MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

The RFM3GPU project

porting the REFMUL3 code to GPUs

T. Ribeiro[1]
F. da Silva[2]

[1]MPI für Plasmaphysik
[2]IPFN/IST U. Lisboa

EUROfusion

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

TÉCNICO LISBOA

Motivation

Port the REFMUL3 code to GPUs

- keep the CPU capabilities available
- minimise source code changes
- use directive-based target device offloading
- use OpenMP standard (version $\geq$ 4.5)

Outline

Outline

Model and numerics

Solve for Maxwell's curl equations + constitutive relation (current density)

$$\nabla \times \mathbf{H} = \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} + \sigma \mathbf{E} + \mathbf{J}$$

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t} - \sigma^{\star} \mathbf{H}$$

$$\frac{\partial \mathbf{J}}{\partial t} = \varepsilon_0 \omega_p^2 \mathbf{E} + \vec{\omega}_c \times \mathbf{J}$$

using the Finite Difference Time Domain (FDTD) method

REFMUL3

a 3D hybrid MPI/OpenMP full-wave code written in C

K.S. Yee, *IEEE Trans Antennas Propag* **14** (1966) 302
L. Xu and N. Yuan *IEEE Antennas Wirel Propag Lett* **5** (2006) 335

F. Silva, S. Heuraux and T. Ribeiro *Proc. IRW13* (2017)
F. Silva, S. Heuraux, T. Ribeiro et al. *Fusion Eng Des* (2023) submitted

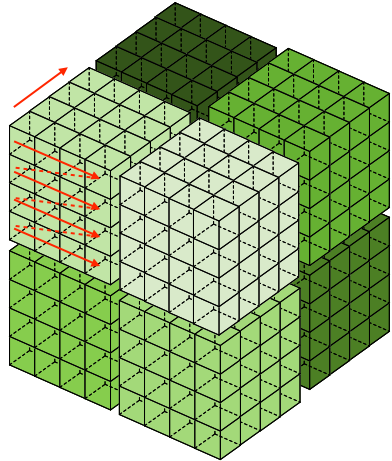
Domain decomposition

MPI parallelisation

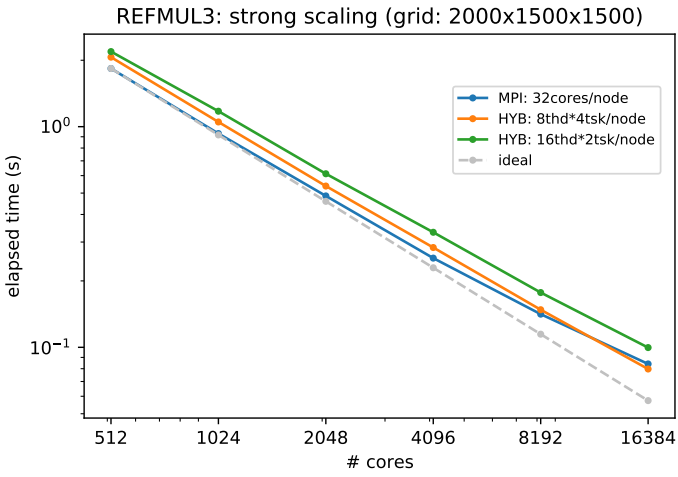
Cartesian mapping for a full 3D domain decomposition

OpenMP threaded workshare

parallel regions along the slowest varying index (outermost loop)



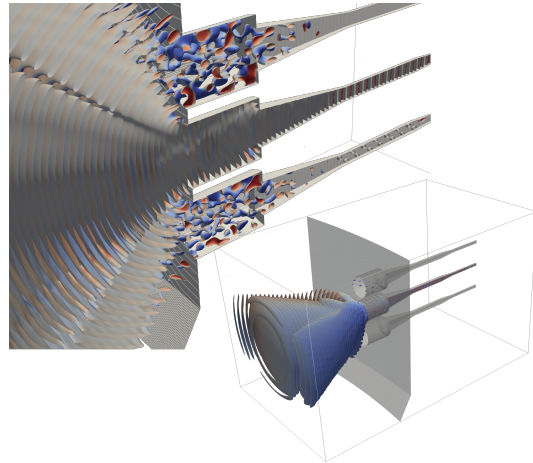
Strong scaling on Marconi (Intel Skylake)



REFMUL3 **applications**

Code has been used for

- interpret experimental microwave reflectometry data
- benchmark 2D simulation results
- design/engineering studies for ITER, DDT and DEMO
- studies of other microwave diagnostics also possible, e.g. interferometry, polarimetry, intensity refractometry, ...



