Teaching an old dinosaur new tricks

FAIR principles and the GAMESS quantum chemistry package

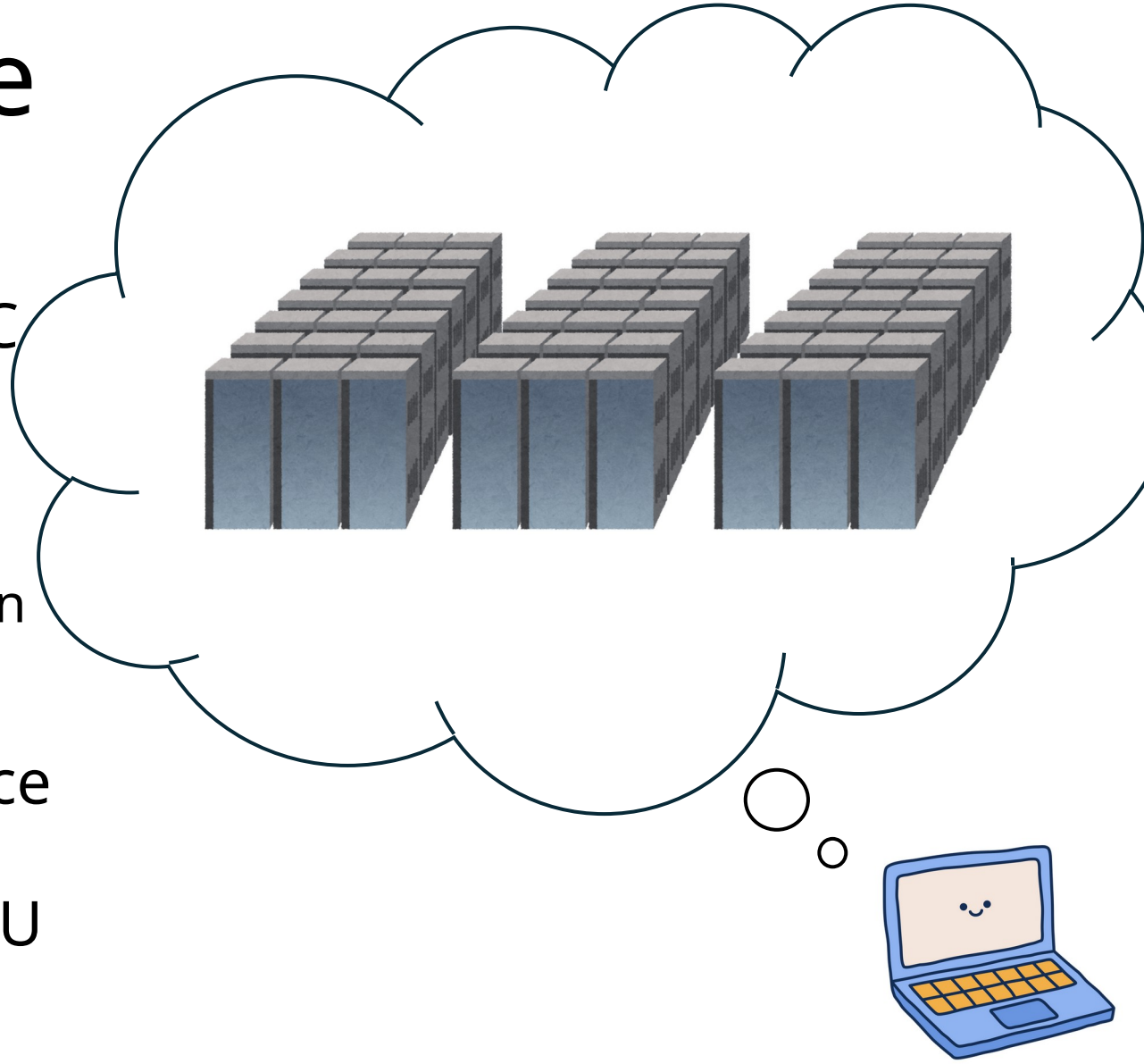
Jorge Luis Galvez Vallejo
National Computational Infrastructure

