Improved load balancing and non-blocking communications for maximal efficiency at high #core

Document Title: Technical Report

Authorship: Adrian Jackson, Colin Roach, David Dickenson

Date: 1st May 2016
Version: 1.0

Table of Contents

1 Introduction

Modelling plasma turbulence in magnetically confined fusion (MCF) devices is a challenging task because the simulations must be able to resolve plasma processes that span space from the electron Larmor radius ρe (~10-5m) to the device minor radius a (~2m), and span time from the shortest turbulent eddy turnover time (~10^{-8}s) to the energy confinement time (~1s). The gyrokinetic approach provides an efficient framework to solve for turbulent fluctuations with frequencies less than the ion cyclotron frequency, and this model captures the turbulence that is responsible for plasma transport.

GS2 is an open source gyrokinetic simulation code that solves the gyrokinetic system of equations for the evolution of the perturbed distribution function, g, and the electromagnetic fields, Φ. This system consists of the gyrokinetic equation governing the evolution of g, and Maxwell's equations for Φ.

GS2 is written in Fortran, parallelised with MPI, and has been demonstrated to scale efficiently to O(1,000) cores for typical problems, greatly helped by recent successful efforts to that have improved both serial performance and scaling efficiency, and reduce runtimes by up to a factor 20 for a typical case at high numbers of cores.

However, the scaling efficiency of the current code still drops significantly at large core counts (typically >4096), and additional improvements are required to improve further the prospects for more realistic and detailed simulations at higher resolutions.

Each time step in GS2 involves four main operations:

- N: non-linear terms in the advance of g
- L: collisionless advance of g
- C: impact of collisions on g
- F: field update

with a full simulation time step consists of the operations "NLCFLC". The work in this project focused on improving the scaling of F, as profiling studies demonstrate that F currently limits scaling performance at high core counts.

F consists of two steps (carried out over independent domains): velocity space integrations of g, and multiplication of the vector result by the "response matrix" determined at initialisation. In recent GS2 upgrades the field routines were rewritten to allow much more efficient velocity space integration using sub-communicators.

Whilst these changes have improved performance, attempts to minimise communication have introduced a load imbalance that significantly degrades the scaling of F above a certain core count. Poor F scaling is the main bottleneck that degrades the efficiency of our test case at around 4,096 cores.

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

2 Simulation Functionality

GS2 is configurable and can run a range of different types of simulation. We have already outlined that there are 3 main operations (aside from the field update) that can be undertaken:

- Linear (collisionless advance of g)
- Non-linear advance of g
- Impact of collisions (of particles) on g

Simulations can run in a purely linear node, or linear with any combination of non-linear and collisions. Enabling non-linear and collision functionality significantly increases computational costs, with collisions in particular being very costly. However, none of these options impact the computational cost of the field update in GS2, so they should not directly impact the areas of the code we are aiming to optimise with this project.

There are three different fields that can used in used in simulations:

- Φ: The basic electrostatic potential
- $A \parallel$: The parallel vector potential
- $B \parallel$: The perturbed parallel magnetic field

Φ is required, but $A \parallel$ and $B \parallel$ are optional, and can be enabled or disabled for any given simulation. Each field requires similar amounts of work, so enabling all 3 fields will make the field calculation approximately 3 times as long.

GS2 lets users specify a data layout for the g data object to provide some guidance on how to parallelise the array storing the perturbed distribution functions for all the plasma species. GS2 uses 5 different indices, denoted as follows by the characters x, y, l, e and s:

- x: Fourier wavenumber in the X direction in space
- y: Fourier wavenumber in the Y direction in space
- l: Pitch angle
- e: Energy
- s: Number of particle species

GS2 supports six different data layouts; $xyles, yxles, lyxes, yxels, lxyes, lexys$. The layout is chosen at run time by the user (through the input parameter file) and controls how the data domain in GS2 is distributed across processes by specifying the order in which individual dimensions in the data domain are distributed (split up). For instance, the $xyles$ layout will decompose s first and x last (depending on the number of processes used), whereas the $lexys$ layout will decompose s first and l last.

GS2 undertakes an initialisation process prior to simulation, where the initial fields are calculated and data decomposition functionality is setup. This initialisation step can also take a significant amount of time, and grows exponentially with the number of fields used (as significant numbers of field calculations have to be performed to calculate the initial fields). This means that any optimisation of the field simulation functionality will also improve the performance of the initialisation process.

2.1 ingen

For any given GS2 input file (which specifies the simulation to be carried out, including the domain decomposition layout) the program `ingen` provides a list of recommended process counts (or "sweet spots") for the GS2 simulation. These recommendations are computed from the data in the input file, and aim to split the data domain as evenly as possible to achieve good "load-balancing". The primary list of recommended process counts is based on the main data layout, `g_lo` (used for the linear parts of the simulations). `ingen` also provides lists of process counts that are suitable for the nonlinear parts of the calculations (referred to as `xxf_lo` and `yxf_lo` process counts) which may differ from the process counts recommended for `g_lo`.

3 Field Calculation in GS2

The electromagnetic fields in GS2, Φ(θ,x,y), depend only on 3-D real space (not the 5-D space of g, and are determined using species summed velocity space integrals of g. Note that in GS2 Φ is not distributed and is known locally on all processes (i.e. the full Φ is on all processes).

The latest field routines in GS2 are structured so that each processor need only contribute to field calculations for the specific x and y values for which it has assigned in the decomposition of g, using MPI allreduce over sub-communicators to perform the integrations.

