The Joy and Pain of using D for HPC

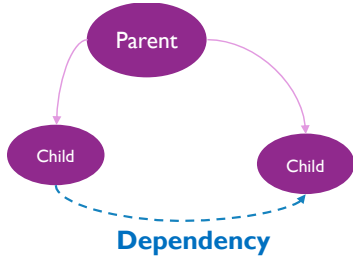
Is D ready for Supercomputers?

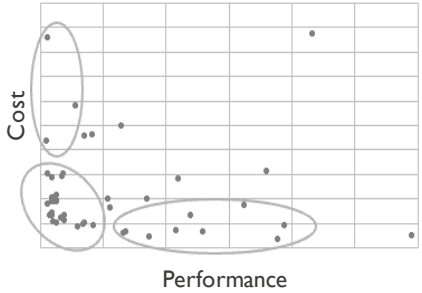
Tom Vander Aa
HPC LAB

Overview

Is D for High Performance Computing?

- Why programming supercomputers is hard
- Tasks as the main paradigm
- Walking the software stack – from D to C
- Intermezzo: our own supercomputer architecture
- The joy and pain on using D for HPC





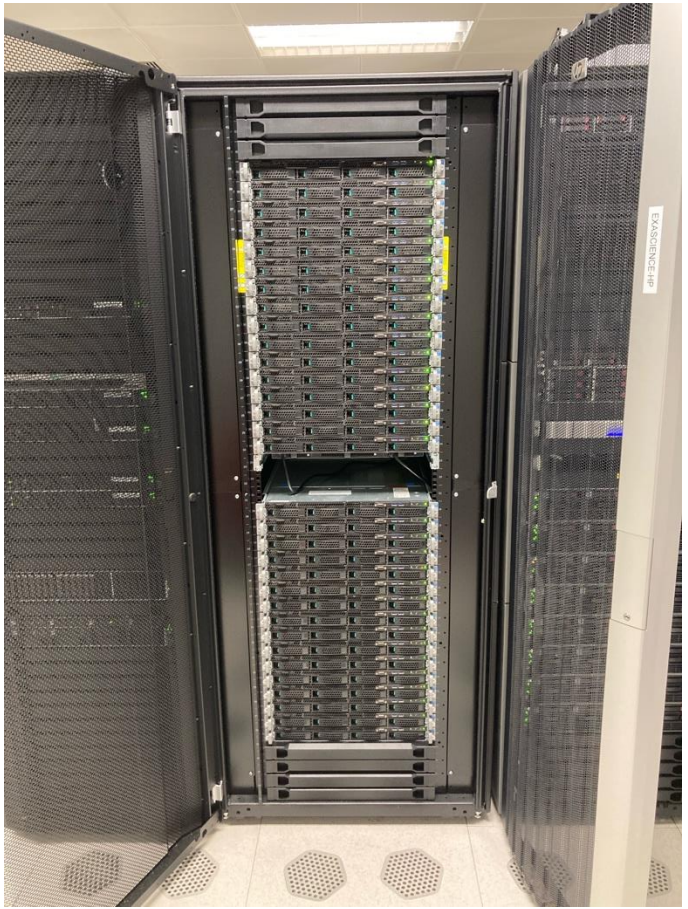
imec is nanotechnology

- Globally, the leading independent R&D center in nanotechnology since 1984
- >5500 international R&D top talents
- >€3.5B invested in leading-edge semiconductor fabs
- Health and life sciences, mobility, industry 4.0, agrifood, smart cities, sustainable energy, etc.

We do Hardware-Software Co-design

Better Software on Better Hardware

Engineering simulations
the backbone of commercial HPC



HPCG Benchmark runs poorly on top supercomputers

#	Name	Peak Performance	Cores	HPCG performance	Actual Usage
1	Frontier AMD EPYC, AMD Instinct	1.7 ExaFlop/s	8,699,904	14.1 PetaFlop/s	0,82%
2	Aurora Intel Xeon Intel DPU	2 ExaFlop/s	9,264,128	5,6 PetaFlop/s	0,28%
3	Eagle Intel Xeon Nvidia H100	857 PetaFlop/s	2,073,600	(not measured)	
4	Fugaku Fujitsu A64FX	537 PetaFlop/s	7,630,848	16 PetaFlop/s	2,98%
5	LUMI AMD EPYC, AMD Instinct	429 PetaFlop/s	2,220,288	3.4 PetaFlop/s	0,86%

