INTEGRATED FORECASTING SYSTEM PERFORMANCE OPTIMIZATION

RICHARD GRAHAM, DMITRY PEKUROVSKY

GOALS OF ECTRANS OPTIMIZATION

- Focus on application performance from the perspective of communication performance
  - Goal is to reduce run-time, not focus on network performance as such
- Systems targeted
  - Current production system
  - Future systems
- Approach
  - Restructure parts of the code in a hardware agnostic manner
  - Leverage hardware optimizations, where it makes sense
- Current high-level technical goals
  - Reduce memory traffic associated with communication
  - Aim to overlap communication and computation
  - Leverage network asynchronous capabilities
- Opportunistic optimizations
  - Several already identified

CAPABILITIES LEVERAGED IN THE OPTIMIZATIONS

INFINIBAND CAPABILITIES TO BE LEVERAGED

- NIC hardware-level gather/scatter capabilities (UMR) to replace data packing
- Leverage BlueField offloaded collectives for blocking (and non-blocking) collectives

INFINIBAND GATHER/SCATTER (UMR) CAPABILITIES

BLUEFIELD DATA PROCESSING UNIT

Infrastructure Services

Data Center on a Chip

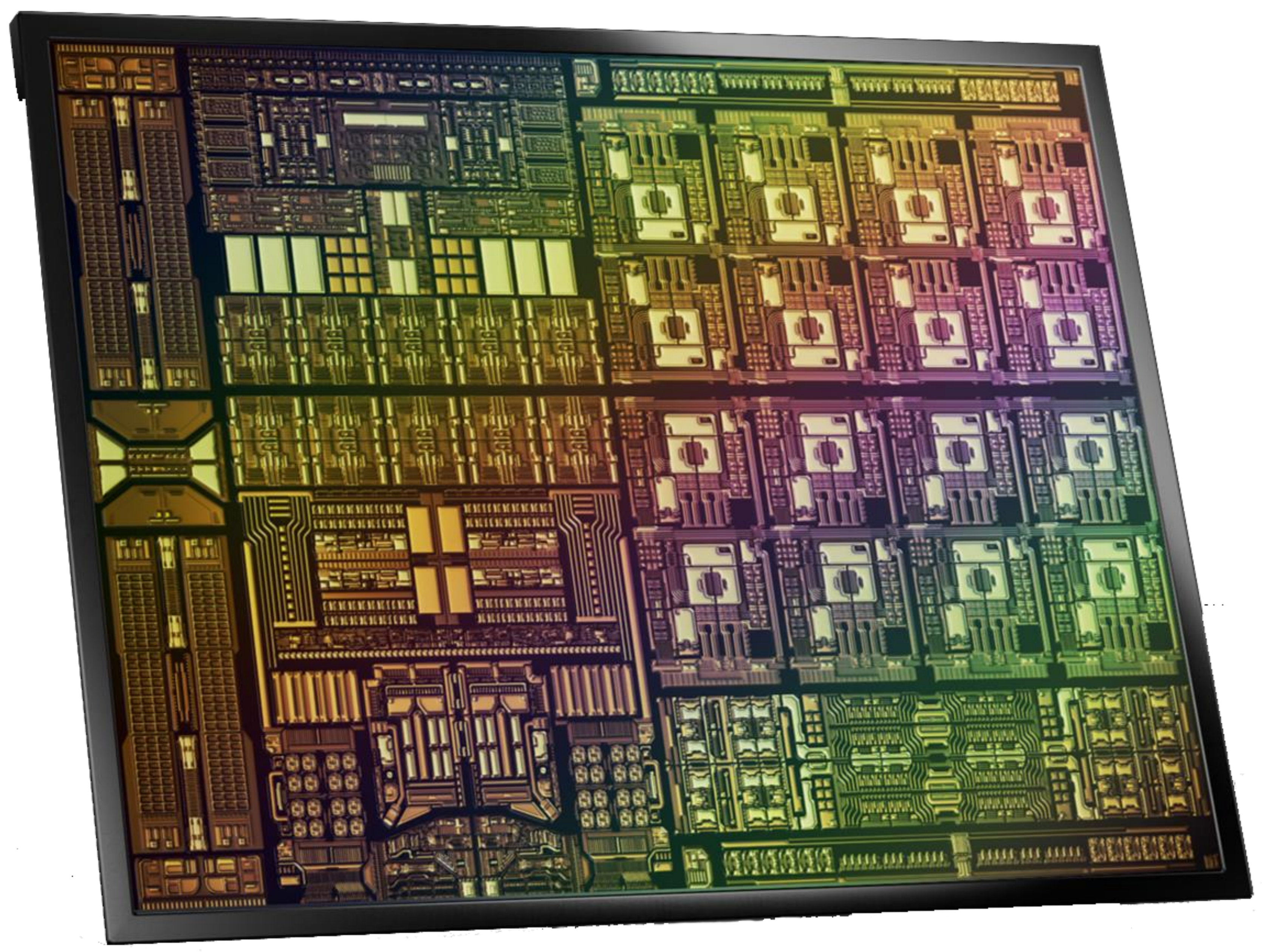
16 Arm 64-Bit Cores

16 Core / 256 Threads Datapath Accelerator

ConnectX InfiniBand / Ethernet

DDR memory interface

PCIe switch



BLUEFIELD DESIGN CONSIDERATIONS CONSIDERATION

- Asynchronous with respect to the compute engines
- At least one order of magnitude less compute capabilities than the compute complex
  - Selective as to how much work to provide, so as not to become the bottleneck
  - Requires work sharing
- DPU cores may be less powerful computationally with respect to the host compute engines
- DPU have targeted acceleration engines
- Host and DPU need to be “in sync”
- Network access
  - Source/destination of network traffic
  - Can post network requests on behalf of memory locations that are host-resident
  - Agnostic to they type of compute host

BLUEFIELD DESIGN CONSIDERATIONS CONSIDERATION

- BlueField enhancements
  - Work requests can be posted on behalf of memory that is host-resident – Cross-GVMI memory keys
  - Some optimized data paths between the host and the BlueField – GGA
- Possess memory bandwidth independent of that of the host
  - Selectively use this memory resource to supplement what is available in the compute complex – not an all or none proposition
- Can’t do any better than saturate the network BW – need to do just enough to saturate the network

MPI DATA TYPES

- Allow the user to create user-defined data layouts
  - May describe non-contiguous data layout
- The data types may be passed to MPI routines that take data type arguments
- It is up to the MPI implementation as to how it handles these
  - May just pack the data with memory copies
  - May use gather/scatter engines