Without further communication each processor only calculates Φ for the portion of the x and y domain which is held locally. This restricts how the field solve may be decomposed amongst processors when x or y are distributed. A further complication arises from the standard parallel boundary condition in these flux tube simulations where different x's are coupled at the ends of the θ grid. The coupling pattern depends on both y and other input parameters.

GS2 has a concept of *cells*, which corresponds to a specific x and y part of Φ, and the of a *supercells*, which are the set of cells coupled by the parallel boundary condition of the flux tube.

The field solve then consists of two operations: firstly a velocity space integration of g for each field being used in the simulation; followed by a matrix-vector product to update each field for each independent supercell. The velocity space integration calculates the vector that is used in the matrix-vector product, with the matrix (known as the field response matrix) constructed during the initialisation of GS2 (using the field calculation code).

The size of the supercell depends on the number of connected cells, which varies with y. This means the work associated with matrix operations varies substantially between supercells. In particular the size of the field response matrix is given by (x *nfield* x_in_supercell)^2 where x_in_supercell can typically vary from 1 to O(100) for current problems.

Therefore, it can be seen that if x and y are distributed by the parallelisation of g there could be significant load balance issues associated with the field calculations, as the amount of work in the matrix operations will vary with the specific x and y portions a process has.

However, the current code does not necessarily fully decompose the matrix-vector operation across all processes in the supercell, as communication costs increase with more participants. Currently, the matrix-vector supercell calculation decomposition considers each supercell in turn and:

1. Calculates a balanced blocksize for all the processes in the supercell.
2. Implements an actual blocksize which is Max(balanced blocksize, user_defined_value), which can introduce a load imbalance to reduce communication costs.
3. Records allocated work, and assigns blocks of work to available processes, starting with those processes with the least amount of work already assigned.

This current scheme provides a coarse way to tune the relative load imbalance and communication costs associated with the field solve.

4 Initial performance

Prior to any optimisation work it's important to understand the current performance of the code. In this section we aim to capture the performance and scaling of GS2 on a representative simulation, and evaluate where performance problems exist. The simulation parameters (GS2 input file) used to collect the performance results presented in this report is included in Appendix A at the end of the report.

We have performed simulations using 3 of the 6 GS2 layout types; $xyles, yxles,$ and $lexys$; as these are the data layouts commonly for simulations on systems such as ARCHER. We have also performed simulations using 1, 2, and 3 fields active in the simulations to evaluate the impact of varying fields on GS2.

4.1 Process Counts

The following process counts are suggested as sweet spots for the layouts we have chosen to profile:

- $lexys$: 448, 576, 1344, 4032, 8064
- $xyles$: 512, 1024, 1536, 2048, 3072, 3584, 4096, 8192
- $yxles$: 512, 1024, 1536, 2048, 3072, 3584, 4096, 8192

We have used a subset of the above process counts for our scaling/performance benchmarking.

4.2 Linear Performance

The first set of results we collected on ARCHER were only undertaking linear simulations (i.e. using the GS2 input parameter `nonlinear_mode='off'`). Figures 1 and 2 show the runtime of the initialisation and advance (main simulation calculation) parts of GS2 for varying numbers of processes on ARCHER.