Source: TOP500 List June 2024, Rmax/Rpeak, HPCG/Rpeak

High performance computing

We get the most out of available hardware, in the ***large*** or the ***small***

Fast Network Interconnect, e.g. Infiniband

MPI

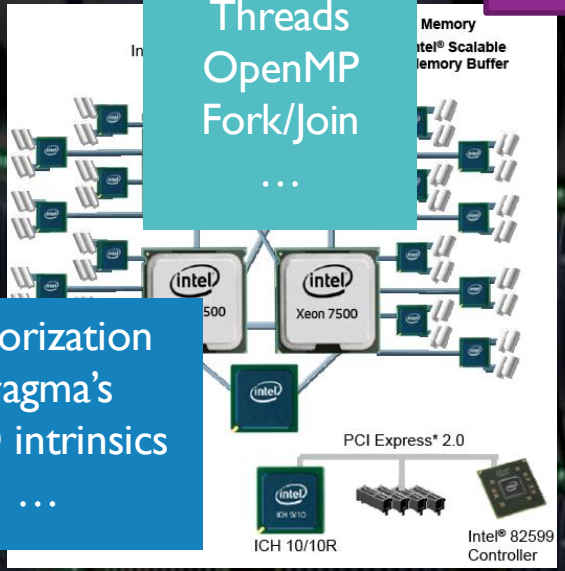
…

Threads
OpenMP
Fork/Join

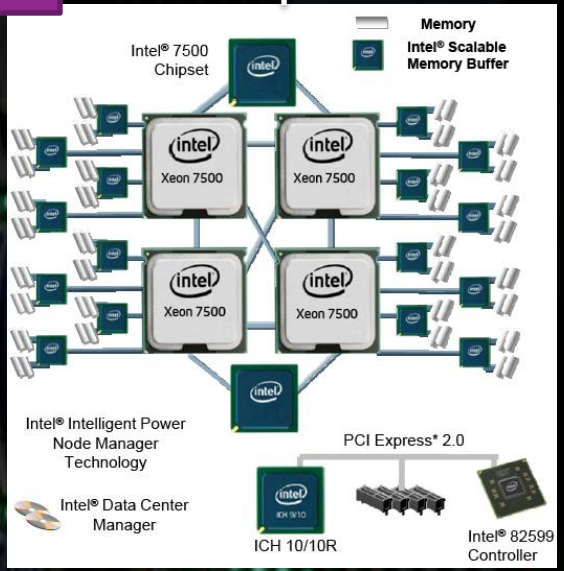
…

Vectorization
Pragma's
SIMD intrinsics

…



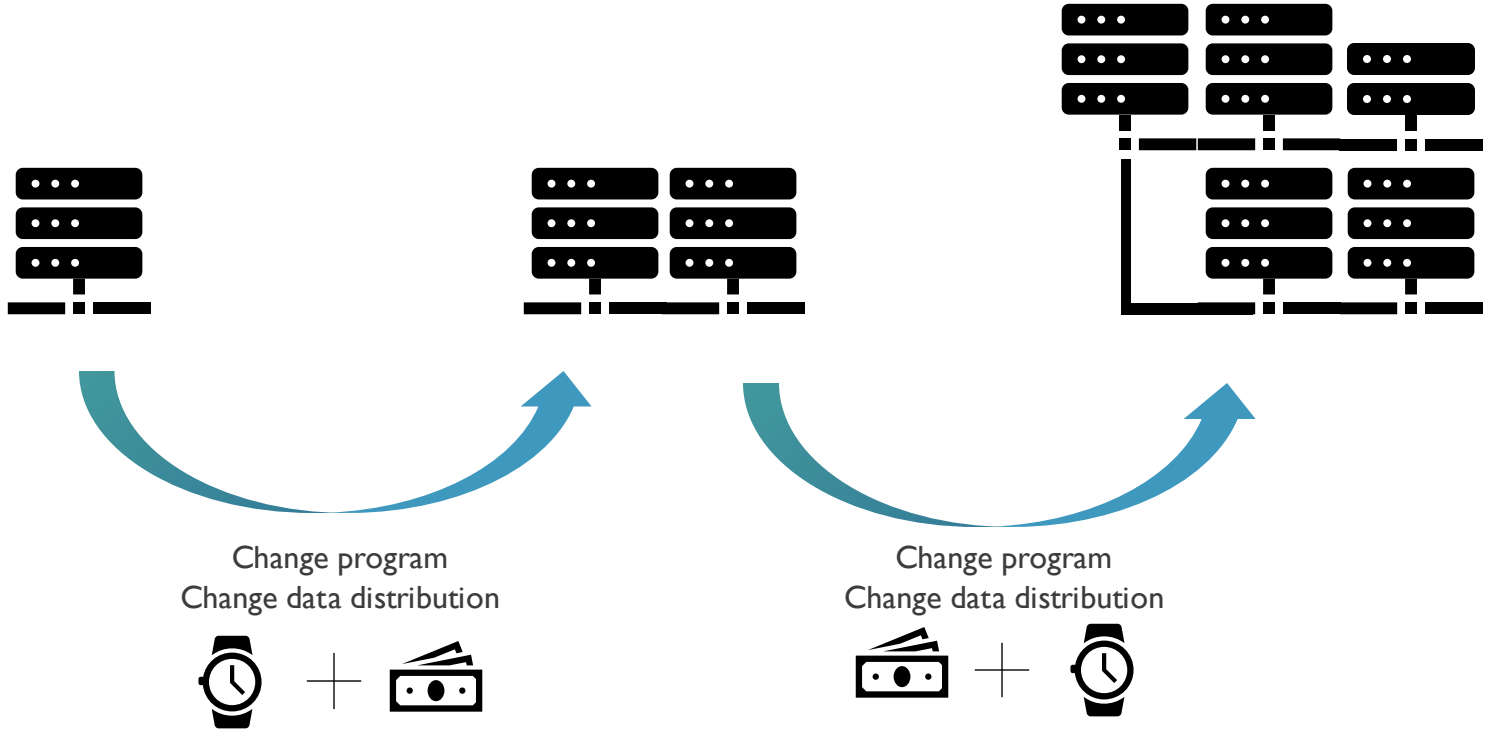
Shared Memory System



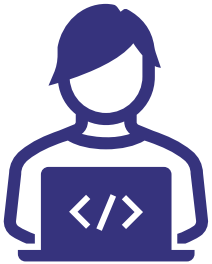
Shared Memory System

Why create a new programming model ?

