Foss4G Norge

Let's Get Parallel!

Graeme Bell

NIBIO
NORWEGIAN INSTITUTE OF
BIOECONOMY RESEARCH

Introduction

- Hi, nice to meet you all.
- I’m a GIS & database engineer at the Norwegian Institute of Bioeconomy Research (NIBIO). We work with large amounts of national land data (farms, forests, terrain).
- My hobby is making GIS code and database code run very fast on cheap computing equipment.

First of all

- Can you raise your hand if you’d like your geometry GIS code or raster code to run… 4-16 times faster?
- Or more…?

Focus of this talk

- Ordinary PCs: get better performance from your desktop machine or your team's GIS database server.
- Ordinary FOSS GIS tools (GDAL, postgres/postgis)
- "Easy" parallel processing opportunities.
- I will NOT talk about:

  GPU-computing, HPC clusters, super-expensive equipment, cloud computing, parallel algorithms, openCL/CUDA…

Motivation: why care?

- Modern CPUs have several cores that can run independently to give higher system performance.
- However <u>many FOSS GIS programs use a single-threaded design</u>: only 1 CPU core is used per task. e.g. Postgres/postgis, most of GDAL.
- Unfortunately, <u>single-threaded CPU performance hasn't improved much in the last 10 years</u>. It may not improve much in the future either.