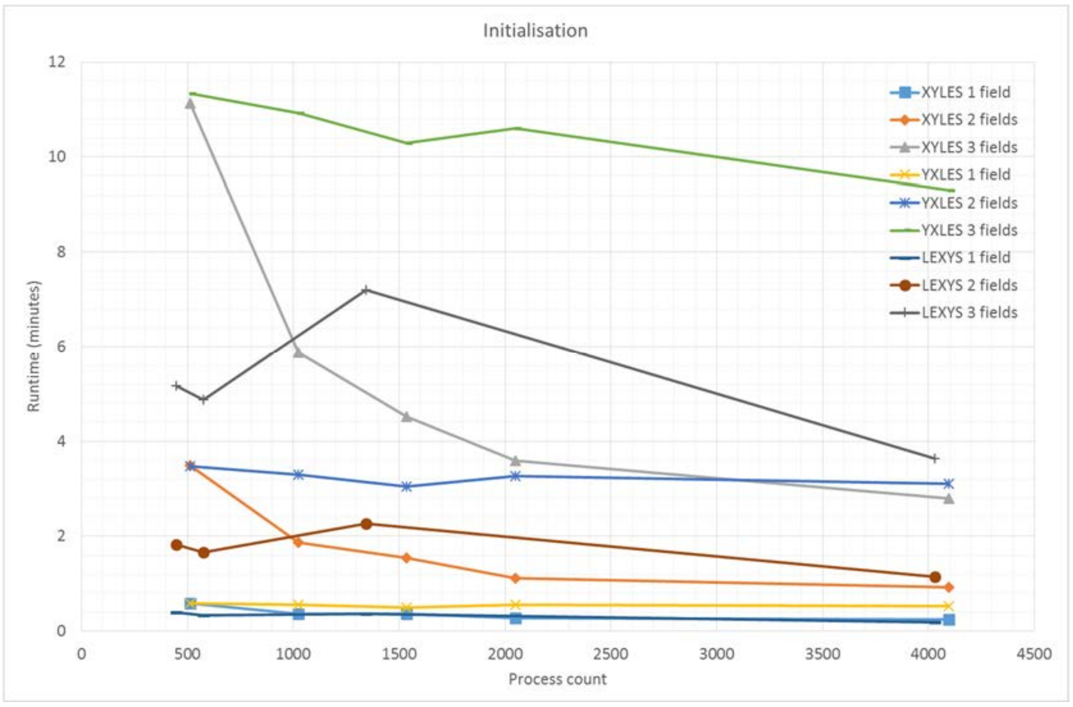

Figure 1: Initialisation time for linear simulations

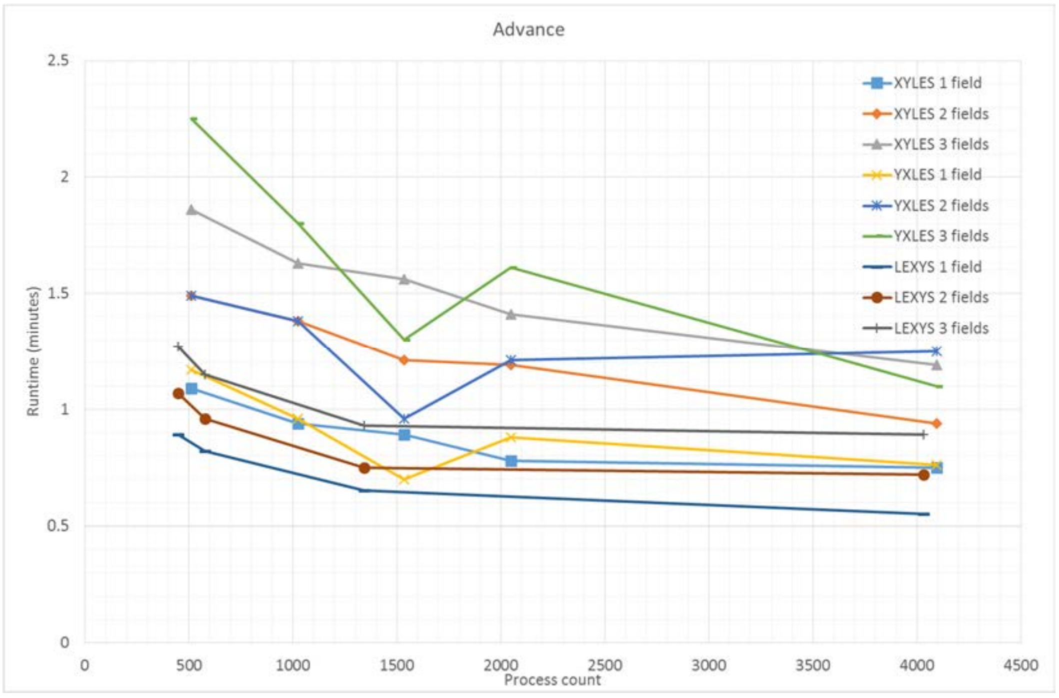

Figure 2: Advance time for linear simulation

4.2.1 Linear initialisation time

Figures 3-5 show the initialisation time for the different numbers of fields separately, to enable comparing data layouts in more detail.