Outline

Target device offload

Several possibilities available

- language-based: CUDA (Nvidia), HIP (AMD), ...
- directive-based: OpenACC, OpenMP
- framework-based: Kokkos, SYCL, oneAPI, RAJA, ...

Target device offload

Several possibilities available

- language-based: CUDA (Nvidia), HIP (AMD), ...
- directive-based: OpenACC, OpenMP
- framework-based: Kokkos, SYCL, oneAPI, RAJA, ...

Decision based on

- conceptual and syntactic familiarity
- performance portability

Single GPU

Initial steps

- port each individual hotspot function
- move data locally between host $\leftrightarrow$ device
- worry about performance later

Important

verify the code results (correctness)

Simplified code snippet of a hotspot function

```
1  void calcEyxField(struct FLDCUBE Eyx, struct FLDCUBE Hzx, struct FLDCUBE Hzy,
2                    struct FLDCUBE Jyx, struct FLDCUBE ElCx, struct PHYCNST phycnst, struct UNITS units)
3  {
4    (...)
5    #pragma omp target data map(tofrom:Eyx.f[0:szEyx]) map(to:,Hzx.f[0:szHzx],Hzy.f[0:szHzy],...)
6    #pragma omp target teams distribute parallel for collapse(3)
         private(i,j,k,A,invB,auxAB)
         firstprivate(c,e,fq,i0,i9,j0,j9,k0,k9,Eyx_i0,Eyx_j0,Eyx_k0,Eyx_ni,Eyx_nj, ...)
7    for(k=k0; k<=k9; k++) {
8      for(j=j0; j<=j9; j++) {
9        for(i=i0; i<=i9; i++) {
10         auxAB=f(ElCx,0,0,i)*fq;
11         A=1-auxAB;
12         invB=1/(1+auxAB);
13         f(Eyx,k,j,i)=invB*( A*f(Eyx,k,j,i)
14                             - c*(f(Hzx,k,j,i)-f(Hzx,k,j,i-1) + f(Hzy,k,j,i)-f(Hzy,k,j,i-1))
15                             - e*f(Jyx,k,j,i)
16                            );
17   }}}
18 }
```

Simplified code snippet of a hotspot function

```
1  void calcEyxField(struct FLDCUBE Eyx, struct FLDCUBE Hzx, struct FLDCUBE Hzy,
2                    struct FLDCUBE Jyx, struct FLDCUBE ElCx, struct PHYCNST phycnst, struct UNITS units)
3  {
4    (...)
5    #pragma omp target data map(tofrom:Eyx.f[0:szEyx]) map(to:,Hzx.f[0:szHzx],Hzy.f[0:szHzy],...)
6    #pragma omp target teams distribute parallel for collapse(3)
         private(i,j,k,A,invB,auxAB)
         firstprivate(c,e,fq,i0,i9,j0,j9,k0,k9,Eyx_i0,Eyx_j0,Eyx_k0,Eyx_ni,Eyx_nj, ...)
7    for(k=k0; k<=k9; k++) {
8      for(j=j0; j<=j9; j++) {
9        for(i=i0; i<=i9; i++) {
10         auxAB=f(ElCx,0,0,i)*fq;
11         A=1-auxAB;
12         invB=1/(1+auxAB);
13         f(Eyx,k,j,i)=invB*( A*f(Eyx,k,j,i)
14                             - c*(f(Hzx,k,j,i)-f(Hzx,k,j,i-1) + f(Hzy,k,j,i)-f(Hzy,k,j,i-1))
15                             - e*f(Jyx,k,j,i)
16                           );
17   }}}
18 }
```

