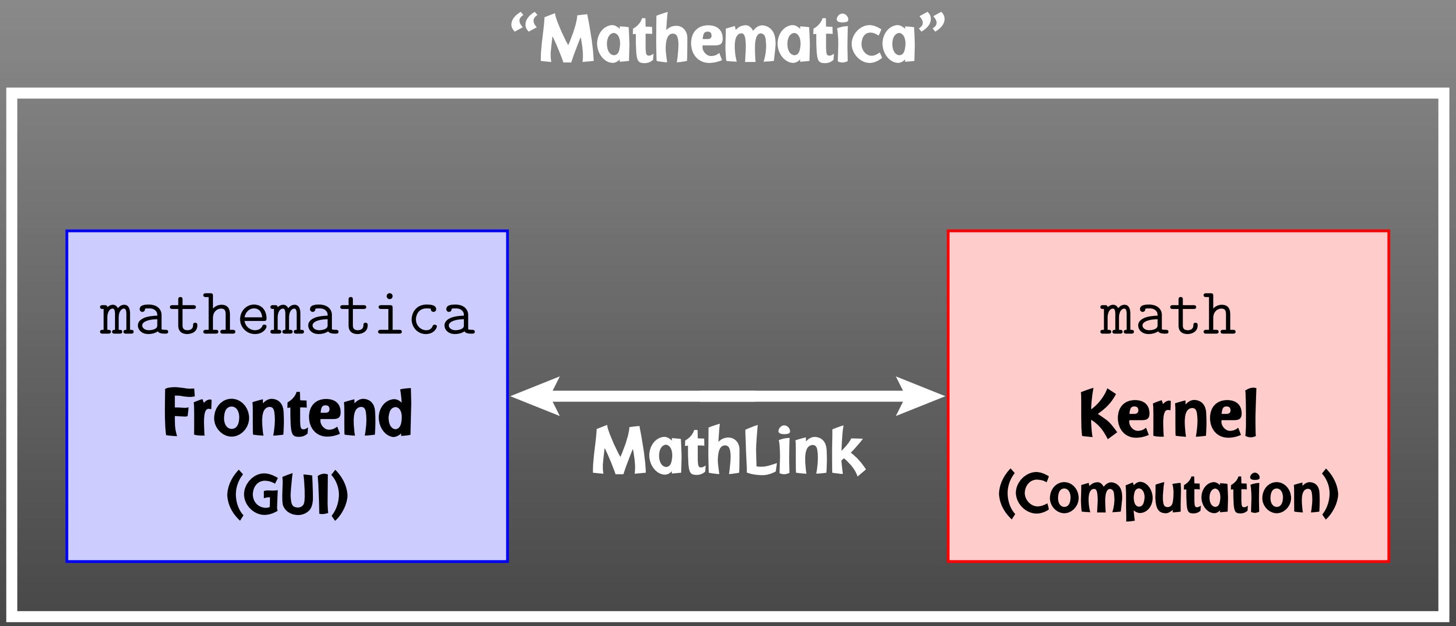
Getting most out of Mathematica

Thomas Hahn

Max-Planck-Institut für Physik
München

Mathematica Components



Why I don't like the Frontend (much)

Frontend:

- ☺ Nice formatting
- ☺ Documentation
- ☺ Ease of use
- ☹ No obvious relation between screen and definitions
- ☹ Always interactive
- ☹ Slow startup

Kernel:

- ☹ Text interface
- ☹ No pretty-printing
- ☺ 1-to-1 relation to definitions
- ☺ Interactive and non-interactive
- ☺ Scriptable
- ☺ Fast startup

Plan

- Program smart!
- Parallelize!
- Script! Distribute! Automate!
- Crunch numbers outside Mathematica!

But: don't overdo it.
If your calculation takes 5 min in total, don't waste time improving.

Program smart!

List-oriented Programming

Using Mathematica's list-oriented commands is almost always of advantage in both speed and elegance.