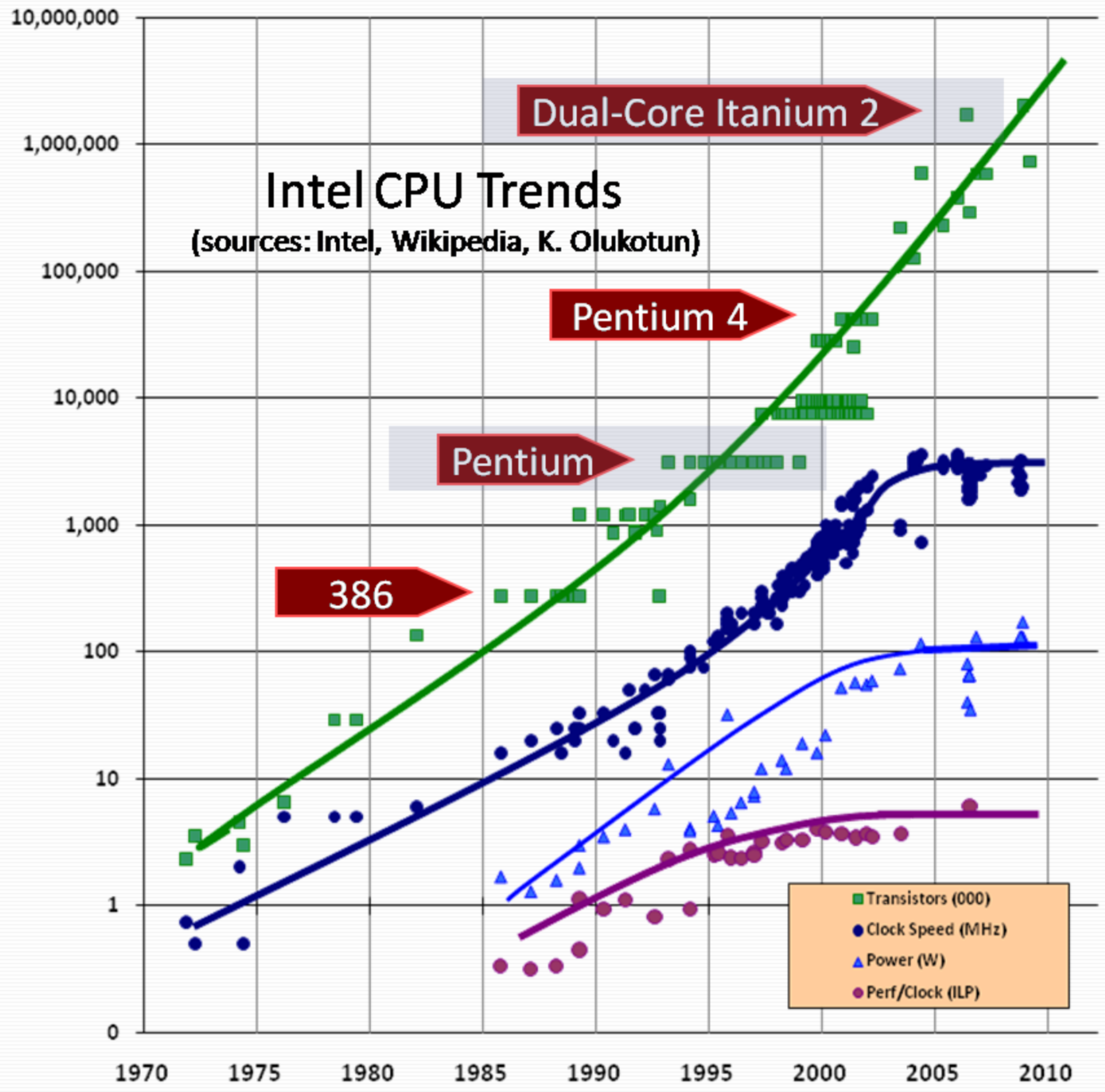
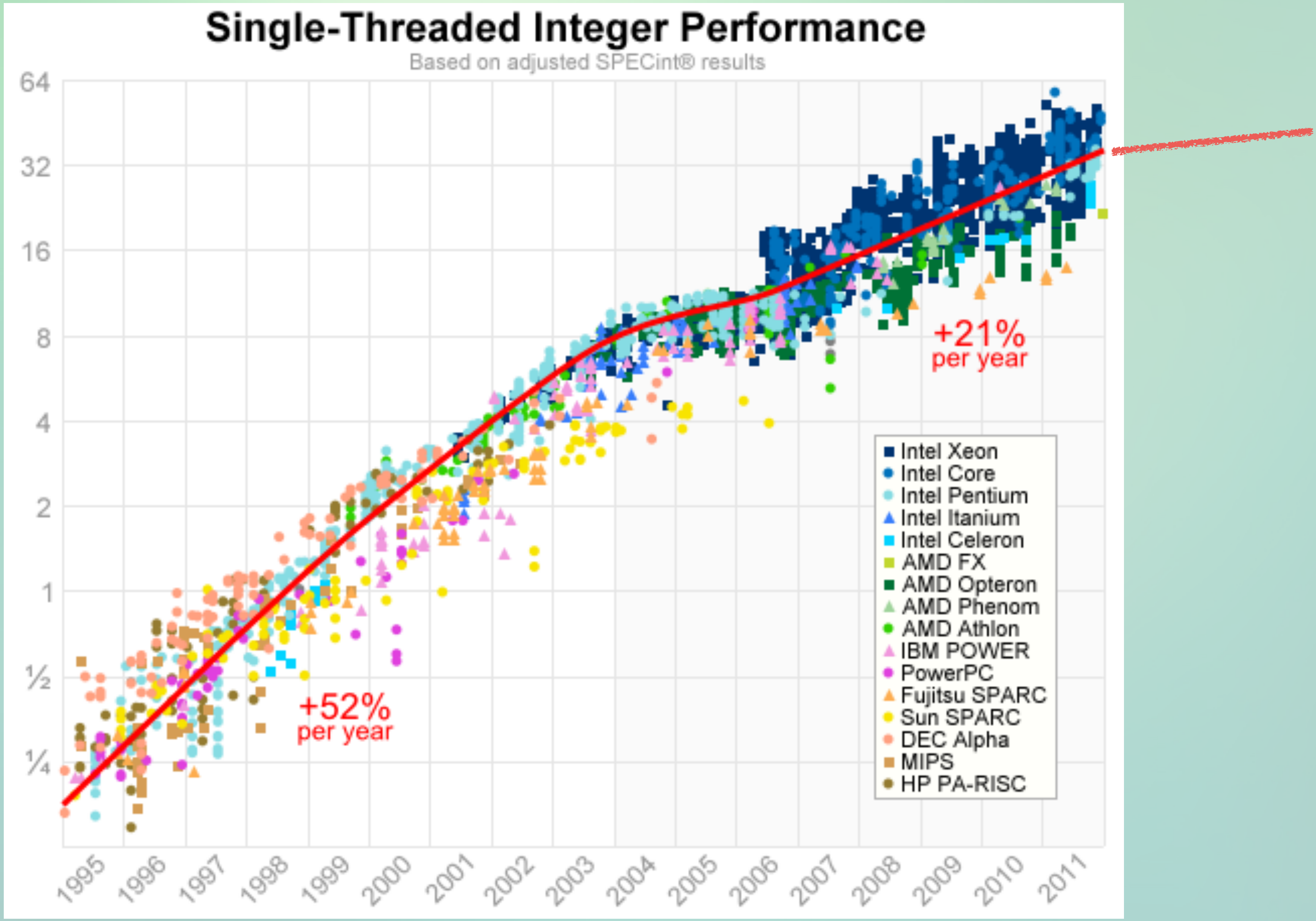
