and task numbers, where Grid job-array tasks numbers are used to calculate the second variable. (As shown with the earlier `loadMyX()` function.)

All results were received within half an hour. Individual results for one combination of variables could take as little as three minutes to calculate, and up to ten minutes. With parallelism, one job taking longer to complete did not cause other results to be delayed.

Mark's job is also remarkable in that it demonstrated "emergent behaviour".

Halve the time, double the jobs



With Mark able to get results back in half an hour instead of some future date he changed his behaviour.

This graph isn't using the usual layout, sorry. Green boxes indicate processes, wallclock is vertical, and the number of the job is horizontal. The original week and a half long job is not shown on this graph as on this scale it would be on the 27th floor. This graph shows 1,500 different processes.

The first batch is 400 jobs, taking about three minutes each. These are MC-MC simulations, so some of them take a bit longer than others. This could also be due to the speed of the random number generator on the host it is running on.

Then, there's another notch up, and this one contains over 1000 jobs, at twice the run time - now taking just over three hours a job.

If you find something on sale at half price, don't you buy twice as many?

(continues)

(continued)

Having discovered a way to run simulations much faster, the researcher increased the accuracy, increased the precision, increased the range under investigation, and ran it again. This graph gets even more spectacular as once these results were calculated, again the precision was turned up and resubmitted.

This is the sort of emergent behaviour that HPC makes possible. New types of analysis are available to us.

How times have changed

HIGH PERFORMANCE COMPUTING CENTER (HPCC)

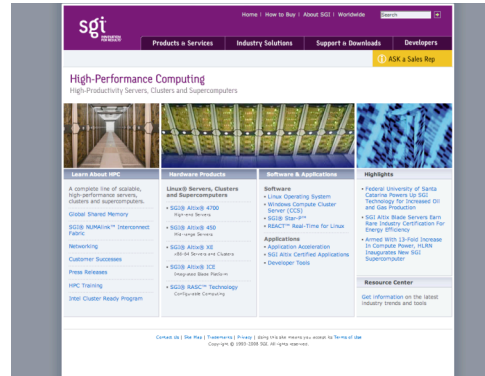


The High Performance Computing Center evolved from the former Physics Detector and Simulation Facility, later the Parallel Distributed Systems Facility (PDSF).

The workstations in the ranch (farm) normally run in coarse-grain parallelism and are available for batch and interactive use. Batch queuing is provided using Network Queueing System (NQS). Applications that require fine-grain parallelism can run under Parallel Virtual Machine (PVM). The large amount of disk space allows use of advanced distributed databases and information retrieval systems. Due to its modular architecture, the system is easily upgradable and expandable. It can be used as a test bed for emerging hardware and software technologies and new interoperability concepts.

The Parallel Distributed Systems Facility (PDSF) provides 12,000 MIPS computing power on a UNIX workstation ranch (farm). A total of 124 processors are organized in groups that are connected by Fiber Distributed Data Interface (FDDI) and Ethernet. Several types of stations are used, including HP 9000/735, Sun SPARC10, and SGI Challenge L and SGI 4D/360. Each workstation has at least 64 MB of RAM and 1 GB of disk storage. Three SGI Challenge L file servers provide an additional 160 GB of disk space. Two Summus STL-2300-12 tape robots are used for archiving.

The Intel iPSC/860 Hypercube provides 2112 MIPS of computing power, using 64 tightly coupled nodes, each with 8 MB of RAM. Total storage space is 5 GB. The Hypercube can be used for massively parallel scientific and technical applications in image processing, ray tracing, climatology, and hydrodynamics.



The “World Wide Web” (WWW) has demonstrated this sort of behaviour. Here’s a screenshot of a website from 1995. The High Performance Computing Center was to be part of a particle physics experiment. Its 124 processors had about 64MB of RAM each.

And look here, this is a modern day website about High Performance Computing, from the SGI website. SGI are very good with shared memory and their hardware allows up to 24 terabytes to be shared around a cluster.

Thanks to increases in bandwidth, colours and pretty backgrounds are common on the web. With increases in CPU speed, you can get much the same advantage.