Consider:

```
tab = Table[Random[], {10^7}];

test1 := Block[ {sum = 0},
  Do[ sum += tab[[i]], {i, Length[tab]} ];
  sum ]

test2 := Apply[Plus, tab]
```

Here are the timings:

```
Timing[test1][[1]] ☞ 8.29 Second
Timing[test2][[1]] ☞ 1.75 Second
```

More Speed Bumps

Consider:

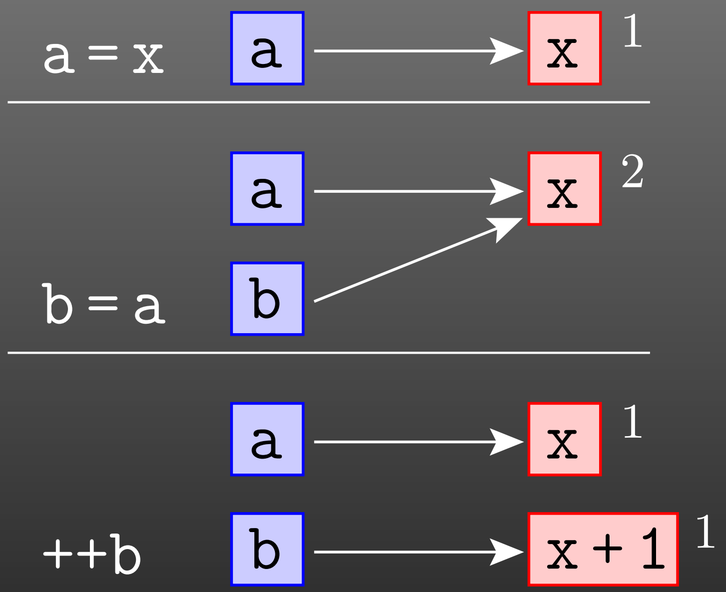
```
tab = Table[Random[], {10^5}];

test1 := Block[ {res = {}},
  Do[ AppendTo[res, tab[[i]]], {i, Length[tab]} ];
  res ]

test2 := Block[ {res = {}},
  Do[ res = {res, tab[[i]]}, {i, Length[tab]} ];
  Flatten[res] ]
```

The timings:

```
Timing[test1][[1]] ☞ 19.47 Second
Timing[test2][[1]] ☞  0.11 Second
```

Reference Count

Assignments that don't change the content make no copy but just increase the Reference Count.



Reference Count and Speed

```
test1 := ...
  ... AppendTo[res, tab[[i]]] ...
  res

test2 :=
  ... res = {res, tab[[i]]} ...
  Flatten[res]
```

`test1` **has to re-write the list every time an element is added:**

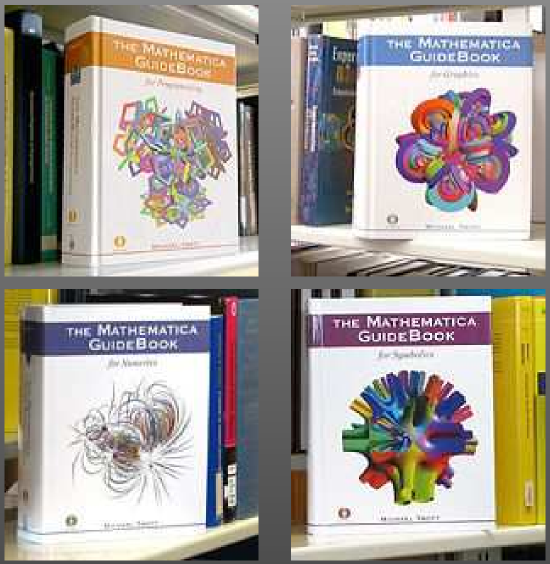
`{}    {1}    {1,2}    {1,2,3}    ...`

`test2` **does that only once at the end with** `Flatten`**:**

`{}    {{},1}    {{{},1},2}    {{{{},1},2},3} ...`

More Programming Wisdom

- Michael Trott
  The Mathematica Guidebook
  for {Programming, Graphics, Numerics, Symbolics} (4 vol)
  Springer, 2004–2006.



Parallelize!

Parallel Kernels

Mathematica has built-in support for parallel Kernels:

```
LaunchKernels[];
ParallelNeeds["mypackage`"];

data = << mydata;
ParallelMap[myfunc, data];
```

Parallel Kernels count toward Sublicenses.
 # Sublicenses = 8 × # interactive Licenses.

MPP: 35 interactive licenses (5k€ each), 288 sublicenses.

Parallel Functions

- **More functions:**

  ```
  ParallelArray  ParallelEvaluate  ParallelNeeds
  ParallelSum    ParallelCombine   ParallelTable
  ParallelDo     ParallelProduct   ParallelTry
  ParallelMap    ParallelSubmit
  DistributeDefinitions  DistributeContexts
  ```

- **Automatic parallelization (so-so success):**
  `Parallelize[`*expr*`]`

- **'Intrinsic' functions (e.g. `Simplify`) not parallelizable.**

- **Multithreaded computation partially automatic (OMP) for some numerical functions, e.g. `Eigensystem`.**

- **Take care of side-effects of functions.**

- **Usual concurrency stuff (write to same file, etc).**

Script! Distribute! Automate!