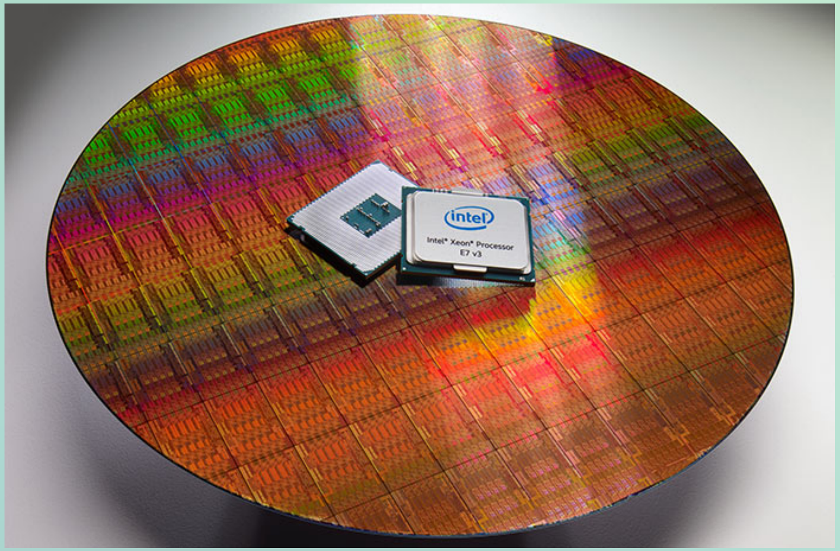