Observation

performance is poor because of excessive data movement

Performance optimisation

Minimise host ↔ device data movement

- unstructured data regions

```
1  /* Host -> GPU data exchange */
2  #pragma omp target enter data map(to: <all field variables>)
3  /*---------------------------------------------------------------------------*/
4  /*                              Main cycle                                   */
5  /*---------------------------------------------------------------------------*/
6    for(cnst.n=0; cnst.n<cnst.npts; cnst.n++) {
7
8      <main REFMUL3 algorithm here>
9
10     /* additional data transfer needed inside the main cycle
11      * (e.g. I/O operations or MPI communication) */
12 #    pragma omp target update (to/from <field variables>)
13
14     <main REFMUL3 algorithm here>
15
16   } /* Closes main cycle */
17 /*---------------------------------------------------------------------------*/
18 /*                           Ends main cycle                                 */
19 /*---------------------------------------------------------------------------*/
20 /* Host <- GPU data exchange */
21 #pragma omp target exit data map(from: <all field variables>)
```

Revisit previous hotspot function

Preprocessor guard macros allow different compilation modes

```
1  void calcEyxField(struct FLDCUBE Eyx, struct FLDCUBE Hzx, struct FLDCUBE Hzy,
2                    struct FLDCUBE Jyx, struct FLDCUBE ElCx, struct PHYCNST phycnst, struct UNITS units)
3  {
4    (...)
5    _OMPTHD_(omp parallel for default(none) shared(Eyx)
6             private(i,j,k,A,invB,auxAB)                                    \
7             firstprivate(c,e,fq,i0,i9,j0,j9,k0,k9,Hzx,Hzy,... ))
8    _OMPTGT_(omp target teams distribute parallel for collapse(3)                                  \
9             private(i,j,k,A,invB,auxAB)                                                           \
10            firstprivate(c,e,fq,i0,i9,j0,j9,k0,k9,Eyx_i0,Eyx_j0,Eyx_k0,Eyx_ni,Eyx_nj, ...) \
11            device(devID))
12   _ACCTGT_(acc parallel loop independent)
13   for(k=k0; k<=k9; k++) {
14     for(j=j0; j<=j9; j++) {
15       for(i=i0; i<=i9; i++) {
16         auxAB=f(ElCx,0,0,i)*fq;
17         A=1-auxAB;
18         invB=1/(1+auxAB);
19         f(Eyx,k,j,i)=invB*( A*f(Eyx,k,j,i)
20                             - c*(f(Hzx,k,j,i)-f(Hzx,k,j,i-1) + f(Hzy,k,j,i)-f(Hzy,k,j,i-1))
21                             - e*f(Jyx,k,j,i)
22                           );
23    }}}
24 }
```

Preprocessor macros for OpenMP & OpenACC directives

`_Pragma` operator (C99) allows preproc. macro within a pragma directive

```
#ifdef _OPENMP
/* OpenMP offloading */
# ifdef _OL_OMP_
#   define _OMPTGT_(x) _Pragma(#x)
/* OpenMP threading */
# else
#   define _OMPTHD_(x) _Pragma(#x)
# endif
#else
/* OpenACC offloading */
# ifdef _OL_ACC_
#   define _ACCTGT_(x) _Pragma(#x)
# endif
#endif
/* defaults if not defined before: all defined to nothing */
#ifndef _ACCTGT_
# define _ACCTGT_(x)
#endif
#ifndef _OMPTGT_
# define _OMPTGT_(x)
#endif
#ifndef _OMPTHD_
# define _OMPTHD_(x)
#endif
```

Multiple GPUs

Use available MPI parallelisation of REFMUL3

- MPI communicator splitting to identify GPUs on each node
- intrinsic MPI type MPI_COMM_TYPE_SHARED

```
1  /* calculate deviceID using modulo and MPI_COMM_TYPE_SHARED type */
2  #ifdef _OL_OMP_
3    /* how many target devices per node */
4    Par->Ndevices = omp_get_num_devices();
5
6    /* sub-communicator within each compute node */
7    MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, &Par->comm_intra_node);
8
9    /* own node-local rank */
10   MPI_Comm_rank(Par->comm_intra_node, &Par->i_MyIntraNodeRank);
11
12   /* how many ranks per node */
13   MPI_Comm_size(Par->comm_intra_node, &Par->i_NIntraNodeRanks);
14   Par->deviceID = Par->i_MyIntraNodeRank % Par->Ndevices;
15 # endif
```

