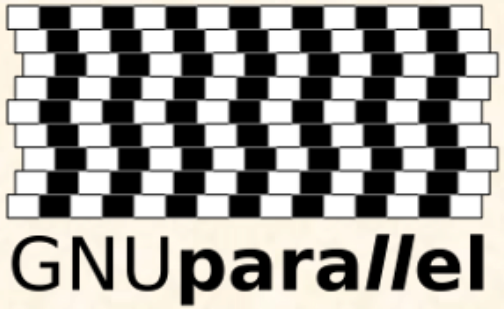
# GNU Parallel Reimplementation in C

**Parallel Processing UNIX Tool, Across Multiple Computers on Network**

**By Thai Flowers and Arthur Karapateas**



# Motivation for Implementing this Tool

- Originally, we wanted to implement a new parallel processing UNIX tool that could work across multiple computers on the same network. This tool would provide a simple method to split up work across multiple computers on the same network.
- GNU Parallel originally released in 2007 written in PERL
- PERL is a scripting language that runs inefficiently with memory and runtime, older language, language is not as supported nowadays
- Our implementation will be more efficient on memory/runtime and will be brought to a language(C) with more feature support for future proofing this tool.

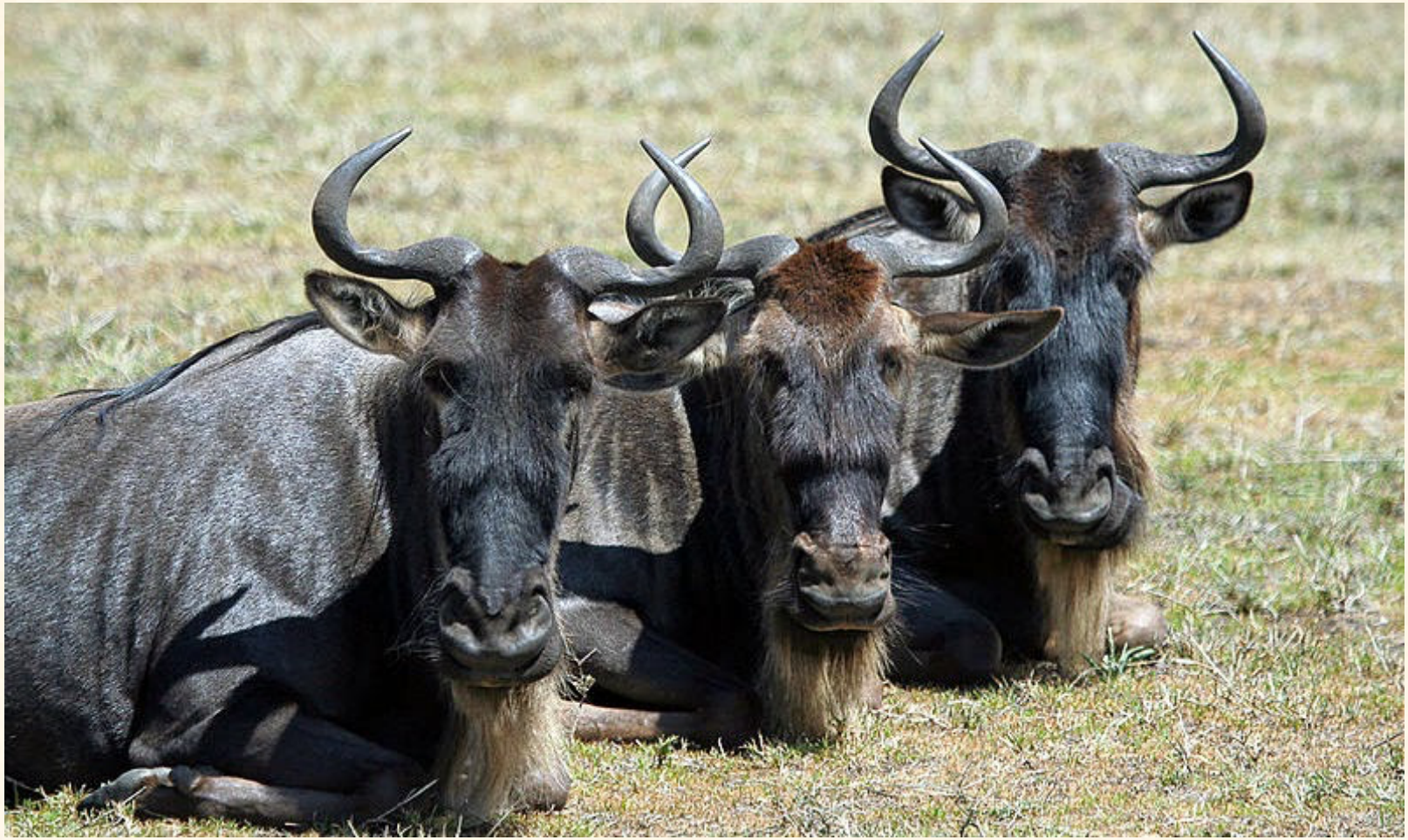
# Project Implementation Tasks

- Implement a bare bones, basic version of a GNU Parallel like program in C. This is currently done.