Figure 1: Intel CPU Introductions (graph updated August 2009; article text original from December 2004)



Single-core performance from 1995 to 2012

A modern CPU



- Your GIS server might have 4 x 18-core intel CPUs nowadays. That’s up to 72 cores!

- Your desktop PC probably has 4 physical cores.

- So if you only use one, that’s a lot of wasted potential.

Structure of the talk

- How to find opportunities for parallelism in GIS.
- Problems for parallelism: things to check.
- Practical tools & tips for parallelism in GIS.

The basic idea: independence

- Since we can't get CPU cores to go faster, we want to find ways to split up our GIS work so it can be done simultaneously on different cores, for a better total speed.

- The easiest way to do this is to look for parts of the GIS work (data, tasks) which are independent, and do not overlap with each other as they run.

Examples of independence

- Completely separate subproblems (e.g. “generate forest data”, “generate lakes data”, “generate roads data”).
- Spatial independence (calculate small parts of the map separately and combine them later)

Examples of independence

- Temporal independence (e.g. perhaps you have different versions of a map over time that you can work on separately)

- Simulations (e.g. use the same source dataset, but processed independently with different simulation models, or different simulation parameters)

Examples of independence

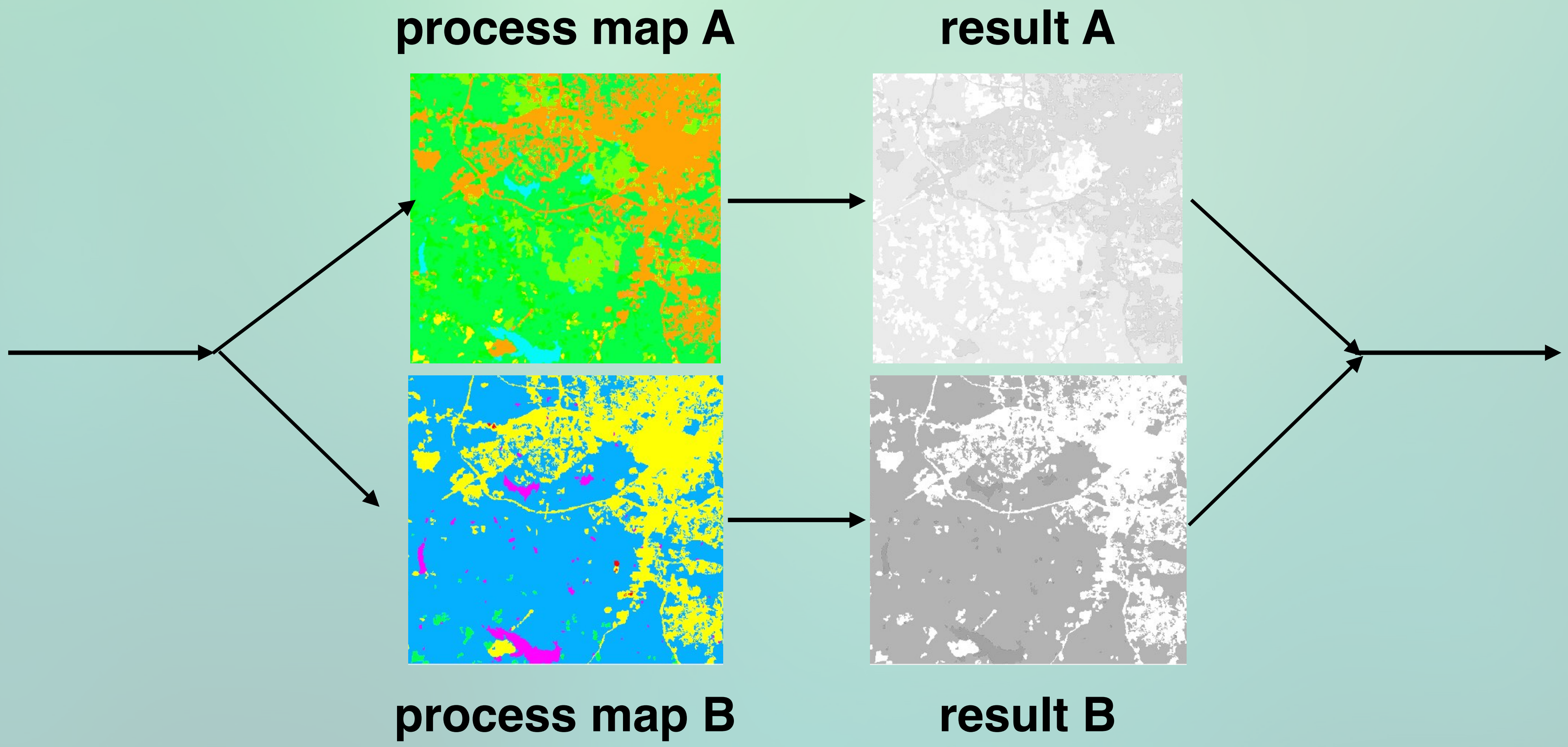
- Record-level independence. Imagine you have a farm map dataset. Every farm has a separate polygon in your database, and they are independent.