History of GAMESS

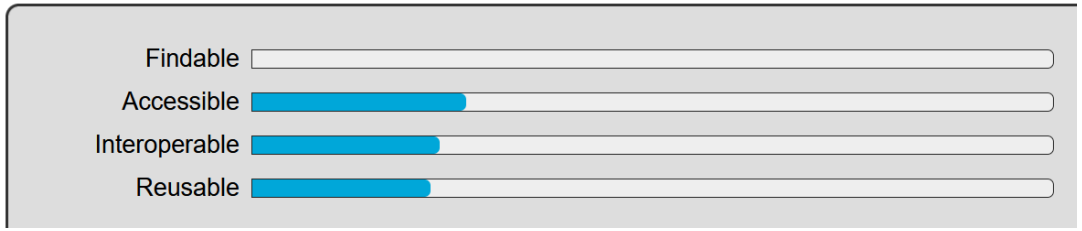
- Successor of “HONDO” written my Michel Dupuis in the 1980
- Development moved to North Dakota state (1982-1982)
- Moved to Iowa State University (1992 – present)
- Early adopters of parallel computing (1991, Windus et al)
- Early MPI wrappers for distributed parallelism via the “distributed data interface” (1999, Olson et al)
- Early adopters of distributed compute for massively parallel architectures
- Funded by the Exascale Computing Project (2017-2023)

Features of the code

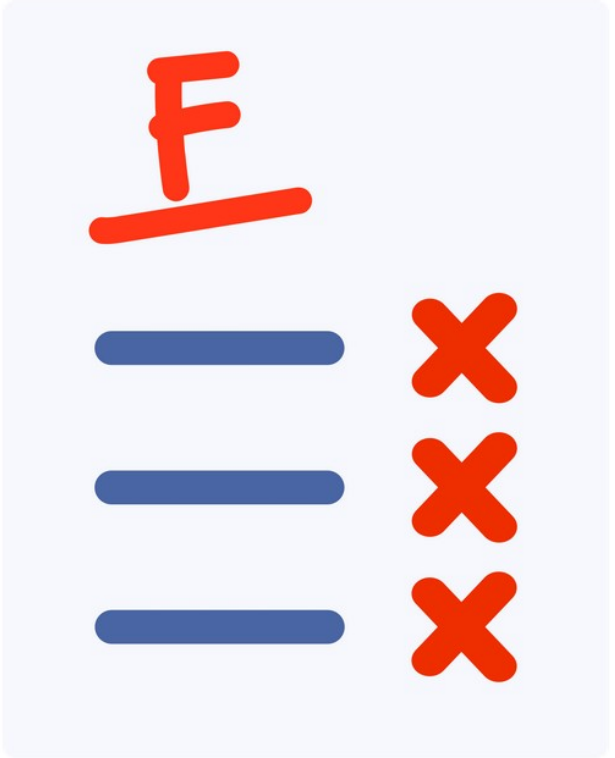
- Written in Fortran (multiple dialects) with more than 2M LOC
  - Emphasis in portability before it became cool
    - Current “portability” of GAMESS is lacking
  - External C++/CUDA development in 2010-2012 (deprecated and slated for replacement)
- DDI written in C89-98 as interface to MPI
- OpenMP as CPU parallelism (GPU parallelism since 2022)



Is GAMESS FAIR?



- Findable:
  - Available in the GAMESS website but website does not use SEO technology nor the name is unique
- Accessible:
  - One needs to sign up and wait for a confirmation link
  - Downloads expire
- Interoperable:
  - Software uses its own “open” input/output format but it is not standardized
  - Cannot read other formats
- Reusable:
  - Software can be modified but the build and execution process is not straightforward – specially to use accelerators



Barrier for FAIR GAMESS

- Licensing:
  - One needs to sign up on the website (unless developer access)
  - License does not allow for a simpler distribution avenue
- Licensing woes only impact the “Accessible” component (from the source code perspective)
- GAMESS can be made “FIR”

Why GAMESS?