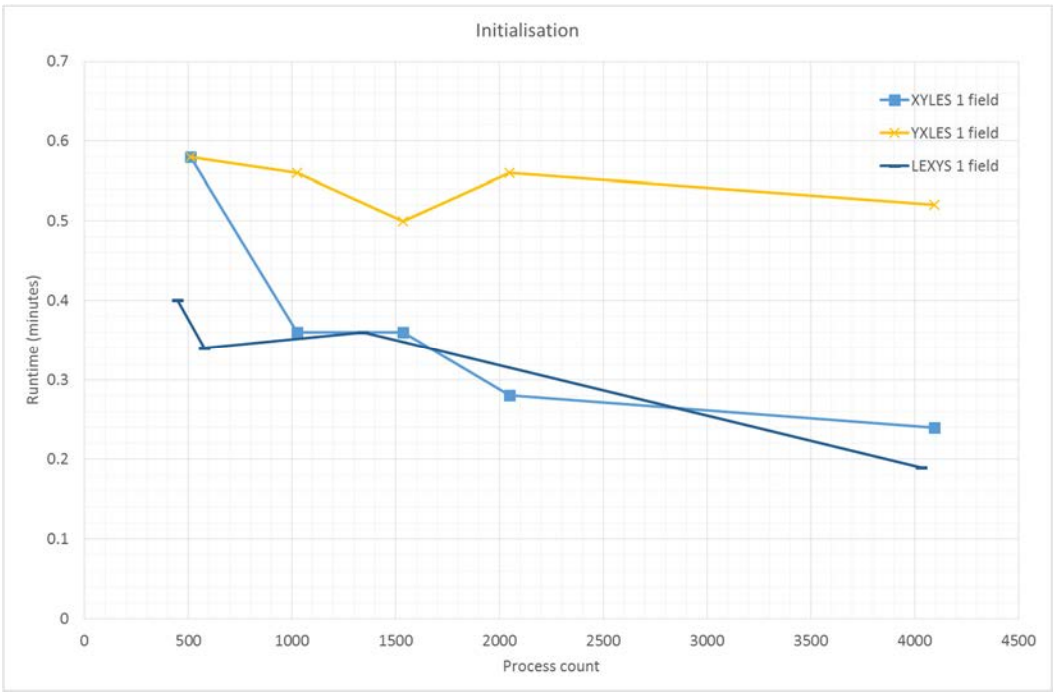

Figure 3: Initialisation time for a single field

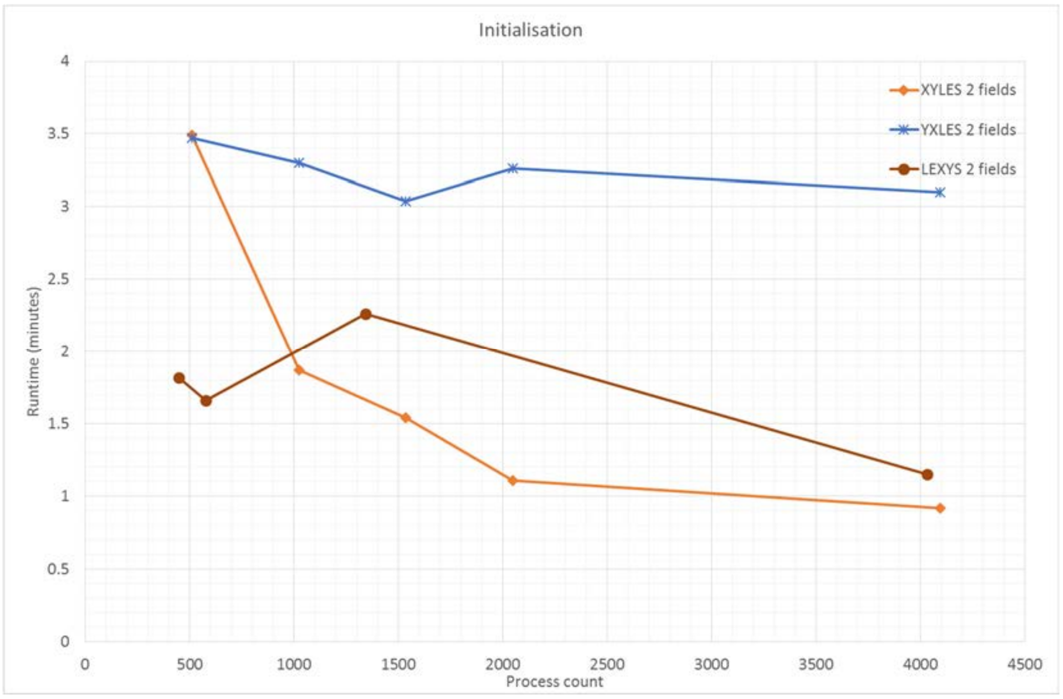

Figure 4: Initialisation time for two fields

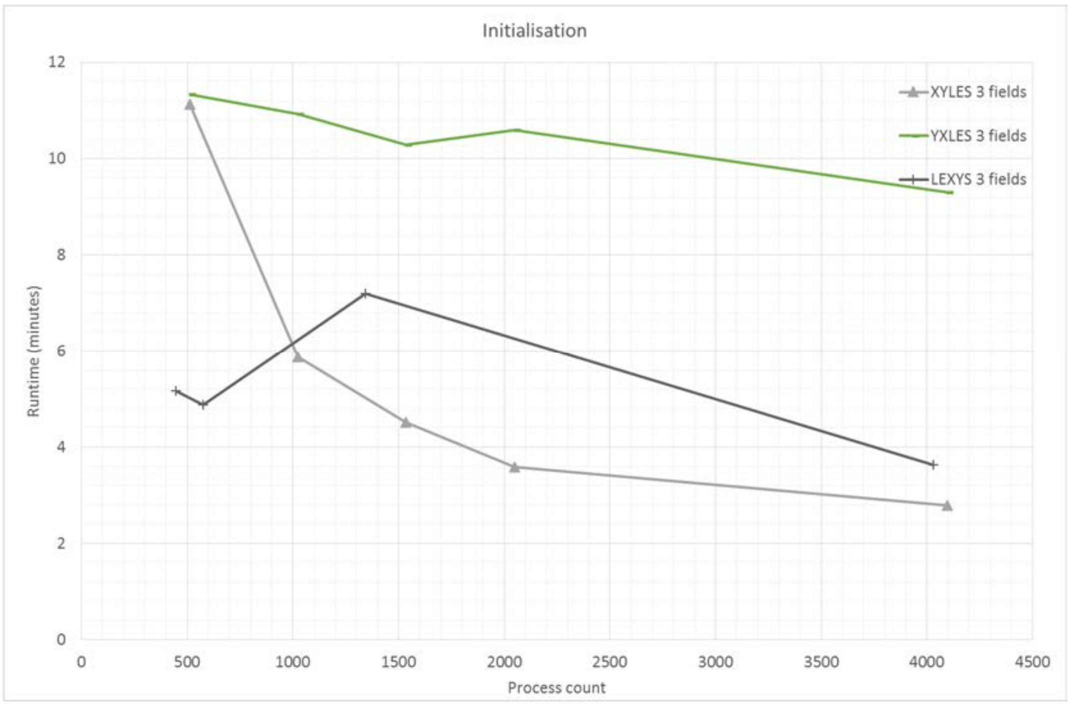

Figure 5: Initialisation time for three fields

Looking at the graphs for initialisation time we can see that the $yxles$ layout initialisation takes significantly longer than the other two layouts, and does not really scale as more processes are used. $xyles$ is similarly costly at low process counts, but scales better and can give the quickest initialisation at high process counts, and $lexys$ generally is the quickest to initialise, but similar to $yxles$ it does not scale very well when using more processes.

We can also see that moving for a single field to three fields can significantly increase the run time (and therefore the computational cost) of the initialisation, from of the order of 0.2-0.6 minutes for 1 field to around 3-11 minutes for 3 fields.

4.2.2 Linear advance time

Figures 6-8 show the advance time for the different numbers of fields separately, to enable comparing data layouts in more detail.