- Par->i_NIntraNodeRanks $\leq$ Par->Ndevices

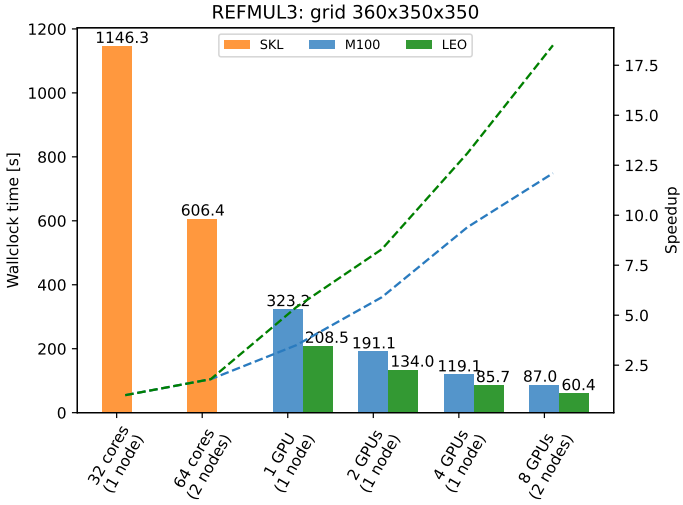
Multiple GPUs

MPI communication + host ↔ device synchronisation

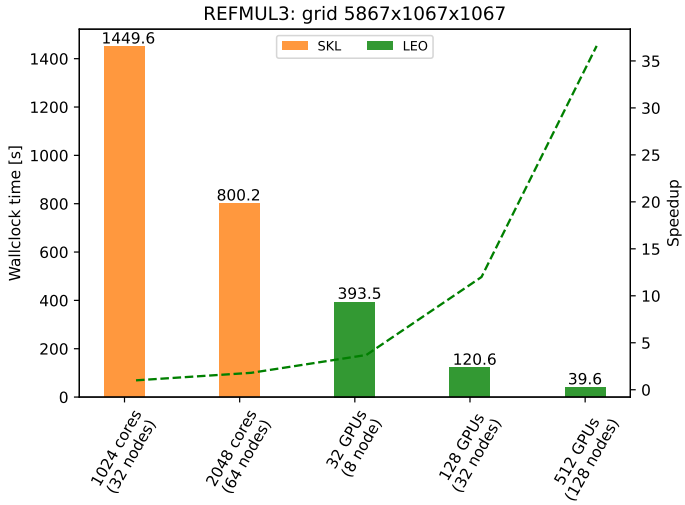
```
1  /* NORTHWARD neighbours exchange:
2   * send NORTH and receive from SOUTH (j->j-1) */
3
4  /* buffer size (2D) */
5  iCount = iNXp * iNZp;
6
7  /* GPUs: update communication buffers from target device */
8  if (Par.i_North != MPI_PROC_NULL) {
9    _OMPTGT_(omp target update from(fldSBufNorth[0:iCount]) device(devID))}
10
11 /* send buffer up -> send from j to j-1 */
12 MPI_Isend(fld.SBufNorth, iCount, MPI_DOUBLE, Par.i_North, 7, Par.comm_cart, &reqsNS[0]);
13 /* recv buffer from below -> recv at j-1 from j */
14 MPI_Irecv(fld.RBufSouth, iCount, MPI_DOUBLE, Par.i_South, 7, Par.comm_cart, &reqsNS[1]);
15 MPI_Waitall (2, reqsNS, statusNS);
16
17 /* GPUs: update communication buffers to target device */
18 if (Par.i_South != MPI_PROC_NULL) {
19   _OMPTGT_(omp target update to(fldRBufSouth[0:iCount]) device(devID))}
```