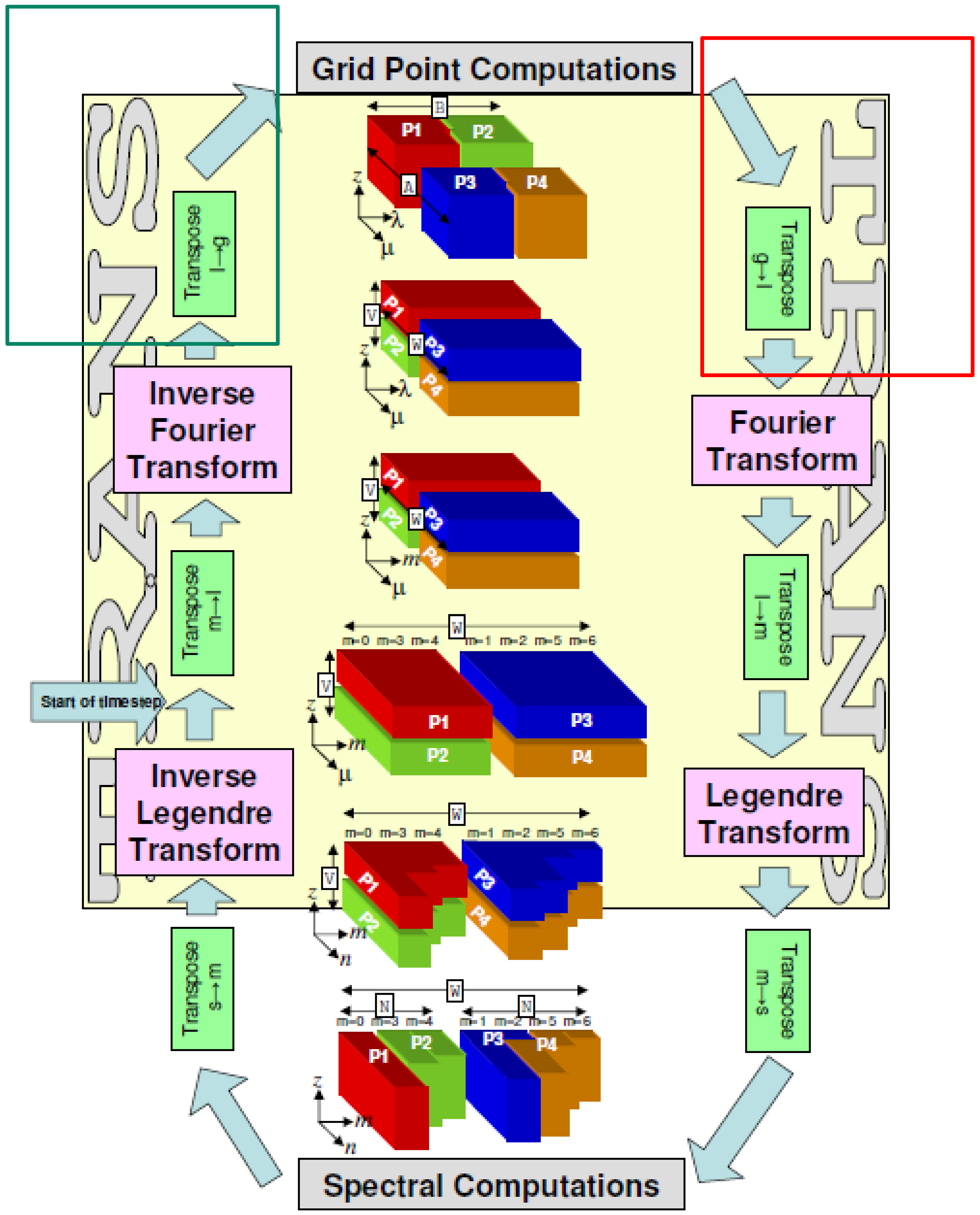
OPTIMIZATION APPROACH

IFS/ECTRANS: OVERVIEW OF ONGOING WORK

- Goal: utilize NVIDIA systems' state of the art capabilities to improve performance of ECTRANS/IFS
  - Phase 1: target UMR mechanism for combined gather/scatter with data transfer – Prep stage for FFT's
    - Use MPI derived datatypes
  - Phase 2: utilize DPU offload
    - Replace point-to-point communication with neighborhood collectives
  - Phase 3: change neighborhood collectives to allow for overlap of communication and computation
  - A side note: a code change suggestion (improve cache utilization)
- This presentation: Phase 1

CODE OVERVIEW

trltog_mod.F90
(unpacking)



trgtol_mod.F90
(packing)