  This is a common situation in GIS and a perfect situation for parallel programming!

TIP 1: LOOK FOR INDEPENDENCE IN YOUR DATA SOURCES AND IN THE WAYS YOU USE THEM.

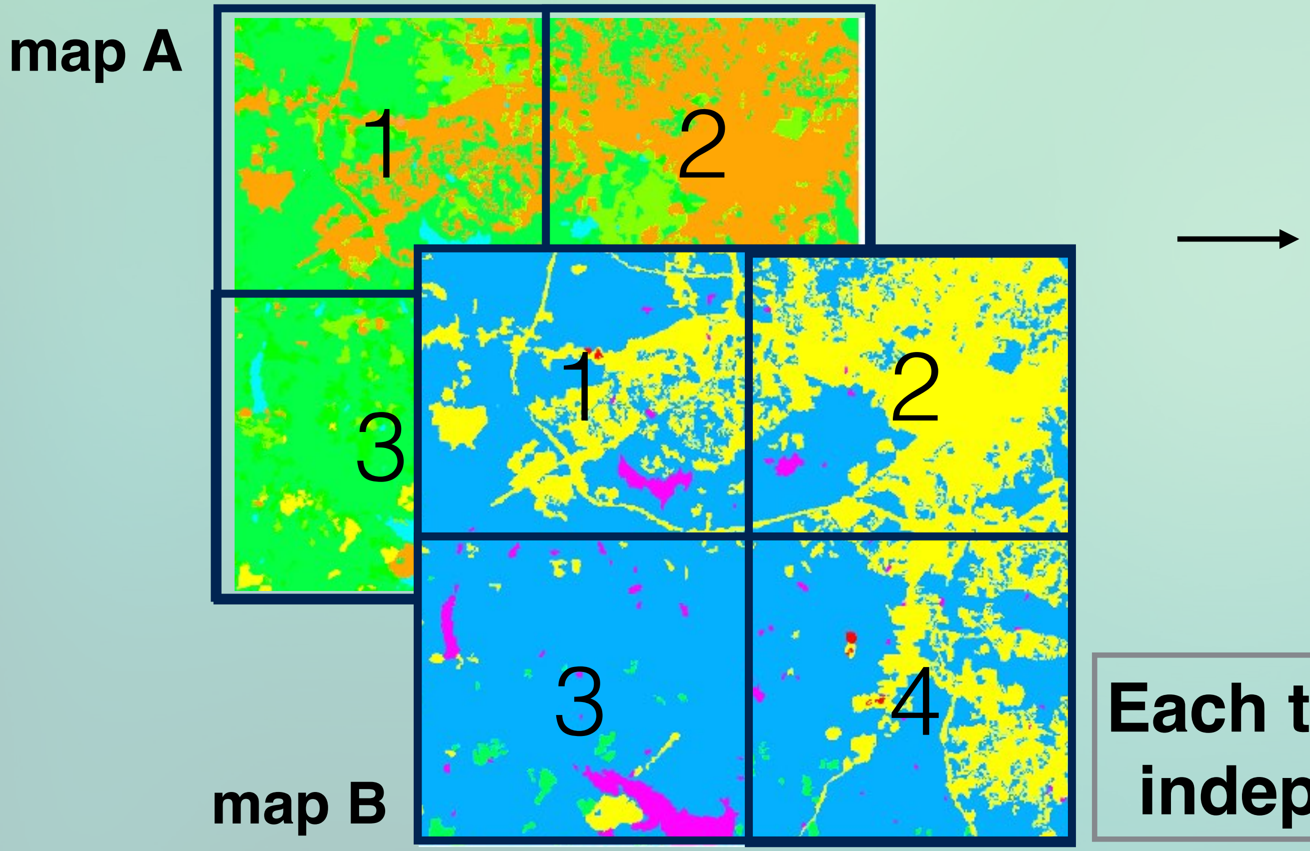
Practical example 1

- Geometry processing. For example, prepare two different source maps you will need, in parallel.