Existing approaches require considering other parallel entities



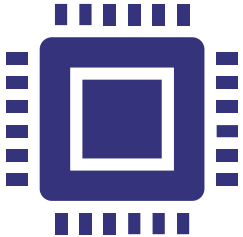
Separation of Responsibility



Dev

Functionality

Parallelism



Runtime

Scheduling

Distributed Data storage

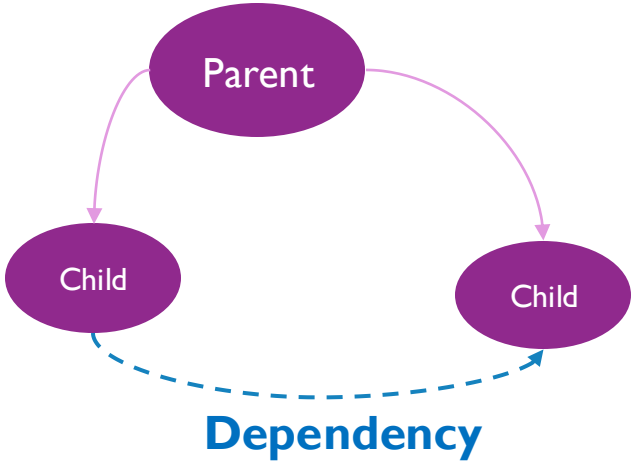
Load balancing

Task is the main building block

A task …

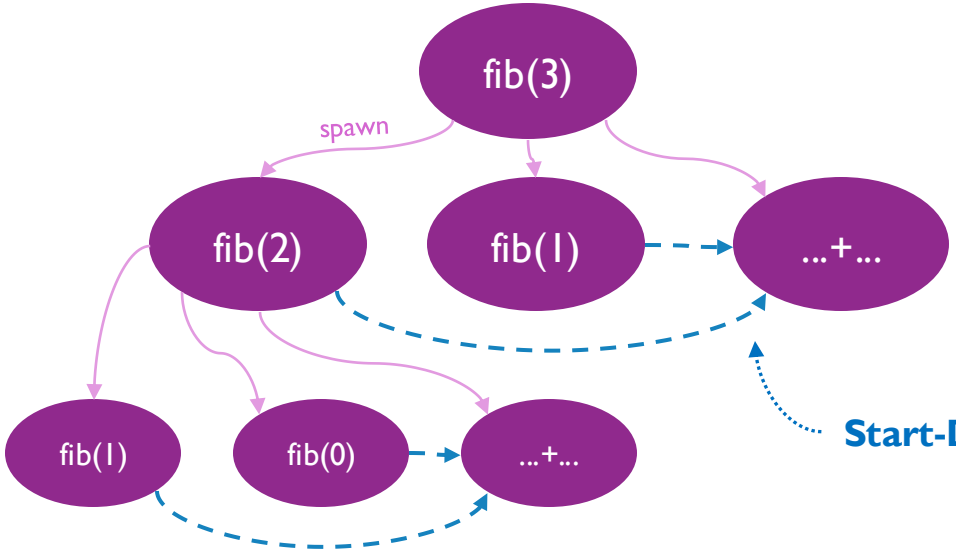
- Smallest Unit of Compute
- Is executable
- May run in parallel
- May spawn new tasks
- Has dependencies on other tasks
- Input is read-only
- Output is write-only