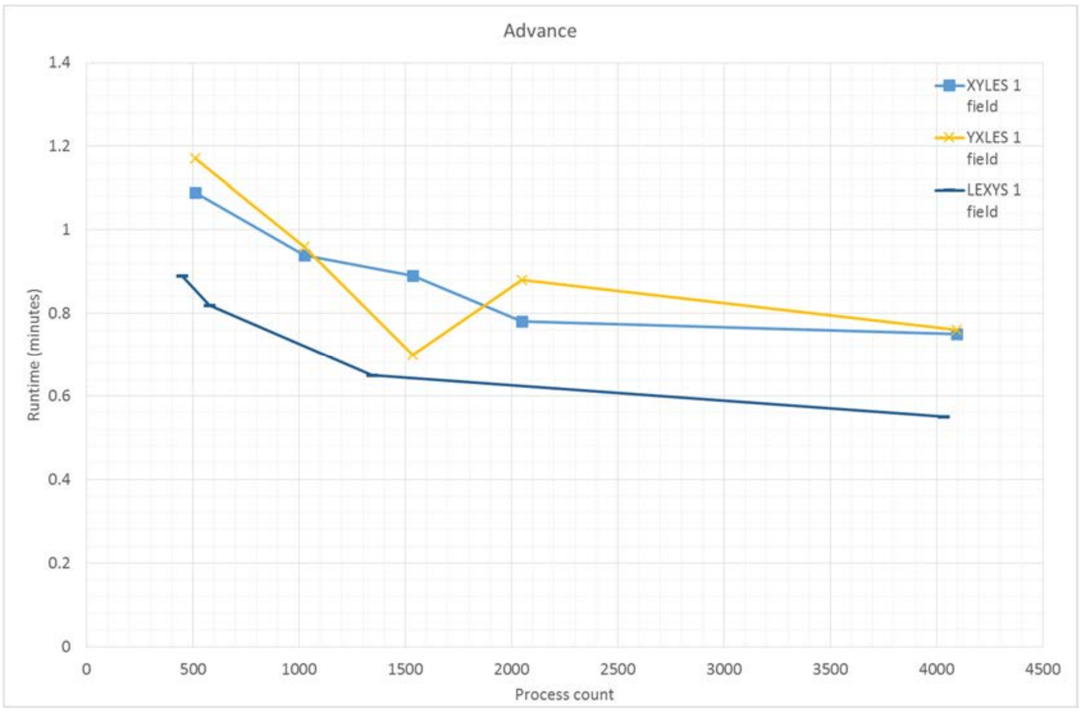

Figure 6: Advance time for 1 field

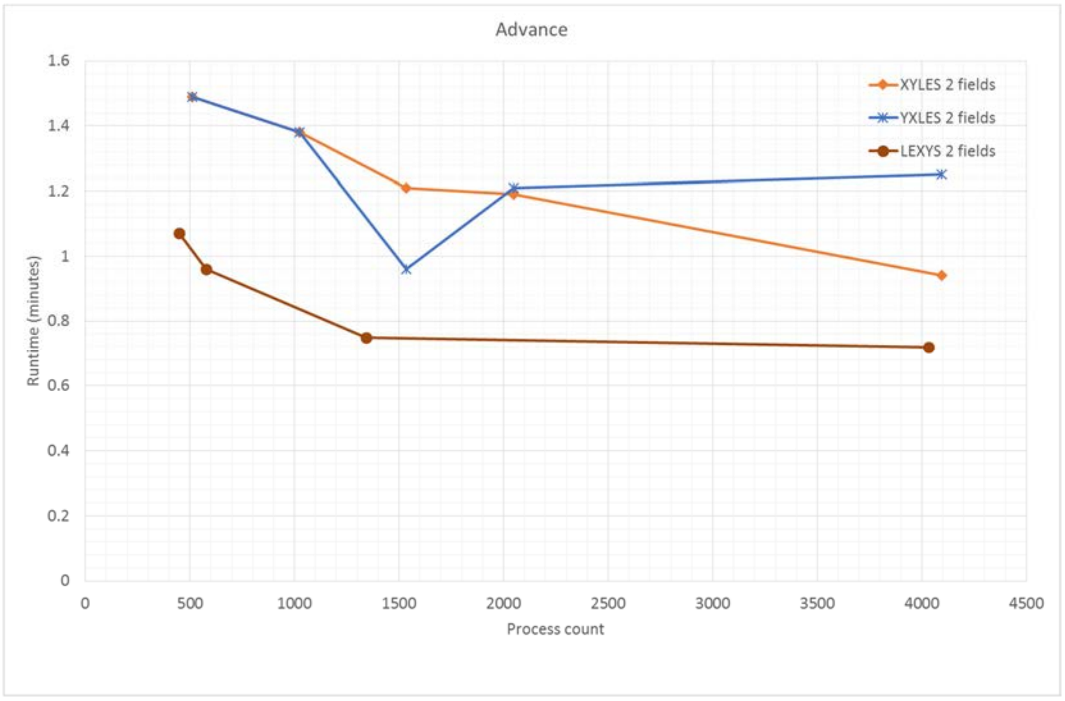

Figure 7: Advance time for two fields

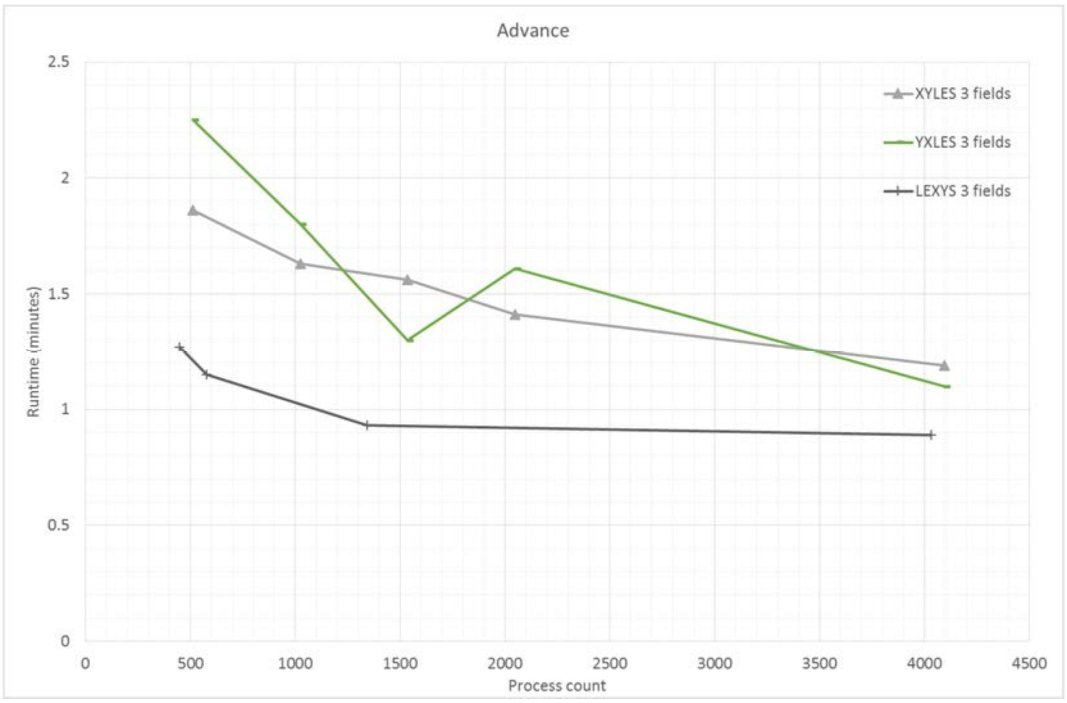

Figure 8: Advance time for three fields

Looking at the advance time, we can see that as with the initialisation time $lexys$ provides the best performance, although it does not scale well. $xyles$ and $yxles$ have similar performance profiles, scaling better than $lexys$ but never achieving its absolute performance.

For all the layouts we can see that scaling becomes a problem when using large process counts. If we profile GS2, using CrayPat, then we can gain some more detailed understanding of what parts of the code are being used, what parts are scaling well, and what the level of MPI communication is on different process counts.

4.2.3 Linear profiling data

Investigations of the detailed performance of GS2 on the linear test case was undertaken with CrayPat to identify the subroutines consuming the more computational time for a given run, and the amount of MPI communications performed for the same run.

We profiled GS2 on the small and large process counts used for the performance benchmarks previously discussed, specifically:

- $lexys$: 448 and 4032 processes
- $xyles$: 512 and 4096 processes
- $yxles$: 512 and 4096 processes

Profiling result for $yxles$:

1 field, 512 processes:

```
  Samp% |      Samp |    Imb. |   Imb. |Group
        |           |    Samp | Samp%  | Function
        |           |         |        |  PE=HIDE

 100.0% | 12,389.4 |      -- |     -- |Total
```

```
|---------------------------------------------------------------------------
|  53.9% |  6,676.5 |       -- |     -- |MPI
||--------------------------------------------------------------------------
||  29.3% |  3,630.5 | 1,213.5 | 25.1% |mpi_bcast
||   9.6% |  1,183.5 | 1,815.5 | 60.7% |MPI_ALLREDUCE
||   7.9% |    982.1 |   752.9 | 43.5% |MPI_REDUCE
||   6.1% |    755.4 |    12.6 |  1.6% |mpi_comm_split
||==========================================================================
|  38.4% |  4,759.0 |       -- |     -- |USER
||--------------------------------------------------------------------------
||  14.4% |  1,789.1 |   108.9 |  5.8% |mat_inv_mp_inverse_gj_
||   5.8% |    723.3 |   122.7 | 14.5% |dist_fn_mp_get_source_term_
||   5.1% |    635.4 |    86.6 | 12.0% |dist_fn_mp_invert_rhs_1_