- Add more features, use regex.h to add more full featured substitutions, use “sh wrapping”. Currently finishing this task.

- Configure passwordless login and use “ssh” wrapping to enable networking to split up work across multiple computers on the same network.

# What is GNU Parallel?



# What is GNU Parallel?

- One of the GNU Projects Official Tools to run shell commands in parallel.
- Replacement for xargs and for loops with parallelization.
- Process based parallelization
- Became an official GNU project on 04-22-2010
- Created in 2007 by Ole Tange

# What is GNU Parallel: Continued

- Very feature rich tool
  - Works on files, command line, standard input, and input sources
  - Allows for remote execution
  - Suite of simple substitution
  - Forces sequential output
  - Perl regex based substitution when simple substitution fails
  - Splits large files into separate chuncks that are processed in parallel
  - Can even do SQL database access

# Original GNU Parallel vs Our Implementation

- Huge number of features
- Complex command line parsing and internals
- Implemented in PERL script of 11,394 lines in main program
- Uses outdated PERL, and implemented in 1 file to aid installing in users personal directories
- Heavy bash dependency/assumption

- Extremely basic feature set.
- Simple command line parsing and internals
- So far we have 451 lines of C code
- Loose sh dependency, once we implement sh wrapping. GNU assumes you are always using bash, and we want this tool to also work in tcshell.
- Mostly standards conformant C.

# Replacement String Support for GNU Parallel

| Replacement string | Value if input is mydir/mysubdir/myfile.myext |
|---|---|
| {} | mydir/mysubdir/myfile.myext |
| {.} | mydir/mysubdir/myfile |
| {/}, {//}, {/.} | myfile.myext, mydir/mysubdir, myfile |
| {#} | *The sequence number of the job* |
| {%} | *The job slot number* |
| {2} | *Value from the second input source* |
| {2.} {2/} {2//} {2/.} | *Combination of* {2} *and* {.} {/} {//} {/.} |
| {= perl expression =} | *Change* $_ *with perl expression* |

- **Our tool supports all basic replacement strings in GNU Parallel except the final {= perl expression =}**
  - **This is easy in normal GNU Parallel because GNU Parallel is written in PERL, this would be very challenging to do in our C implementation.**

# Syntax our Tool Supports

- Parallel --dryrun -j<#jobs> command ::: a b c d
  - a b c d are the command line sources
- -j<#jobs> or -- jobs<#jobs>
- Will support  :::: file1 file2 … Filen (4 colons)
  - ::: and  :::: may be interleaved in GNU parallel, our implementation might be stricter in final version
- Maximum of 8 command line sources
- --dryrun just outputs the parsed/substituted command line

# Our Process to get Networking Version Working

- 1) Basic “raw” implementation
  - args = [“echo”, “a”, NULL]
  - execvp(“echo”, args);
- 2) “sh wrapping” (currently here in our implementation)
  - Allows bash/sh function calls, pipes etc
  - args = [“sh”, “-c”, “echo”, “a”, NULL]
  - execvp(“sh”, args”);

# Our Process to get Networking Version Working

- 3) Config passwordless ssh login for desired user on all remote systems
  - Relatively simple process, but we have never done it before
  - Involves generating public/private key pairs
- 4) “ssh wrapping”
  - Modify parallel to accept –S server-name option
  - Ex command
    - Parallel –S server1 –S server2
      - “hostname; echo {} ::: foo bar
  - args = [“ssh, “user@server”, “echo”, “a”, NULL]
  - Problem of determining # of processes per connection

# Examples of Usage

- Compress all html files, 2 jobs per “thread”
  - parallel --jobs 200% gzip ::: *.html
- Convert wav files to mp3, 1 job per cpu(default)
  - parallel lame {} –o {.}.mp3 ::: *.wav
- parallel --dryrun echo {2} {1} ::: bird flower fish ::: R G B
  - {2} {1} must be quoted on csh/tcsh

# Additional Research Takeaways

- GNU Parallel could have been introduced in 1987 not 2007
- Multics(1969) was multiprocessor system for main frames
  - Multiplexed Information and Computing Service)
- Non-commercial MUNIX from Naval Postgrad School 1975 thesis
  - “Multiprocessor UNIX Operating Systems” Bach and Buroff 1983 report listed 1975 thesis for similar parallel tool
- Hardware, C multi-process libraries, and network capabilities were all available by 1987