} Key to good performance



Tasks and dependencies

Tasks have dependencies on when they can start




```
fib_2 = newTask("FIB_2", &fib_tsk,
                [fib2, 2], []);

fib_1 = newTask("FIB_1", &fib_tsk,
                [fib1, 1], []);

fib_add = newTask("F_ADD", &fib_add,
                  [fib3,fib1,fib2],
                  [fib_1, fib_2]);
```

```
int fib(n) {
  if( n < 2) return 1;
  return fib(n-1) + fib(n-2);
}
```

The Software Stack

Let's walk down the Stack

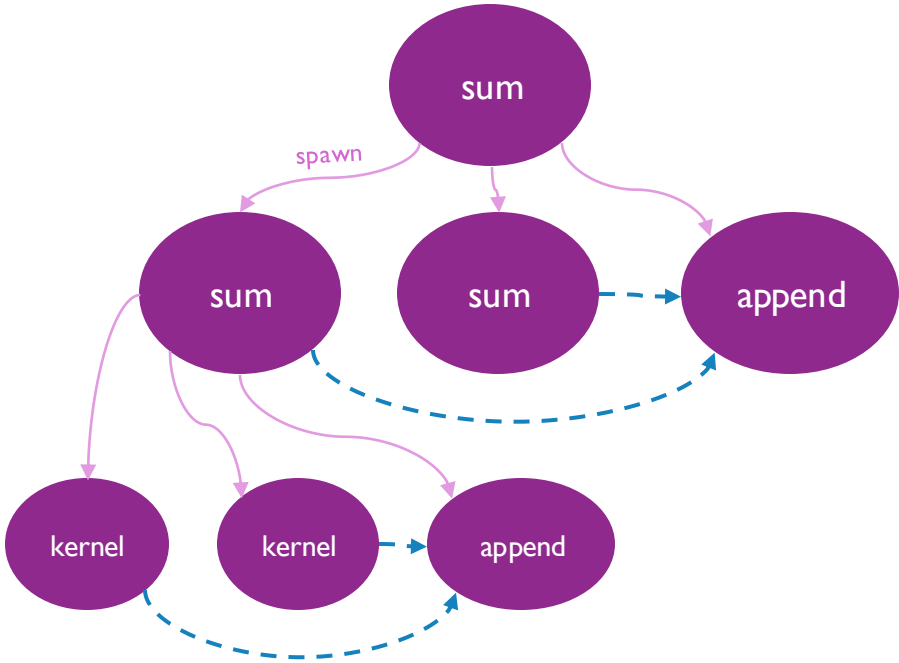
<table>
<tr><td rowspan="4">D</td><td colspan="3">User Application</td></tr>
<tr><td colspan="3">Application Libraries: Neural Nets, Linear Algebra</td></tr>
<tr><td colspan="3">Standard Library: Task, Map, Reduce</td></tr>
<tr><td colspan="3">Runtime (D)</td></tr>
<tr><td rowspan="3">C</td><td colspan="3">Runtime (C)</td></tr>
<tr><td colspan="2">FreeRTOS common runtime in C</td><td>POSIX Threads</td></tr>
<tr><td>FreeRTOS RISC-V</td><td>FreeRTOS POSIX</td><td>Standard Linux</td></tr>
<tr><td>HW</td><td>Management Processor</td><td colspan="2">Your Laptop / Your Supercomputer</td></tr>
</table>

Fundamental Pattern

Task Splitting and Kernels

```
void sum(
      Vector!float C,
      in Vector!float A,
      in Vector!float B)
{
      if (length <= cutOff)
            xo!sumKernel(C, A, B);

      else // if (length > cutOff)
            xo!sum(C1, A[0..n/2], B[0..n/2])
            .and(xo!sum(C2, A[n/2..n], B[n/2..n]))
            .then!appendRows(C, C1, C2);
}
```




```
@bariKernel
void sumKernel(Matrix!float C, in Matrix!float A, in Matrix!float B)
{
   foreach(rix; 0..n)
      foreach(cix; 0..m)
         C[rix, cix] = A[rix, cix] + B[rix, cix];
}
```