Scripting Mathematica

Efficient batch processing with Mathematica:

Put everything into a script, using sh's Here documents:

```
#! /bin/bash ............... Shell Magic
math << \_EOF_ ............ start Here document (note the \)
  << FeynArts`
  << FormCalc`
  top = CreateTopologies[...];
  ...
_EOF_ ..................... end Here document
```

Everything between "<< *tag*" and "*tag*" goes to Mathematica as if it were typed from the keyboard.

Note the "\\" before *tag*, it makes the shell pass everything literally to Mathematica, without shell substitutions.

Scripting Mathematica

- Everything contained in one compact shell script, even if it involves several Mathematica sessions.
- Can combine with arbitrary shell programming, e.g. can use command-line arguments efficiently:

```
#! /bin/sh
math -run "arg1=$1" -run "arg2=$2" ... << \END
  ...
END
```

- Can easily be run in the background, or combined with utilities such as make.

Debugging hint: `-x` flag makes shell echo every statement,

```
#! /bin/sh -x
```

Crunch numbers outside Mathematica!

Code generation

- Conversion of Mathematica expression to Fortran/C painless.
- Optimized output can easily run faster than in Mathematica.
- Showstopper: Functions not available in Fortran/C, e.g. `NDSolve`, `Zeta`. Maybe 3rd-party substitute (GSL, Netlib).
- Mathematica has built-in C-code generator, e.g.

```
myfunc = Compile[{{x}}, x^2 + Sin[x^2]];
Export["myfunc.c", myfunc, "C"]
```

  But no standalone code: shared object for use with Mathematica (i.e. also needs license).
- FormCalc's code-generation functions produce optimized standalone code.

Code-generation Functions

FormCalc's code-generation functions are public and disentangled from the rest of the code. They can be used to **write out an arbitrary Mathematica expression as optimized Fortran or C code:**

- *handle* `= OpenCode["`*file.F*`"]`
  opens *file.F* as a Fortran file for writing,
- `WriteExpr[`*handle*`, {`*var*` -> `*expr*`, ...}]`
  writes out Fortran code which calculates *expr* and stores the result in *var*,
- `Close[`*handle*`]`
  closes the file again.

Code generation

Traditionally: Output in Fortran.
Code generator is meanwhile rather sophisticated, e.g.

- Expressions too large for Fortran are split into parts, as in

```
var = part1
var = var + part2
...
```

- High level of optimization, e.g. common subexpressions are pulled out and computed in temporary variables.
- Many ancillary functions make code generation versatile and highly automatable, such that the resulting code needs few or no changes by hand:
  `VarDecl`, `ToDoLoops`, `IndexIf`, `FileSplit`, ...

C Output

- **Output in C99** makes integration into C/C++ codes easier:

```
SetLanguage["C"]
```

Code structured by e.g.

- **Loops and tests handled through macros,** e.g. `LOOP(var, 1, 10, 1) ... ENDLOOP(var)`
- **Sectioning by comments,** to aid **automated substitution** e.g. with `sed`, e.g. `* BEGIN VARDECL ... * END VARDECL`
- Introduced **data types** `RealType` and `ComplexType` for better abstraction, can e.g. be changed to different precision.

MathLink

The **MathLink API** connects Mathematica with external C/C++ programs (and vice versa). **J/Link** does the same for Java.

```
:Begin:
:Function:      copysign
:Pattern:       CopySign[x_?NumberQ, s_?NumberQ]
:Arguments:     {N[x], N[s]}
:ArgumentTypes: {Real, Real}
:ReturnType:    Real
:End:

#include "mathlink.h"

double copysign(double x, double s) {
  return (s < 0) ? -fabs(x) : fabs(x);
}

int main(int argc, char **argv) {
  return MLMain(argc, argv);
}
```

For more details see arXiv:1107.4379.

Mathematica ↔ Fortran

Mathematica → Fortran:

- **Get FormCalc from http://feynarts.de/formcalc**
- **Write out arbitrary Mathematica expression:**

```
h = OpenCode["file"]
WriteExpr[h, {var -> expr, ...}]
Close[h]
```

Fortran → Mathematica:

- **Get http://feynarts.de/formcalc/FortranGet.tm**
- **Compile:** `mcc -o FortranGet FortranGet.tm`
- **Load in Mathematica:** `Install["FortranGet"]`
- **Read Fortran code:** `FortranGet["file.F"]`

FORM ↔ Mathematica

Mathematica → FORM:

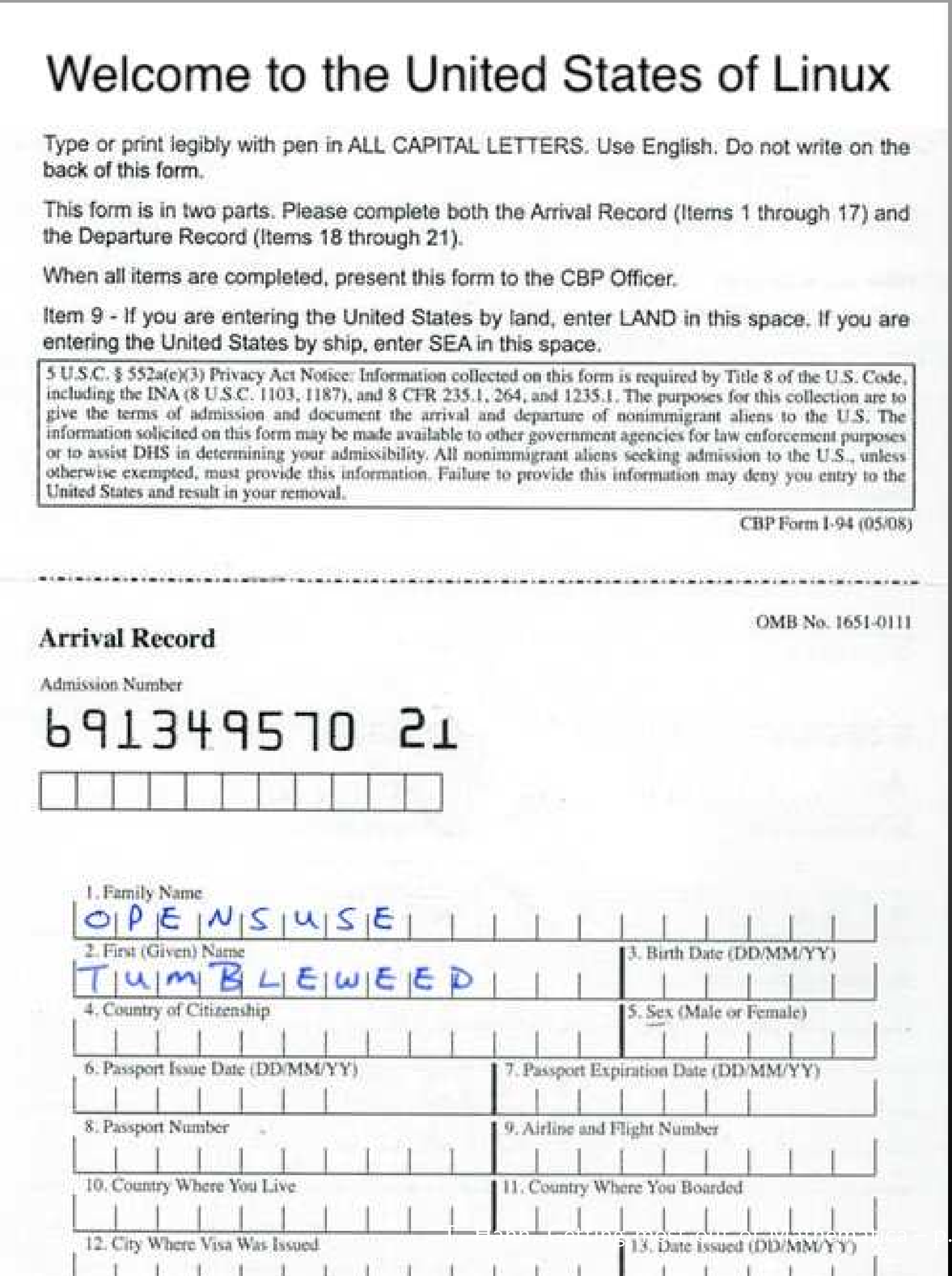
- Get FormCalc from http://feynarts.de/formcalc
- After compilation the `ToForm` utility should be in the executables directory (e.g. Linux-x86-64):

```
ToForm < file.m > file.frm
```

FORM → Mathematica:

- Get http://feynarts.de/formcalc/FormGet.tm
- Compile it with `mcc -o FormGet FormGet.tm`
- Load it in Mathematica with `Install["FormGet"]`
- Read a FORM output file: `FormGet["file.out"]`
  Pipe output from FORM: `FormGet["!form file.frm"]`

Computing at MPP



Welcome to the United States of Linux

Type or print legibly with pen in ALL CAPITAL LETTERS. Use English. Do not write on the back of this form.

This form is in two parts. Please complete both the Arrival Record (Items 1 through 17) and the Departure Record (Items 18 through 21).

