NEW PROGRAMMING LANGUAGES AND PARADIGMS IN HPC

SEMINAR „NEUESTE TRENDS IM HOCHLEISTUNGSRECHNEN“

Lukas Stabe / 2015-12-07

STRUCTURE

1. Introduction
2. Definitions
3. Advantages of new languages/paradigms
4. Problems
5. Examples
    - SciPy
    - Rust
    - Swift/T
    - OpenMP 4
6. Conclusion

DEFINITIONS: LANGUAGE

- *„A programming language is a formal constructed language designed to communicate instructions to a machine, particularly a computer."*
  – Wikipedia
- A programming language defines how you tell the computer to do something
- Languages are closely related to their standard library
  - Boundaries are often unclear

DEFINITIONS: PARADIGM

- *„A programming paradigm is a fundamental style of computer programming, serving as a way of building the structure and elements of computer programs.“*
  – Wikipedia
- Describes a way to approach problems
- Defines common patterns
- Often explicitly forbids some *anti-patterns*

DEFINITIONS: RELATION

- *„Capabilities and styles of various programming languages are defined by their supported programming paradigms; some programming languages are designed to follow only one paradigm, while others support multiple paradigms.“*
  – Wikipedia
- Most languages support a mix of paradigms
- Standard library may be written with a concrete paradigm in mind

ADVANTAGES OF NEW LANGUAGES/PARADIGMS

- Simplify development
- Fewer kinds of errors possible
- Produces easier-to-maintain code
  - Easier to write (in a good/idiomatic manner) for inexperienced programmers
  - This is a result of the community surrounding the language
  - Unit-testing
  - Documentation
- Better utilize available resources

PROBLEMS

- A large existing codebase of C/Fortran code
- Smaller ecosystem of libraries/tools (esp. related to HPC)
- Huge expertise of experienced programmers
- C/Fortran compilers have been worked on for decades, so they can optimize code extremely well

EXAMPLE: SCIPY

- Python library
- Wraps compiled Fortran and C code
- Write program flow and high-level structure in Python
- Keep hotspots in compiled code
- Near-native performance

EXAMPLE: RUST

- Compiled low-level language
- Strong type and generics system with type inference
- Guarantees memory safety
- Thread-safety
- MPI bindings in development

EXAMPLE: RUST

```
fn main() {
    // A simple integer calculator:
    // `+` or `-` means add or subtract by 1
    // `*` or `/` means multiply or divide by 2
    let program = "+ + * - /";
    let mut accumulator = 0;

    for token in program.chars() {
        match token {
            '+' => accumulator += 1,
            '-' => accumulator -= 1,
            '*' => accumulator *= 2,
            '/' => accumulator /= 2,
            _ => { /* ignore everything else */ }
        }
    }
}
```