Map-Reduce

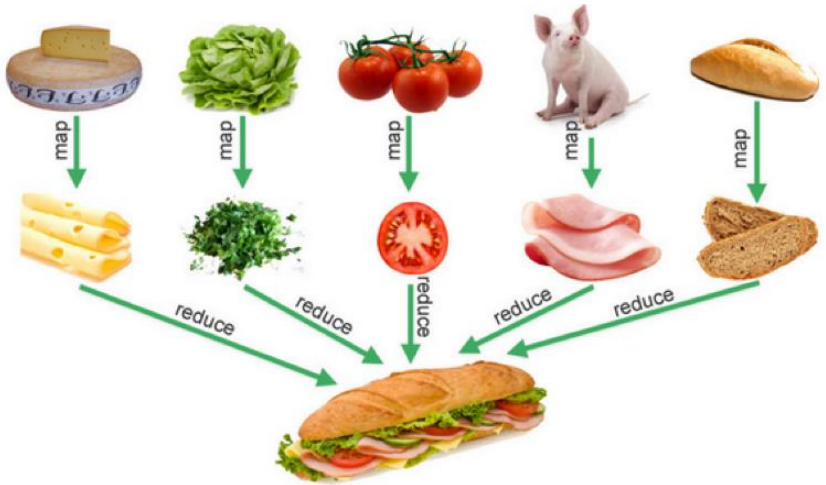
As in std.parallelism Taskpool.map / reduce

```
auto N = 16 * 1024 * 1024;
auto vector = Vector!float(N);
auto result = Vector!float(1);

auto initTask = map!(init)(vector, N);

auto sumTask = reduce!(sum)(
    result, [vector], N, [initTask]
);

newTask!printVec(result, N, [sumTask]);
```



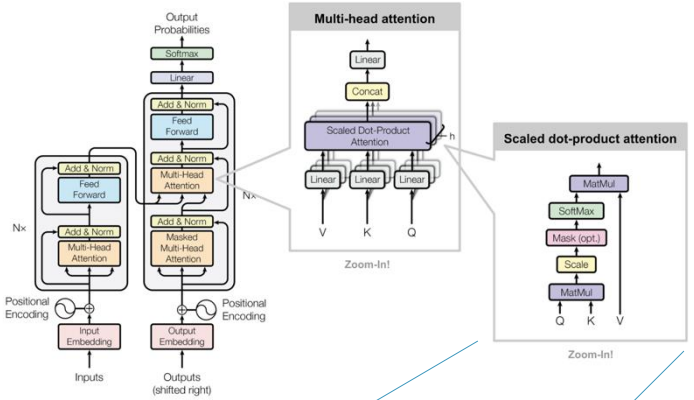
Composing task into libraries and applications

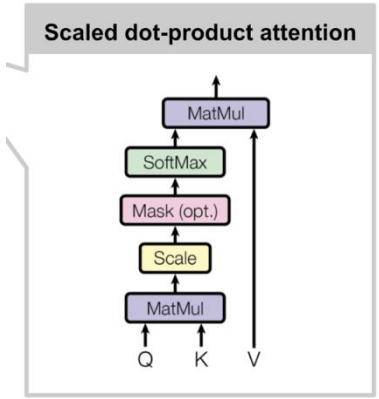
GPT Application

- Similar abstraction level as using PyTorch
- 3 layers, 1.34M parameters

```
//ATTENTION LAYER
auto tsk = xo!dotProduct(K, X, Wk)
          .and(xo!dotProduct(Q, X, Wq))
          .then!dotProductTranspose(QKt, Q, K)
          .then!softmax(A, QKt);


xo!dotProduct(V, X, Wv)
   .and(tsk)
   .then!dotProduct(Out, A, V)
   .then!printMatrix("Att", Out);
```





tensor library

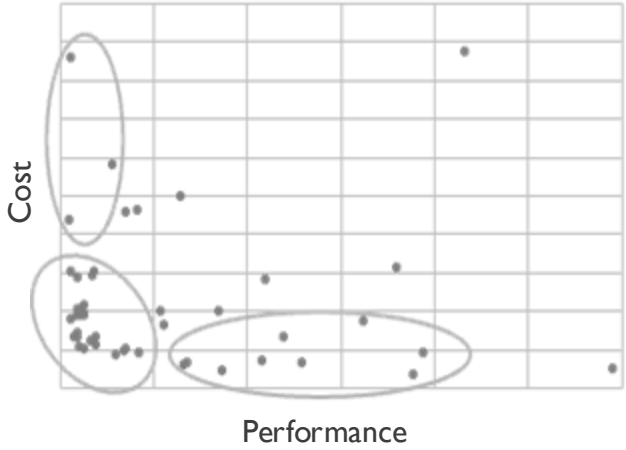
The Software Stack

Let's walk down the Stack

<table>
<tr><td rowspan="4">D</td><td colspan="3">User Application</td></tr>
<tr><td colspan="3">Application Libraries: Neural Nets, Linear Algebra</td></tr>
<tr><td colspan="3">Standard Library: Task, Map, Reduce</td></tr>
<tr><td colspan="3">Runtime (D)</td></tr>
<tr><td rowspan="3">C</td><td colspan="3">Runtime (C)</td></tr>
<tr><td colspan="2">FreeRTOS common runtime in C</td><td>POSIX Threads</td></tr>
<tr><td>FreeRTOS RISC-V</td><td>FreeRTOS POSIX</td><td>Standard Linux</td></tr>
<tr><td>HW</td><td>Management Processor</td><td colspan="2">Your Laptop / Your Supercomputer</td></tr>
</table>