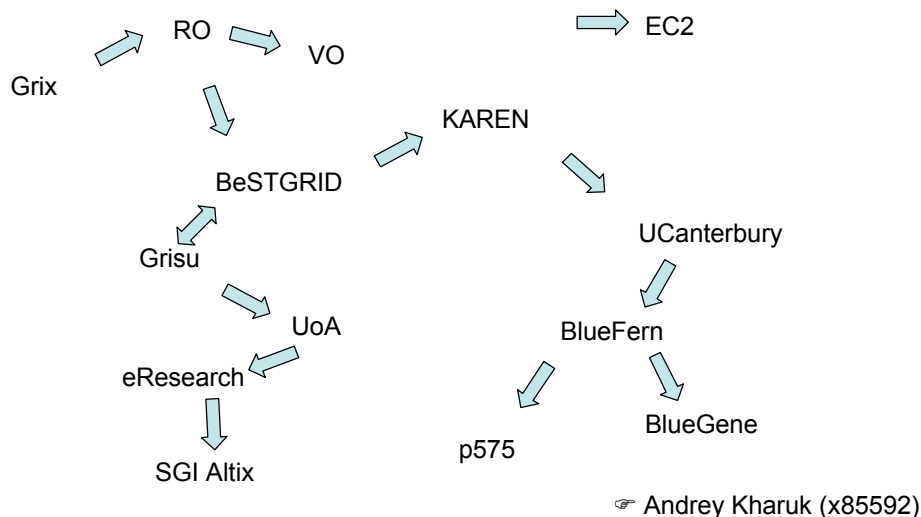
More capacity

- Within our Department we have 72 cpus
 - Run `qsub -help` on `stat1`
 - Ask me
- We have access to BeSTGRID
 - Auckland's Computation Grid Server
 - Canterbury's BlueFern Supercomputer

Our Department's resources are not limitless. We have 72 CPUs, all ready to run your jobs. Have a look at the output from `qsub -help` on `stat1` for rather cryptic advice on running your jobs. Best yet, ask me and as the "Catalyst" I'll help you get your jobs running.

As part of the University of Auckland we have access to BeSTGRID. BeSTGRID includes the Auckland computation grid server in the Centre for eResearch. BeSTGRID also includes up to 10% of the University of Canterbury's BlueFern Supercomputer. Through various financial arrangements you can get access to more of BlueFern, or the resources at Massey and Otago.

Increasing your jargon^W capacity



Run Grix to generate a certificate

Meet the RO to validate your identity

Join the /APRS/BeSTGRID virtual organisation (VO)

Our University is connected to Broadband Enabled Science Technology Grid (BeSTGRID) via Kiwi Advanced Research Network (KAREN)

Run Grisus and then you can use your certificate to submit a job to the University of Auckland cluster, handled by the Centre for eResearch, who have a SGI Altix running in the Owen G Glenn Data Center.

You can also submit jobs to BlueFern which is at the University of Canterbury. This supercomputer is one of the Top500 in the world and is a combination of an IBM BlueGene and p575.

Alternatively you can just shove your job on a virtual machine with Amazon's EC2, which you pay by the CPU-hour.

This is rather complicated and I personally have only traversed as far as BlueGene.

For the University of Auckland computational cluster the best person to talk to is Andrey Kharuk, who will be able to help. He's also our local RO for BeSTGRID, so he'll get you all sorted out with a certificate and added to all the correct groups.

Questions

You can also stop by my office

Thanks: Ivan Kojadinovic, Mark Holmes

That's what I have slides for for the second half of this talk. I'm sure you have questions. The concept of rewriting your code so that multiple copies can be run in parallel is not limited simply to R. I used R because it is familiar to some of you and also I had examples of those languages on hand. It can be used in other languages.

If your questions are too in depth we can talk about this afterwards, and you can even stop by my office (303.205).

Windows programs?

- Q: Is it possible to run Windows programs, eg, WinBUGS, on the Grid?
- A: All our compute nodes are running Linux. Perhaps with Wine you could? Maybe a VM, but then it would be better to leap straight to Amazon EC2.
- A: WinBUGS is an updated version of BUGS. There is also a Java port which will run on Linux.

Who pays for this?

- Q: What's the budget? As an UoA researcher, who pays for access to it?
- A: UoA is part of BeSTGRID, so as an UoA researcher you get access to it. If you need far more resources you'll need to pay for it.
- Q: What if I'm not a UoA researcher?
- A: We may have some spare CPU time.
- Q: How expensive is Amazon EC2?
- A: As low as 10 US cents per CPU-hour.