Practical example 2

- Raster processing. Split your map into a grid of tiles. Process tiles (or groups of tiles) in parallel.



combine A1 + B1

combine A2 + B2

combine A3 + B3

combine A4 + B4

Each tile (or stack of tiles) is independent of the others.

Part 2:
Problems for parallelism

- So, we know what we want to parallelise, but what might go wrong along the way?

- The big problem is ‘competition for shared resources’. In particular:

  Memory, Disk, Network, Write-access, and Heat.

- Problem 2 is the configuration of your machine.

- Problem 3 is the availability of easy tools.

Memory

- If you run 10 tasks on 10 CPU cores, you may need 10x as much RAM.

- The impact when you run out of memory can be huge!

- Accessing a record in memory
  ~ 100 nanoseconds.
  Accessing a record on disk (or virtual memory)
  ~ 10000000 nanoseconds.

- If you're forced to go to disk, it's up to 10000x slower!

TIP 2: OBSERVE MEMORY USE - MAYBE BUY MORE.

Input/Output (disk, network)

- A hard disk drive can only do 100 to 300 operations per second (IOPS). Compare that against the 3,000,000,000 operations per second each CPU core can achieve (and we have many cores).
- Data input can sometimes be made faster with extra memory (cache), but data output always needs a fast disk system. When you are creating or updating a map, this is a huge problem.
- The problem gets much worse with many parallel tasks..
- A cheap modern SSD flash drive can achieve 30000-300000 IOPS. That’s 100x-3000x better than a hard drive!

TIP 3: WRITING DATA IN PARALLEL? GET AN SSD. HARD DISKS CAN HARDLY KEEP UP WITH 1 CORE.

Locks (write-access)

- Let’s say your program wants to change a table.
- If you run several copies of your program, they all want to change the same table at the same time.
- To prevent the changes from overlapping, the database or operating system may refuse to let multiple writes take place on the same data at the same time (‘LOCKING’).
- So, try to avoid writing directly to a result table - use several temporary output tables.

TIP 4: WRITING DATA IN PARALLEL? OUTPUT TO DIFFERENT FILES/TABLES AND JOIN THEM LATER.

Heat

- More cores on a chip = more heat
  Faster cores on a chip = more heat

- Processors with the maximum number of cores set each core to be slower, to keep within their heat limit.

- A processor that is good for 18-way tasks will be worse at 1-way tasks (things that can’t be parallelised).

TIP 5: AN 18-CORE INTEL CPU RUNS AT 2.5Ghz, WHEREAS THE 8-CORE RUNS AT 3.2Ghz. THINK ABOUT YOUR TASKS AND CHOOSE CAREFULLY!

Configuration

- The setup of your computer/server can have a big impact on your ability to parallelise too.
- However, not everyone is a server techie, so I’ve put some suggestions for server admins at the very end of this talk.
- I’m going to skip this topic for now but note that it’s also quite important.

Tools

- Many/most FOSS GIS tools are single threaded.
- That means if you want parallel performance, you have to figure out how to do it yourself.
- So, I’m now going to show you some tools I’ve made that make this much easier.

par_psql

- par_psql is a drop-in replacement for the ‘psql’ postgres command. It lets you massively speed up your work by allowing parallel queries from a single GIS script.

- You can now run parallel queries from a single SQL script with a simple notation: - - &

- Any line with ‘- - &’ at the end will run in parallel.

- ‘- - &’ is simply a comment if viewed by other SQL programs

Synchronisation

- However, starting things in parallel is half the story.
- How can you plan the rest of your script if you don’t know when your parallel tasks are finished? Maybe you are depending on these tables later.
- par_psql automatically synchronises your parallel work - waits for everything to complete - whenever it encounters a non-parallel (serial) command.
- You can mix ‘serial’ and ‘parallel’ in one script.