Hardware-Software System Co-design

for HPC and AI applications



- Co-design of system software and hardware
- Focus on solving data movement bottlenecks
- Match application performance and cost sweet-spot

- A novel hardware/software system
  providing implicit application scalability
  - RISC-V based compute core optimized for HPC and AI
  - Hardware accelerators for task and data management

Resulting in this System Board

- Compute Array
- Management Processor
- Many components dedicated to moving data

The Software Stack

Let's walk down the Stack

<table>
<tr><td rowspan="4">D</td><td colspan="3">User Application</td></tr>
<tr><td colspan="3">Application Libraries: Neural Nets, Linear Algebra</td></tr>
<tr><td colspan="3">Standard Library: Task, Map, Reduce</td></tr>
<tr><td colspan="3">Runtime (D)</td></tr>
<tr><td rowspan="3">C</td><td colspan="3">Runtime (C)</td></tr>
<tr><td colspan="2">FreeRTOS common runtime in C</td><td>POSIX Threads</td></tr>
<tr><td>FreeRTOS RISC-V</td><td>FreeRTOS POSIX</td><td>Standard Linux</td></tr>
<tr><td>HW</td><td>Management Processor</td><td colspan="2">Your Laptop / Your Supercomputer</td></tr>
</table>

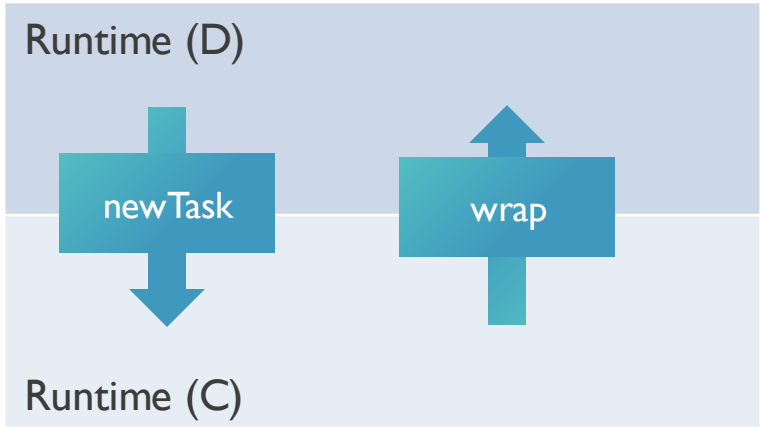
Bridge D to C using templates and mixins

D → C: newTask

```
template
TaskId newTask(alias fun, ArgsGiven...)(auto ref ArgsGiven argv)
{
   // create an "extern (C)" equivalent with argc, argv[]
   DvmTaskFunc funptr = &(wrap!fun);
   ulong[] args = new ulong[Args.length];


   // verify + convert arguments
   static foreach (int i, ArgT; Args)
      static if (hasUDA!(ArgT, DistributedDatastructure))
         args[i] = argv[i].oid;
      else
         args[i] = argv[i].asULong;


   return newTask(funptr, args);
}
```

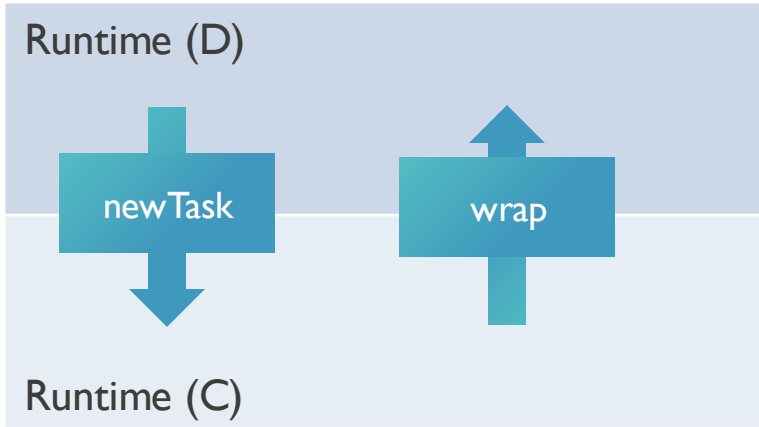


Bridge D to C using templates and mixins

C → D: wrap

```
template wrap(alias fun)
{
   import std.traits;


   alias Args = Parameters!fun;
   enum ArgsCnt = Args.length;
   extern (C) void wrap(ulong* argv, ulong argc)
   {
      static foreach (int i, arg; Args)
      {
         mixin(ParamsHelper!(i, arg, "funArg" ~ i.stringof));
      }
      mixin("fun(" ~ InjectParams!ArgsCnt ~ ");");
   }
}
```