- ~~Stockholm Syndrome~~
- GAMESS is a long standing, widely used (20k+ users around the world)
  - Highly utilized in the global south and Latin America
  - Favoured in Japan and Korea
  - Competitive in USA, Australia, Europe, etc.
- GAMESS supports multiple methods that are not available in other programs

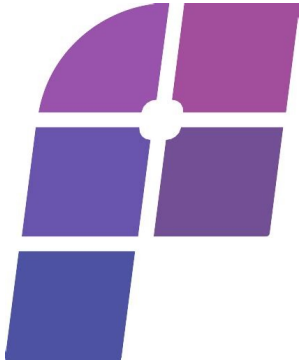
Interoperable and Reusable

- GAMESS relies on old architectur
  - Build scripts, hard coded cases, no automation integration for creating executables
  - Entry barrier for new users
  - Barrier even for experienced users
- Make building and using simple
  - Making it extendable
- Automatize dependencies
- Package manager

CMake
Cross-platform Make

Spack

ANACONDA®



Humble beginnings

- Current GAMESS build system:
  - `config` wizard (requires manual input/output of paths and libraries)
  - `comp` single “`$FC $file.f90 –o $file.o`”
  - `compall “for each file in files, run `comp`
  - `Makefile` calls compall (lack of full build parallelism)
- Addition of an automatable CMake build system
  - Industry standard
  - Supported across Operating Systems and Hardware Architectures
  - Open source
  - Thoroughly documented

Requirements

- Works out of the box
- Coexists with old build system
- Portable across already supported architectures:
  - Windows
  - Mac (ARM & Intel)
  - ARM
  - IBM
- Support same compilers:
  - GNU
  - Intel Classic
  - Intel LLVM
  - NVIDIA HPC SDK
  - AMD (aocc)
  - AMD (flang)
  - Flang
  - Cray CCE
  - Apple Clang
- Support accelerated builds
  - NVIDIA GPUs
  - Intel GPUs
  - AMD GPUs

Consequences of modernizing the build system

- Building GAMESS is now: `cmake ../`
- Covered all major features in GAMESS
- Added automated interfacing to unit testing frameworks
- Added a conda environment to install simple dependencies (GNU, Intel compilers)
- Created a Spack recipe to deploy by building from scratch

```
-- Feature Summary:
--   DDI: ON
--   OpenMP: OFF
--   MPI: OFF
--   Sockets: ON
--   GPU support: OFF
--   OpenMP GPU Offloading: OFF
--   GMS-HPC: OFF
--   MDI: OFF
--   TINKER: OFF
--   MSUCC: OFF
--   VB2000: OFF
--   NEO: OFF
--   RSMCSED: OFF
--   LibXC: OFF
--   Magma: OFF
--   Data Servers: ON
-- Configuring done (1.4s)
-- Generating done (0.1s)
```

Deploying

- As simple as:
  - spack env activate tools/spack-env
  - cmake -B build
  - make -C build -j install
- Or:
  - conda env create -f environment.yml
  - conda activate games
  - ./tools/build_gms_conda.sh
- Or:
  - spack install gamess

Key features enabled through a better build system

- Deployment of GAMESS as a library to be consumed by other packages
  - Impossible with previous build system without extreme tinkering of the delicate build process
- Integration to standardized unit testing frameworks
  - GAMESS has no unit tests
- Simplification of automated CI procedure
- Sped up compilation times (2x speedup over previous build system thanks to parallel builds)
  - Decrease recompilation cascades from modules
- Integration of GAMESS as a cmake package

Conclusions

- So, was this just adding a new build system?
  - Yes
- So, why the big deal?
  - Imagine going from hand assembly to an assembly line
- Is this already in the publicly available GAMESS version?
  - No, due to staffing issues development at GAMESS has slowed down
- This looked nice, do you need beta testers?
  - Yes, please. Reach out to me or come talk to me after the talk
- Why CMake in particular and not meson or Make?
  - Because I’ve been doing CMake for 6 years and I’m Stockholm Syndrome’d into it
- Will you explore the Fortran package manager?
  - Yes. The goal will be to be able to do “fpm install games –profile release”
- Are you thinking on rewriting GAMESS in C/C++?
  - No. Fortran all the way.