- addition of `device(devID)` to OpenMP directives

Results, small test case



Results, large production case



Nvidia compiler bug found (help from Thomas Hayward-Schneider)

```
1  #include <stdlib.h>
2  int main(int argc, char** argv) {
3    int ny = 1024; int nx = 1024;
4    int iy, ix;
5    double* __restrict__ Aarr=(double*) malloc(nx*ny*sizeof(double));  // allocate the memory
6    // exchange data with device
7  # pragma omp target data map(tofrom: Aarr[0:nx*ny])
8    {
9      // update @ device
10     // line 11 crashes @ compile time || workaround line 12
11 #   pragma omp target teams distribute parallel for firstprivate(nx,ny) private(ix,iy) collapse(2)
12 //workaround//
13 //# pragma omp target teams distribute parallel for private(ix,iy) collapse(2)
14     for( iy = 0; iy < ny; iy++)
15       for( ix = 0; ix < nx; ix++ )
16           Aarr[iy*nx+ix] += 11./ny/nx;
17   }
18   // release memory
19   free(Aarr);
20 }
```

Internal bug (nvbug 4101016) filled by Markus Hrywniak via MPCDF

fix provided by Nvidia in `nvhpcsdk` v.23.7

Outline

Summary & outlook

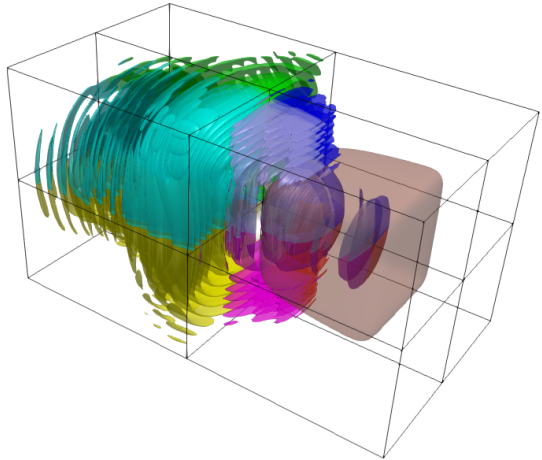
REFMUL3 porting to GPU

- implemented using OpenMP offload directives (+ GitLab CI)
- good performance and scalability achieved
- enables larger cases & access to additional HPC facilities
- Nvidia compiler bug found, meanwhile corrected

Outlook

- test code in different HPC machines (& compilers)
- `mapper` modifier (OpenMP ver. $\geq$ 5.0): simplify source code
- RDMA GPU-aware MPI: avoid explicit host-device data movement
- implement OpenACC offloading

Backup slides



MPI communication

- find good 3D domain decomposition
  `MPI_Dims_create(NRanks,3,dims);`
- create Cartesian virtual topology
  `MPI_Cart_create(comm_world,3,dims,periods,0,&comm_cart);`
- get Cartesian virtual topology local x, y, z coordinates (`MyRankXYZ[3]`)
  `MPI_Cart_coords(comm_cart,MyRank,3,MyRankXYZ);`
- get neighbors ranks (e.g. y-direction)
  `MPI_Cart_shift(comm_cart,DirectionY,1,&North,&South);`
- asynchronous point-to-point communication (sequencially in dims)
  `MPI_Isend`, `MPI_Irecv` & `MPI_Waitall`

Memory allocation and access

Sub-domains of the global 3D domain $N_i * N_j * N_k$

- indices (global) of first & last nodes in each dimension $m \in \{i, j, k\}$

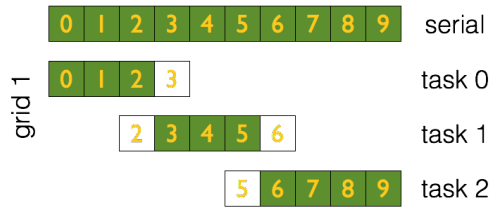
  $F_m = (N_m * MyRank_m)/NRanks_m + (BndyF_m - w)$
  $L_m = [N_m * (MyRank_m + 1)]/NRanks_m - BndyL_m$

- $w = 1$ ghost-cell/face, except at boundaries where $w = 0$
- size in each dimension (w/ ghost-cells): $NPar_m = L_m - F_m + 1$
- local memory allocated linearly: $NPar_i * NPar_j * NPar_k$
- local access using global index space: `f[ind(k,j,i)]`
  with $i \in \{0,...,N_i-1\}$, $j \in \{0,...,N_j-1\}$ and $k \in \{0,..,N_k-1\}$
- macro to map local and global index spaces

```
#define ind(k,j,i) (k-Fk)*NParj*NPari+(j-Fj)*NPari+(i-Fi)
```