||   3.6% |    445.7 |   106.3 | 19.3% |dist_fnget_source_term_mp_set_source_
||   1.6% |    198.6 |   288.4 | 59.3% |dist_fn_mp_invert_rhs_linked_
||   1.2% |    146.8 |    80.2 | 35.4% |dist_fn_mp_getan_nogath_
||==========================================================================
|   7.6% |    936.5 |       -- |     -- |ETC
||--------------------------------------------------------------------------
||   5.2% |    639.0 |   166.0 | 20.7% |__intel_memset
||   2.0% |    245.0 |   109.0 | 30.9% |__intel_ssse3_rep_memcpy
|===========================================================================
```

1 field, 4096 processes:

```
  Samp% |     Samp |     Imb. |  Imb. |Group
        |          |     Samp | Samp% | Function
        |          |          |       |  PE=HIDE

 100.0% | 9,638.4 |       -- |    -- |Total
|------------------------------------------------------------------------------
|  80.2% | 7,727.3 |       -- |     -- |MPI
||-----------------------------------------------------------------------------
||  49.8% | 4,801.5 |    839.5 | 14.9% |mpi_bcast
||  16.7% | 1,613.0 | 1,185.0 | 42.4% |MPI_ALLREDUCE
||   7.6% |   730.0 | 1,045.0 | 58.9% |MPI_REDUCE
||   3.2% |   307.2 |     17.8 |  5.5% |mpi_comm_split
||   2.6% |   248.3 |    109.7 | 30.7% |mpi_waitall
||=============================================================================
|  16.9% | 1,632.6 |       -- |     -- |USER
||-----------------------------------------------------------------------------
||   7.3% |   705.4 |    106.6 | 13.1% |mat_inv_mp_inverse_gj_
||   2.4% |   230.5 |     67.5 | 22.6% |redistribute_mp_c_redist_33_mpi_copy_nonblock_
||   1.2% |   111.0 |     47.0 | 29.7% |dist_fn_mp_get_source_term_
||=============================================================================
|   2.7% |   262.1 |       -- |     -- |ETC
||-----------------------------------------------------------------------------
||   1.6% |   154.0 |     67.0 | 30.3% |__intel_memset
|==============================================================================
```

It is evident from the 1 field profiling that MPI significantly dominates performance both at the small and large process counts. MPI accounts for ~55% of the runtime at 512 processes, and over 80% of the runtime at 4096 processes. MPI broadcast is the most costly MPI routine, followed by allreduce.