Compatibility

- psql scripts will run immediately in par_psql
- par_psql scripts will run immediately run in psql
- par_psql takes all the same options that psql takes
- A few things aren’t allowed (e.g. temporary tables), but workarounds are easy (e.g. unlogged tables)
- OK, now an example…

psql

- select sum(st_area(geom)) from forestmap2012;
- select sum(st_area(geom)) from forestmap2013;
- select sum(st_area(geom)) from forestmap2014;
- select ‘Finished!’;

- psql   -h gis -U bob forestdb —file=areas.sql

par_psql

- select sum(st_area(geom)) from forestmap2012; - - &
- select sum(st_area(geom)) from forestmap2013; - - &
- select sum(st_area(geom)) from forestmap2014; - - &
- select ‘Finished!’;

- par_psql -h gis -U bob forestdb —file=areas.sql

par_psql

- You can also group commands for parallelisation using pl/pgsql functions or e.g. this idiom:
- CREATE TABLE a …. ; CREATE INDEX b….; - - &
- Easy to install
  - http://github.com/gbb/par_psql
  - git clone   (or download zip file)
    ./install.sh

Examples (4-core desktop)

Test	psql	par_psql	improvement
Select statements 1 million rows	185s	52s	3.6x
pl-pgsql static function, 200000 calls	62s	16s	3.9x
Overhead test	0.012s	0.22s	-

What kind of things can you parallelise?

- Pretty much anything:

- Reprojecting datasets
  Transforming datasets
  Analysis on datasets
  Simulations based on datasets

- You can make parallel “previews” using smaller parts of your map while the main map is being transformed without slowing it down.

- Also available at http://parpsql.com (redirects to github)

A useful trick for parallel SQL

- Sometimes you want to split up a map so that you can run queries or functions on smaller parts of the map more quickly.

- One option is to use bounding boxes to split up a map, but this trick - modulo division - will split up any table (GIS or non-GIS). For example, here, odd and even GIDs go to different CPU cores. You can also do e.g. 16-way if you like.

- select function() from table where gid%2=0; - -& 50% of map
  select function() from table where gid%2=1; - -& 50% of map

- By using gid to split up tables this way, you don't need to know anything about the maps you work on (e.g. extents, SRID).

fast_map_intersection

- GIS people often need map intersections.
  Intersections are often slow (hours or days) with national maps.

- Fast_map_intersection lets you write your intersection using some pieces of normal SQL.

- It then rewrites your SQL so it can be run as many smaller pieces in parallel. It runs all the pieces in parallel for you, and then puts everything back into one map again automatically, so the result is identical to the original query.

- This can be e.g. 8 times faster than an ordinary intersection.

- You can find it at http://github.com/gbb/

rbuild

- (Presented at foss4g Nottingham: http://graemebell.net/foss4g2013.pdf)
- rbuild is a tool that lets you perform very fast parallel raster calculations on geometry / raster maps.
- It will automatically tile your maps, quickly perform raster calculations, and rebuild the map back into geometry again.
- The main goal is to offer parallel raster processing while also optimising the configuration of GDAL & Postgis to be super-fast.
- It also lets you design complex GIS transformations using Excel spreadsheets, and converts them to high-speed raster algebra.

make/drake/bpipe

- Make, drake and bpipe are tools that can be used to manage complicated data workflows.
- You tell them what you want to build, how it should be built, and what order to build things. Then they automatically run your SQL/BASH scripts for you, using parallelism whenever they can.
- This can be useful if you have source data that changes often, and a project that must be rebuilt often when it happens.
- Tools like make can be challenging for GIS users to set up if you are running SQL rather than compiling C++. But if you have a huge GIS workflow with hundreds of tasks, it may be worth it.