When all items are completed, present this form to the CBP Officer.

Item 9 - If you are entering the United States by land, enter LAND in this space. If you are entering the United States by ship, enter SEA in this space.

5 U.S.C. § 552a(e)(3) Privacy Act Notice: Information collected on this form is required by Title 8 of the U.S. Code, including the INA (8 U.S.C. 1103, 1187), and 8 CFR 235.1, 264, and 1235.1. The purposes for this collection are to give the terms of admission and document the arrival and departure of nonimmigrant aliens to the U.S. The information solicited on this form may be made available to other government agencies for law enforcement purposes or to assist DHS in determining your admissibility. All nonimmigrant aliens seeking admission to the U.S., unless otherwise exempted, must provide this information. Failure to provide this information may deny you entry to the United States and result in your removal.

CBP Form I-94 (05/08)

OMB No. 1651-0111

Arrival Record

Admission Number

691349570 21

1. Family Name
OPENSUSE

2. First (Given) Name
TUMBLEWEED

3. Birth Date (DD/MM/YY)

4. Country of Citizenship

5. Sex (Male or Female)

6. Passport Issue Date (DD/MM/YY)

7. Passport Expiration Date (DD/MM/YY)

8. Passport Number

9. Airline and Flight Number

10. Country Where You Live

11. Country Where You Boarded

12. City Where Visa Was Issued

13. Date Issued (DD/MM/YY)

HTCondor

Batch system HTCondor manages Jobs

First time:
Condor needs a Keytab to obtain a valid Kerberos ticket on the Execute hosts

```
> condor_keytab
Password for kabel@MPPMU.MPG.DE:
-rw------- 1 kabel THEORY 58 Sep 15 13:18 /home/pccn2/condor/users/kabel/krb5.keytab
```

HTCondor

Every Job needs a Submit File. Example:

hello.sh:

```
#! /bin/bash
echo "Hello, $1"
```

hello.submit:

```
universe = vanilla
executable = hello.sh
arguments = $(Process)
output = hello.$(Process).out
error = hello.$(Process).err
log = hello.$(Process).log
requirements = Pool == "Theory"
queue 5
```

```
condor_submit hello.submit
```

More (RTM): `condor_remove` `condor_q` `condor_status`

Virtualization

General idea: run one compute environment inside another.

- Case 1: run Windows (programs) on Tumbleweed.
  Purpose: run 'incompatible' software on Linux.
  - Run wine (works for many Windows programs), start with `wineboot`.
  - Run VirtualBox: near-perfect binary encapsulation.
- Case 2: run other Linux flavor on Tumbleweed.
  Purpose: decouple machine OS from software stack.
  - Run Docker/Singularity container (no performance penalty over 'bare metal').

Scientific Software Stacks

Software Stack (hep-ex) =

User software

↳ Subgroup software

↳ Experiment software

↳ Community software

↳ Frameworks (ROOT, Geant4, Anaconda, ...)

↳ Low-level libraries (FFTW, Cuda, ...)

Typical problems:

- Compilers too old/new
- Libraries missing/wrong version
- Build system errors (autotools etc)
- System upgrade breaks software stack

Linux Containers

Linux 4.x+ allows to ‘compartmentalize’ systems, i.e.

- same Kernel (syscalls very stable across versions),
- potentially different set of libraries,
- own process table,
- yet access to common resources like GPUs,
- no virtualization layer, i.e. ‘bare-metal’ performance,
- essentially running different flavor side-by-side.

Main Implementations:

- Docker
- Singularity
- For ATLAS: cvmfs

Using Singularity

Run a shell in the Singularity container:

```
singularity shell /path/to/container.sqsh
```

Run a program directly:

```
singularity exec /path/to/container.sqsh program args...
```

`--nv` enables Cuda features.

Or use Oliver Schulz's wrapper at

github.com/oschulz/singularity-venv

Images corresponding to former versions of Ubuntu at MPP can be found here:

```
/remote/ceph/common/vm/singularity/images
```

Mathematica Summary

- Mathematica makes it wonderfully easy, even for fairly unskilled users, to manipulate expressions.
- Most functions you will ever need are already built in. Many third-party packages are available at MathSource, https://library.wolfram.com/infocenter/MathSource.
- When using its capabilities (in particular list-oriented programming and pattern matching) right, Mathematica can be very efficient.
  Wrong: `FullSimplify[veryLongExpression]`.
- Mathematica is a general-purpose system, i.e. convenient to use, but not ideal for everything.
  For example, in numerical functions, Mathematica usually selects the algorithm automatically, which may or may not be a good thing.