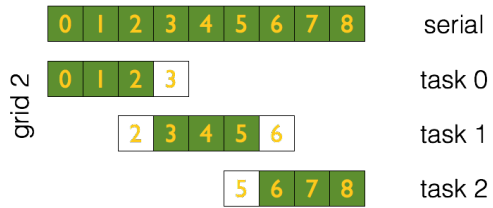
Staggered grids: 1D example

		global domain	sub-domain		
		serial	rank 0	rank 1	rank 2
$grid_1$	`NPar`	10	4	5	5
	`F:L`	0:9	0:3	2:6	5:9
$grid_2$	`NPar`	9	4	5	4
	`F:L`	0:8	0:3	2:6	5:8



Staggered grids: 1D example

		global domain	sub-domain		
		serial	rank 0	rank 1	rank 2
$grid_1$	`NPar`	10	4	5	5
	`F:L`	0:9	0:3	2:6	5:9
$grid_2$	`NPar`	9	4	5	4
	`F:L`	0:8	0:3	2:6	5:8



Parallel I/O in REFMUL3

Standard I/O operations

- multi-file: one HDF5 and/or VTK file / MPI rank
- single-file: parallel HDF5 library
- global metadata descriptor (XDMF/PVTI, respectively)
- data can be directly visualised in ParaView

Checkpoint & restart (single-file parallel HDF5)

- independence from domain decomposition
- simplicity handling the data
- seldom I/O, impact on scaling performance not critical

C-structures and local pointers

Electromagnetic fields stored in C-structures

- metadata + data
- not possible to map such data types (OpenMP v4.5)

```
1  struct FLDCUBE {
2    (...)
3    /* global grid-count in each direction (excluding ghost-cells) */
4    int i_kZLays;  /* global # grid-cells in z */
5    int i_jYRows;  /* global # grid-cells in y */
6    int i_iXCols;  /* global # grid-cells in x */
7    /* first and last grid-index of the sub-domains in each direction */
8    int i_kZLayFirst; i_kZLayLast;
9    int i_jYRowFirst; i_jYRowLast;
10   int i_iXColFirst; i_iXColLast;
11   /* local grid-count in each direction (including ghost-cells) */
12   int i_kZLaysPar;  /* local # grid-cells in x */
13   int i_jYRowsPar;  /* local # grid-cells in y */
14   int i_iXColsPar;  /* local # grid-cells in z */
15   /* pointers to the allocated memory for the domain */
16   myFloatType   *__restrict__ f;
17 };
```

C-structures and local pointers

Local pointers to access mapped field data on device

```
1  void calcConvIR(struct FLDCUBE ES, struct UTS uts, ...)
2  {
3    /* local pointers */
4    double* __restrict__ ESf = ES.f;
5
6    /* needed array metadata: bounds and sizes */
7    int ES_i0 = ES.i_iXColFirst; int ES_ni = ES.i_iXColsPar;
8    int ES_j0 = ES.i_jYRowFirst; int ES_nj = ES.i_jYRowsPar; int ES_k0 = ES.i_kZLayFirst;
9
10   _OMPTHD_(omp parallel for default(none)                    \
11            shared(ESf)                                      \
12            private(j,k)                                     \
13            firstprivate(uts,dz0,dz9,dy0,dy9,                \
14                         ES_i0,ES_j0,ES_k0,ES_ni,ES_nj))
15   _OMPTGT_(omp target teams distribute parallel for collapse(2) \
16            private(j,k)                                     \
17            firstprivate(uts,dz0,dz9,dy0,dy9,                \
18                         ES_i0,ES_j0,ES_k0,ES_ni,ES_nj))
19   _ACCTGT_(acc parallel loop independent)
20   for(k=dz0; k<=dz9; k++)
21     for(j=dy0; j<=dy9; j++) {
22       lf(ES,k,j,0)=uts.Fs;
23     }
```