Broadcast is used in a number of places in the code, including initialisation, progress checking, and the fields update. Analysis of the increase in broadcast time shows that a significant proportion of the increase in the broadcast time comes from the fields calculation functionality.

Allreduce is entirely associated with fields calculation, so the increased in impact of allreduce on the overall profile is directly related to the fields.

However, 1 field linear calculations do not have large amounts of work associated with them, meaning initialisation costs, and field calculations, dominate a short simulation such as the one profiled for these results. Therefore, examining the 2 and 3 field simulations may give us a fuller picture of the scaling of GS2.

2 fields, 512 processes

```
  Samp% |      Samp |     Imb. |  Imb. |Group
        |           |     Samp | Samp% | Function
        |           |          |       |  PE=HIDE

 100.0% | 32,399.2 |       -- |    -- |Total
|-----------------------------------------------------------------------------
|  62.7% | 20,306.1 |       -- |    -- |USER
||----------------------------------------------------------------------------
||  50.1% | 16,234.1 |   872.9 |  5.1% |mat_inv_mp_inverse_gj_
||   2.4% |    773.4 |    87.6 | 10.2% |dist_fn_mp_get_source_term_
||   2.1% |    686.7 |    78.3 | 10.2% |dist_fn_mp_invert_rhs_1_
||   1.5% |    478.8 |    72.2 | 13.1% |dist_fnget_source_term_mp_set_source_
||   1.3% |    418.7 |    96.3 | 18.7% |dist_fn_mp_getan_nogath_
||   1.0% |    340.0 |   119.0 | 26.0% |le_grids_mp_integrate_species_master_
||============================================================================
|  33.1% | 10,735.6 |       -- |    -- |MPI
||----------------------------------------------------------------------------
||  13.7% |  4,426.5 | 1,342.5 | 23.3% |mpi_bcast
||  11.8% |  3,818.1 | 7,204.9 | 65.5% |MPI_ALLREDUCE
||   6.0% |  1,946.0 | 1,067.0 | 35.5% |MPI_REDUCE
||============================================================================
|   4.1% |  1,336.9 |       -- |    -- |ETC
||----------------------------------------------------------------------------
||   2.8% |    893.6 |   216.4 | 19.5% |__intel_memset
||   1.2% |    378.7 |   129.3 | 25.5% |__intel_ssse3_rep_memcpy
|=============================================================================
```

2 fields, 4096 processes

```
  Samp% |      Samp |      Imb. |  Imb. |Group
        |           |      Samp | Samp% | Function
        |           |           |       |  PE=HIDE

 100.0% | 26,344.0 |        -- |    -- |Total
|--------------------------------------------------------------------------------
|  49.3% | 12,985.1 |        -- |    -- |USER
||-------------------------------------------------------------------------------
||  43.9% | 11,567.3 |  1,997.7 | 14.7% |mat_inv_mp_inverse_gj_
||   1.2% |    312.2 |     95.8 | 23.5% |redistribute_mp_c_redist_33_mpi_copy_nonblock
||===============================================================================
|  49.0% | 12,909.8 |        -- |    -- |MPI
||-------------------------------------------------------------------------------
||  21.9% |  5,766.7 | 10,028.3 | 63.5% |MPI_ALLREDUCE
||  20.5% |  5,393.7 |    950.3 | 15.0% |mpi_bcast
||   3.4% |    890.3 |  1,324.7 | 59.8% |MPI_REDUCE
||   2.0% |    515.0 |    245.0 | 32.2% |mpi_waitall
||   1.2% |    306.9 |     14.1 |  4.4% |mpi_comm_split
||===============================================================================
|   1.6% |    430.3 |        -- |    -- |ETC
|================================================================================
```

We can see from the profiles above that moving to the 2 field calculations means GS2 is spending more time performance the calculations to initialise and update the fields (`mat_inv_mp_inverse_gj_` is a routine involved in the initialisation of the fields).

However, we can still see similar patterns associated with the MPI communications when moving from 512 processes to 4096 processes, the broadcast and allreduce time increases, and this time can be attributed to the field calculations. Indeed, in this 2 field simulation we now see MPI allreduce becoming the most costly MPI routine at 4096 cores.