NVIDIA

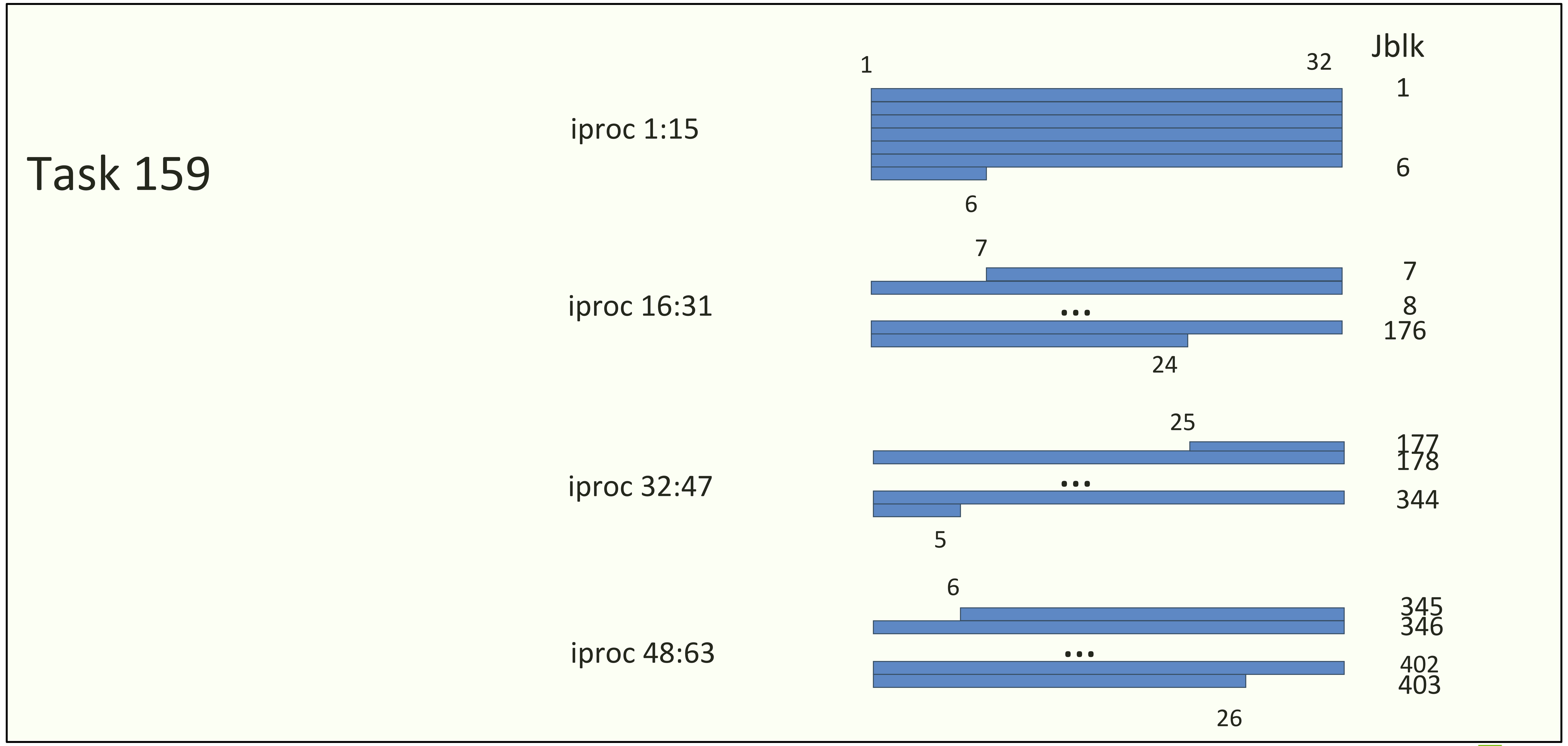
ORIGINAL CODE DATA STRUCTURES

Grid arrays and send buffer

- Grid-to-lattice transpose (trgtol): start with grid arrays, e.g. PGPUV

PGUV(32,	137,	2,	403)
ij block size	vert. levels	fields (U/V)	ij blocks

- Pack/restructure into a Send Buffer, combining several variables (PGP2,PGPUV,PGP3A/B)
- Send Buffer structure: (hor_ij, levels, fields) – need to coalesce 1st and 4th dimension of PGPUV and other arrays. This **gather** operation can be combined with data transfer through UMR.
- Complicated packing procedure:
  - Not entire ij plane is sent.
  - The layout is process dependent.



MPI DATATYPES CONCEPT

MPI Derived Datatypes: an elegant solution for packing/unpacking.

- Create a data descriptor, based on patterns of memory access
- Use (and reuse) the descriptor in MPI communication as the datatype argument.