ParamsHelper!(1, double, ”c”)	double c=argv[1].fromULong!(double);
ParamsHelper!(2, Vector!ulong, ”b”)	Vector!ulong b=Vector!ulong.fromOID(argv[2]);

The joy and pain of betterC

Small and efficient code on a small RISC-V processor

- ⭐ We take happily with us in betterC land
  - Mixins, Templates
  - Unit Tests
  - Imports
  - Array Slicing
  - Some of the standard library (string, …)

- ⛔ We have to leave some things behind
  - Most of the standard library (arrays, stdio)
  - Classes 😔 (only struct)
  - Type Info (fullyQualifiedName)
  - Threading and synchronizations

DVM Runtime (C)		
FreeRTOS common runtime in C		POSIX Threads
FreeRTOS RISC-V	FreeRTOS POSIX	Standard Linux
Management Processor	Your Laptop / Your Supercomputer	

The joy and pain of betterC

Use FreeRTOS to overcome the limitations

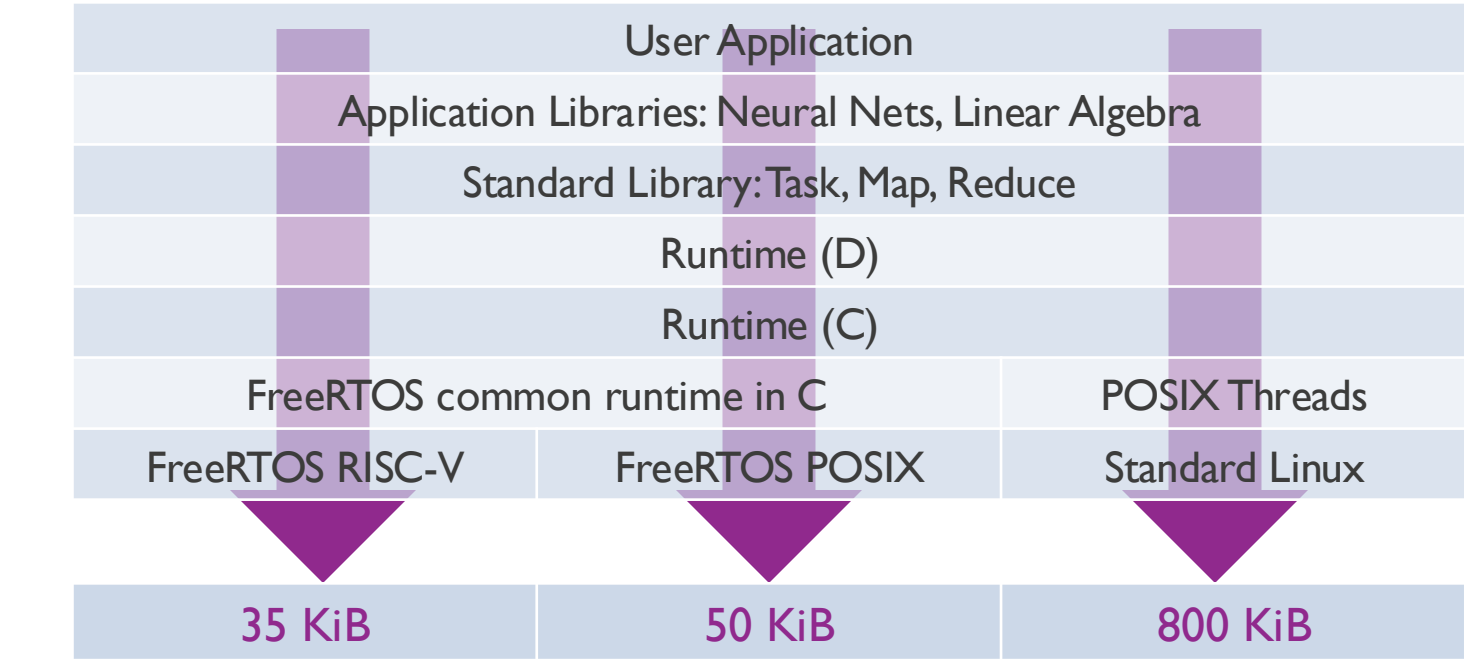
Runtime (C)		
FreeRTOS common runtime in C		POSIX Threads
FreeRTOS RISC-V	FreeRTOS POSIX	Standard Linux
Management Processor	Your Laptop / Your Supercomputer	