Gnu parallel

- Gnu Parallel can be used as a framework to build new parallel GIS programs. It allows you to quickly set up groups of commands to run in parallel (e.g. Python, BASH, …)
- It has many nice/cute features - e.g. progress report across all your cores, automatically detecting CPUs etc.
- It is able to coordinate work across several machines via ssh.
- The tutorial is excellent and gives a good idea of the possibilities. http://www.gnu.org/software/parallel/parallel_tutorial.html
- Rbuild uses Gnu Parallel to control the parallel tasks.

Summary of tools

- Rasters: rbuild http://github.com/gbb/rbuild
  GDAL (read the manual! some tools have parallel processor options)

- PostGIS: par_psql http://github.com/gbb/par_psql
  fast_intersect http://github.com/gbb/…

- Workflow: make (built into most Linux distributions)
  drake https://github.com/Factual/drake
  GNU Parallel www.gnu.org/software/parallel/

- These are all free open source software.

- If you use github, why not login and star* some of them now?

Thanks for listening!

- If you'd like to download a copy of these slides to look at later, here's a quick and easy link: http://graemebell.net/foss4gcomo.pdf
- Here's my github account: http://github.com/gbb
- Let me know if you try par_psql, rbuild or fast_intersect, I'm hoping for feedback.
- There are 3 extra 'bonus' slides in the PDF after this slide with 'techie tips' if you're setting up a new machine and want to get great parallel performance.

For Techies: Hardware

- Disable hyper-threading in the BIOS if your server is only for postgres. Many people report better multi-core performance with HT off.
- However, in my own experience, HT is helpful to performance if your server is used for a mix of e.g. postgres and e.g. R/GDAL work.
- If you use a RAID array, RAID10 will be a lot faster than RAID5/6 when you have many CPU cores reading or writing data to disk.
- Make sure you have good cooling. Modern processors will automatically run a bit faster if they’re cool.
- Disable NUMA (non-uniform memory access) in the BIOS. Many people report problems with NUMA, postgres and multi-core servers.

For Techies: Linux

- Make sure you’re using the latest Linux kernel. We saw a doubling of postgres' small transaction performance, going from RHEL6 kernel to a recent mainline kernel.
- If you have an SSD, set the disk scheduler to ‘deadline’. This is fast & fair to many running tasks.
- Set the disk read-ahead to e.g. 4MB, you should find this greatly improves results with large tables.
- Also consider raising the IO queue size (‘nr_requests') .

For Techies: Postgres

- Increase effective_io_concurrency in the conf. file if you have SSDs.
- The WAL log can be a choke point for parallel performance. Try using unlogged tables if you're writing lots in in parallel.
- If you're using pl/pgsql, use the 'STATIC' volatility marker wherever you can, it's *extremely important* for scaling.
- Reduce the 'work_mem' setting if you're running many tasks at once.
- If you get an SSD, set 'random_page_cost' carefully in the conf. file.
- Consider running pgbouncer to limit the problem of 'too much parallelism', which can make the system grind to a halt.

Summary of top tips

TIP 1: LOOK FOR INDEPENDENCE IN YOUR DATA SOURCES AND HOW YOU WORK WITH THEM.

TIP 2: MONITOR MEMORY USE - MAYBE BUY MORE.

TIP 3: WRITING DATA IN PARALLEL? GET AN SSD.

TIP 4: WRITING DATA IN PARALLEL? OUTPUT TO DIFFERENT FILES/TABLES AND JOIN THEM LATER.

TIP 5: AN 18-CORE INTEL CPU RUNS AT 2.5Ghz, WHEREAS THE 8-CORE RUNS AT 3.2Ghz. THINK ABOUT YOUR TASKS AND CHOOSE CAREFULLY!

References

- Graph of Mhz vs time: http://www.gotw.ca/publications/concurrency-ddj.htm
- Graph of single CPU performance vs time: http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/
- Image of intel processor: http://hexus.net/media/uploaded/2015/5/2774036d-8b27-4eae-b052-25c852167446.jpg, likely copyright of Intel.