EXAMPLE: RUST

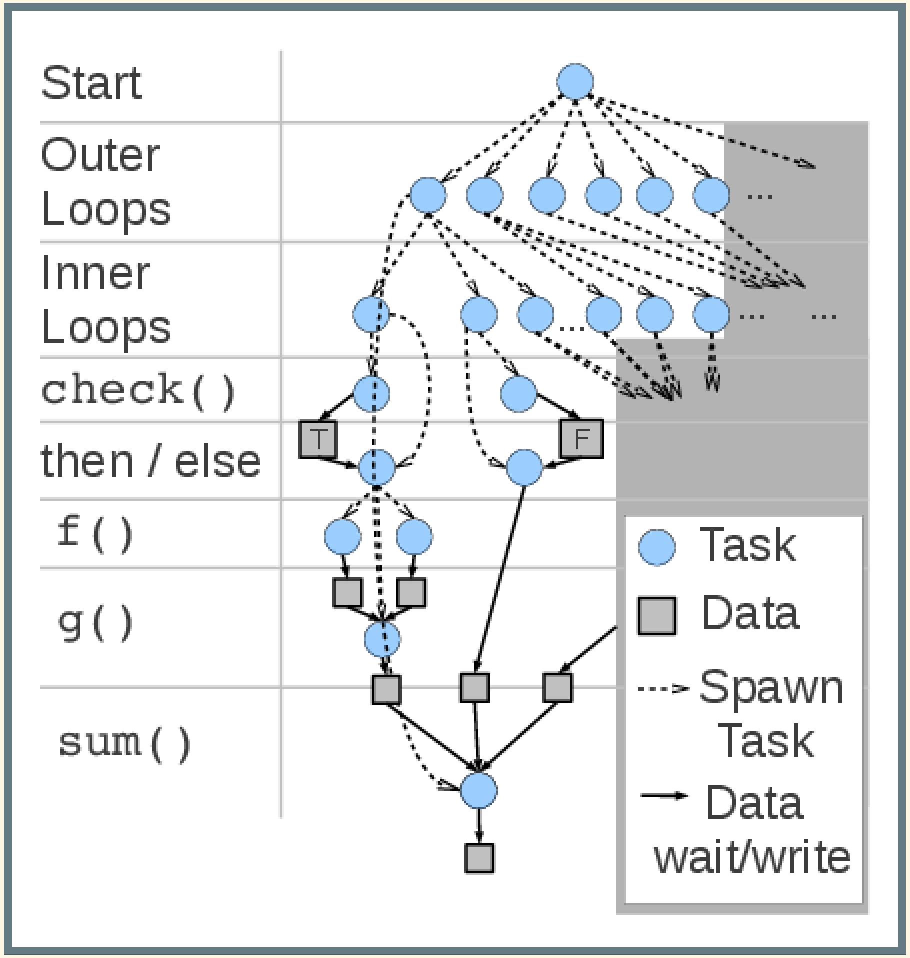
```rust
extern crate mpi;
use mpi::traits::*;
fn main() {
    let universe = mpi::initialize().unwrap();
    let world = universe.world();
    let size = world.size();
    let rank = world.rank();

    if size != 2 {
        panic!("Size of MPI_COMM_WORLD must be 2, but is {}!", size);
    }
    match rank {
        0 => {
            let msg = vec![4.0f64, 8.0, 15.0];
            world.process_at_rank(rank + 1).send(&msg[..]);
        }
        1 => {
            let (msg, status) = world.receive_vec::<f64>();
            println!("Process {} got message {:?}.\nStatus is: {:?}",
                rank, msg, status);
        }
        _ => unreachable!()
    }
}
```

EXAMPLE: SWIFT/T

- Swift script translates into MPI program
- Calls leaf tasks written in C, C++, Fortran, Python, R, Tcl, Julia, Qt Script, or executable programs
- Coordinates data flow between leaf tasks
- Executes leaf tasks concurrently where possible

EXAMPLE: SWIFT/T

```
int X = 100, Y = 100;
int A[][];
int B[];
foreach x in [0:X-1] {
  foreach y in [0:Y-1] {
    if (check(x, y)) {
      A[x][y] = g(f(x), f(y));
    } else {
      A[x][y] = 0;
    }
  }
  B[x] = sum(A[x]);
}
```



EXAMPLE: OPENMP 4

- Compiler directives on top of C, C++ and Fortran
- Interesting new features in version 4
  - SIMD directive
    - Uses vector units like AVX/SSE and NEON to do multiple numeric operations in parallel on one core
    - Works combined with `omp parallel`
  - TARGET directive
    - Runs code on accelerators
    - transfers in- and output data back and forth

EXAMPLE: OPENMP 4

```
void vadd_openmp(float *a, float *b, float *c, int len)
{
    #pragma omp target map(to:a[0:len],b[0:len],len) map(from:c[0:len])
    {
        int i;
        #pragma omp parallel for
        for (i = 0; i < len; i++)
            c[i] = a[i] + b[i];

    }
}
```

CONCLUSION

- New languages and paradigms can provide big benefits
  - Easier development
  - Easier-to-maintain code
  - Utilize new types of hardware
- They need to overcome some significant challenges
  - Large existing codebase/ecosystem
  - Raw speed
- Nothing can replace C/C++/Fortran right now
  - Rust looks promising

SOURCES

- Quote on slide 3: Wikipedia: Programming language
- Quote on slide 4, 5: Wikipedia: Programming paradigm
- Sample code on slide 10: rust-lang.org
- Sample code on slide 11: GitHub: bsteinb/rsmpi
- Image and sample code on slide 13: swift-lang.org/Swift-T
- Sample code on slide 15: TI Wiki: OpenMP Accelerator Model 0.3.3