- FreeRTOS has
  - semaphores
  - threads
  - malloc
  - console logging
- We implemented an new DVec

```
version (D_BetterC)
{
  extern (C) void console_log(ulong level, const char* fmt, ...);
  void log(Args...)(LogLevel level, const(char*) format, Args args)
  {
    console_log(level, format, args);
  }
}
else
{
  void log(Args...)(LogLevel level, const(char*) format, Args args)
  {
    import core.stdc.stdio;
    if (level < LogLevel.DEBUG)
      printf(format, args);
    }
}
```

The joy and pain of betterC

The result: small code



static stripped binary
all inclusive

The Good, The Bad And The Ugly

- D is a very powerfull language, if you know what you're doing
- D is not easy to learn and has some quirks
- Sometimes you need to write ugly code

The Good

D is a very powerful language, if you know what you are doing

- Small code footprint with betterC
- LDC for RISC-V was a breeze

- Powerful language
  - Single codebase with version and static if/for
  - Templates, mixins, reflection

- Some cool features (gems)
  - Uniform Function Call Syntax (UFCS)
  - Scope guards
  - And many more

The Bad

D is not easy to learn and has some quirks

- Not an easy language to learn
  - Small user base
- Integration with C
  - Should I use ImportC, dpp, dtep, ctod or htod???
  - Not clear what is in betterC
- Thread local is the default
  - shared ripples through
  - __gshared → you're on your own

The Ugly

Sometimes you need to write ugly code

- Proprietary dub build system
  - CMake hack to integrate dub
  - Generated dub.json (366 lines of dub)

```
add_custom_command(
        OUTPUT lib_rtm_d.a
        COMMAND export DPATH=${CMAKE_CURRENT_BINARY_DIR} &&
        rm -f ${CMAKE_CURRENT_BINARY_DIR}/lib_rtm_d.a &&
        dub bu   ild --config=rtmlib --build=${DUB_TARGET} ${DUB_DEBUG} &&
        mv bin/lib_rtm_d.a ${CMAKE_CURRENT_BINARY_DIR}
        BYPRODUCTS ${CMAKE_SOURCE_DIR}/d_apps/bin
        WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}/d_apps
        DEPENDS ${SST_RTM_D_SOURCES} stats
)
```

- Integration with existing libraries is cumbersome
  - MPI, HDF5

Hacking support for MPI and HDF5

The Ugly

```
/**
   mpi.d: MPI wrapper, it is a partial
          conversion of mpi.h.
*/

// only 50 MPI functions supported
```

- MPI
  - Found at least 3 repos on GitHub
  - Each using own *mpi.d*
  - Each with different functions supported
- HDF5
  - We could not get it to work
  - Using *h5dump* + pipes 

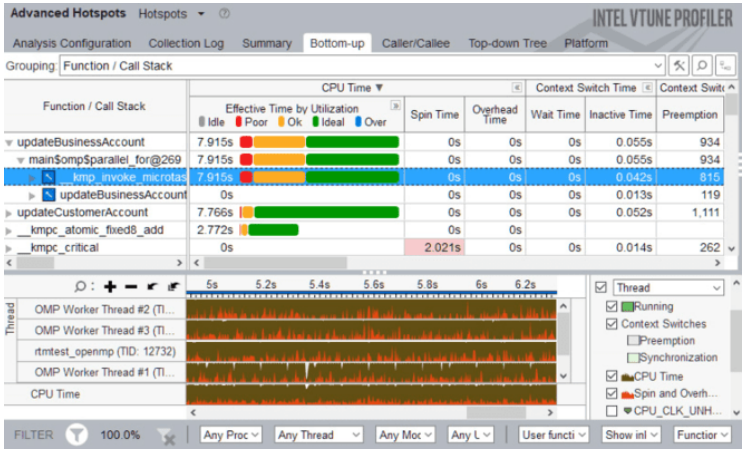
```
auto args = [
    "h5dump",
    /* raw data */ "-y",
    "-d", dataset,
    /* Litle Endian*/ "-b", "LE",
    /*Output raw to stderr */ "-o", "/dev/stderr",
    /*Inputfile: */ filename
];

auto pipes = pipeProcess(args, stdout | stderr);

while (true)
{
    T[] buf = pipes.stderr.rawRead(buffer);
    result ~= buf;
    if (buf.length < bufferLen)
        break;
}
```

Conclusion

Is there a Future for D in HPC?



Benefits of D for HPC

- Powerful, Fast, Compact, Efficient
- We keep on using D 👍

Roadblocks for acceptance in HPC 🚧

- Not enough users in HPC
  - Bootstrapping problem
- Bindings for the fundamental libraries
  - MPI, BLAS, HDF5, …
- Profiling and debugging
  - Distributed profiling and debugging

My goal for this DConf