Advantages of using MPI Datatypes:

- Define your data pattern once, then reuse it
- Less error prone
- Allows MPI implementations to provide streamlined solutions for packing/communication interface “under the hood”

MPI DATATYPES IN ECTRANS

1. Create a temporary datatype to coalesce 1st and 4th dimensions

```
call mpi_type_vector(Nblocks,dims(1),dims(1)*dims(2)*dims(3),
MPI_DOUBLE_PRECISION,type_tmp1,ierr)
```

2. Combine these temporaries to lay out vertical levels (2nd dimension):

```
call mpi_type_hvector(nlevels,1,nproma*8,type_tmp1,sendtype,ierr)
```

3. Commit the final datatype

```
call mpi_type_commit(sendtype,ierr)
```

4. Now we're ready to roll: send just one element of the new datatype:

```
call mpi_send(pgpuv(1,1,ifield,1),1,sendtype,dest,tag,mpi_comm_world,ierr)
```

Note 1: combining gather with sends is done by the MPI implementation behind the scenes.

Note 2: for incomplete blocks, record the first and last index, to be used when unpacking the receive buffer

ADDITIONAL CONSIDERATIONS

Memory bandwidth optimizations

- Instances of inefficient memory access identified
  - Changing the ordering of array indices multiple times back and forth
  - Extra array copies
- Removing the extra copies and transposes should be straightforward
  - Can be done independently from other work
- Expect to see improved performance at all scales

SUMMARY

- Several ways to improve ECTRANS performance have been identified
- Work ongoing on MPI Datatypes conversion. Expect the new version of the code to use our state-of-the-art hardware optimizations to achieve better performance.
- Future directions identified
  - Remove extra memory transposes and copies
  - Use collective communication where possible and overlap communication with computation
- Extending existing software components – ongoing effort
  - UCC - a collectives acceleration library
  - UCX – point-to-point acceleration library
  - Open MPI datatype support

NVIDIA®

SUMMARY OF PROPOSED CHANGES FOR PHASE 1

1. Replace zcombufs (the original send buffer combining all arrays) with sending the data directly from their source arrays using the derived datatype sendtype
2. Modify the receive buffer unpacking a bit to incorporate incomplete blocks information

New code flow:

- Add code in the initialization phase, setting up the bookkeeping and the datatypes
- Post receives for each non-zero array and each field (U/V), using mpi_irecv
- Send the data for each non-zero array and each field (U/V), using mpi_send and the sendtype derived datatype.
- Do self-copy (this piece is unchanged)
- Do a loop with mpi_waitany() to unpack receive buffers for each array and field, taking into account the incomplete blocks information

ADDITIONAL CONSIDERATIONS

Altering memory ordering

Currently a number of spectrum space data structures in ECTRANS/IFS have the following layout:

Ar(fields,horizontal plane index)

This implies inefficient use of cache while accessing the horizontal plane data, where the Fourier and Legendre transforms take place. To make these transforms more efficient, there is a memory copy with local transpose, interchanging the array indices to make the plane index first. This makes for an efficient Fourier/Legendre transform, however the memory copies themselves are (1) inefficient and (2) redundant.

Array name	Routine/module
ZCOMBUFR: (HorDim; Flds; Proc)	
PGLAT(Flds =27; HorDim)	TRGTOL
PREEL <--> ZFFT	EXEC_FFTW
FOUBUF_IN	FTDIR_CTL
FOUBUF	
PSIA, PAIA	LEDIR, PRFI2B
ZB	
ZCA	
POA1,POA2	
ZOA1,ZOA2	LTDIR
PSPSCALAR,PSPSC3A/B,PSCSC2,PSPVOR,PSPDIV	On output

ADDITIONAL CONSIDERATIONS, CONTD.

Altering memory ordering

- Proposed changes (Phase 1a)
  1. Keep the structure of stride-1 in space throughout the unpacking of receive buffer and Fourier transform. If needed, afterwards transpose the data structure to adapt to the rest of the code
     a. Or, if desired, change all the structures in the spectral space to the new format, thus eliminating the need for the memory copies altogether (Note: help from IFS experts is likely needed here)
  2. If and when we do the memory transpose, utilize the cache blocking method to make it more efficient (alternatively, use BLAS)
- These changes are mostly independent of the rest of the work (Datatype conversion). They can be done separately, and combined with a simple one line code change in receive buffer unpacking