3 fields, 512 processes

```
  Samp% |      Samp |      Imb. |  Imb. |Group
        |           |      Samp | Samp% | Function
```

```
       |          |          |       |  PE=HIDE

 100.0% | 82,205.3 |        -- |    -- |Total
|-----------------------------------------------------------------
|  76.6% | 62,965.7 |        -- |    -- |USER
||----------------------------------------------------------------
||  70.0% | 57,576.2 |  2,139.8 |  3.6% |mat_inv_mp_inverse_gj_
||   1.0% |    831.0 |    131.0 | 13.6% |dist_fn_mp_get_source_term_
||================================================================
|  21.3% | 17,520.1 |        -- |    -- |MPI
||----------------------------------------------------------------
||  10.4% |  8,530.0 | 22,185.0 | 72.4% |MPI_ALLREDUCE
||   6.9% |  5,691.9 |  1,475.1 | 20.6% |mpi_bcast
||   3.3% |  2,725.0 |  1,524.0 | 35.9% |MPI_REDUCE
||================================================================
|   2.1% |  1,698.2 |        -- |    -- |ETC
||----------------------------------------------------------------
||   1.4% |  1,132.7 |    253.3 | 18.3% |__intel_memset
|=================================================================
```

3 fields, 4096 processes
```
  Samp% |     Samp |     Imb. |  Imb. |Group
        |          |     Samp | Samp% | Function
        |          |          |       |  PE=HIDE

 100.0% | 69,154.0 |        -- |    -- |Total
|------------------------------------------------------------
|  67.2% | 46,486.6 |        -- |    -- |USER
||-----------------------------------------------------------
||  64.3% | 44,472.2 |  3,166.8 |  6.6% |mat_inv_mp_inverse_gj_
||===========================================================
|  31.9% | 22,045.2 |        -- |    -- |MPI
||-----------------------------------------------------------
||  19.9% | 13,736.1 | 30,416.9 | 68.9% |MPI_ALLREDUCE
||   9.3% |  6,453.4 |    997.6 | 13.4% |mpi_bcast
||   1.8% |  1,253.6 |  1,930.4 | 60.6% |MPI_REDUCE
|============================================================
```

Examining the profiles for the 3 field simulations we can see the same trend observed moving from 1 field to 2 fields, the field initialisation functionality is becoming more costly, and the MPI associated with the fields calculations is dominating the MPI part of the code.

The profiles of $yxles$ match the scaling that is demonstrated in Figures 1-8, i.e. that the initialisation does not scale well, and gets significantly more costly as multiple fields are considered. We can also see from the graphs that the advanced time for $yxles$ is significantly smaller than the initialisation time (for the number of simulation steps we have profiled), which means it is hard to draw any conclusions on the advance time performance from the above profiles.

However, the other 2 layouts ($xyles$ and $lexys$) exhibit different scaling characteristics and runtimes to $yxles$ so we examine their profiles next to see if we can gain further understanding of the performance of the fields code (and GS2 in general) from them:

Profiling result for $xyles$:

1 field, 512 processes
```
  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  PE=HIDE

 100.0% | 11,025.3 |      -- |    -- |Total
|-----------------------------------------------------------------------
```

```
|  50.9% |  5,613.9 |       -- |     -- |MPI
||-----------------------------------------------------------------------
||  31.9% |  3,517.0 | 1,094.0 | 23.8% |mpi_bcast
||   8.8% |    970.8 | 1,754.2 | 64.5% |MPI_ALLREDUCE
||   7.4% |    815.1 |   701.9 | 46.4% |MPI_REDUCE
||   1.8% |    195.6 |    11.4 |  5.5% |mpi_comm_split
||=======================================================================
|  40.1% |  4,422.9 |       -- |     -- |USER
||-----------------------------------------------------------------------
||  16.2% |  1,782.2 |    63.8 |  3.5% |mat_inv_mp_inverse_gj_
||   5.3% |    587.2 |    67.8 | 10.4% |dist_fn_mp_invert_rhs_1_
||   4.3% |    474.9 |    68.1 | 12.6% |dist_fn_mp_get_source_term_
||   3.5% |    390.0 |    65.0 | 14.3% |dist_fnget_source_term_mp_set_source_
||   1.8% |    194.2 |   241.8 | 55.6% |dist_fn_mp_invert_rhs_linked_
||   1.3% |    145.2 |    42.8 | 22.8% |dist_fn_mp_getan_nogath_
||   1.0% |    115.3 |    57.7 | 33.4% |le_grids_mp_integrate_species_master_
||   1.0% |    111.8 |   317.2 | 74.1% |redistribute_mp_c_fill_3_
||=======================================================================
|   8.8% |    970.9 |       -- |     -- |ETC
||-----------------------------------------------------------------------
||   5.8% |    638.5 |   101.5 | 13.7% |__intel_memset
||   2.2% |    241.7 |    64.3 | 21.1% |__intel_ssse3_rep_memcpy
|========================================================================
```

1 field, 4096 processes

```
  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  PE=HIDE

 100.0% | 8,182.4 |       -- |     -- |Total
|-----------------------------------------------------------------------------
|  87.0% | 7,118.8 |       -- |     -- |MPI
||----------------------------------------------------------------------------
||  57.1% | 4,672.3 |    877.7 | 15.8% |mpi_bcast
||   9.7% |   795.3 | 1,488.7 | 65.2% |MPI_REDUCE
||   9.6% |   787.9 |    280.1 | 26.2% |MPI_ALLREDUCE
||   4.5% |   368.9 |     72.1 | 16.3% |MPI_ALLGATHERV
||   3.1% |   253.8 |     41.2 | 14.0% |mpi_waitall
||   2.4% |   200.4 |     13.6 |  6.4% |mpi_comm_split
||============================================================================
|   9.5% |   777.8 |       -- |     -- |USER
||----------------------------------------------------------------------------
||   2.2% |   177.2 |     52.8 | 22.9% |redistribute_mp_c_redist_33_mpi_copy_nonblock_
||   1.0% |    83.8 |     36.2 | 30.2% |dist_fn_mp_invert_rhs_1_
||============================================================================
|   3.3% |   269.6 |       -- |     -- |ETC
||----------------------------------------------------------------------------
||   1.6% |   131.5 |     53.5 | 28.9% |__intel_memset
|=============================================================================
```

With this layout we can see different performance moving from 512 to 4096 processes. Firstly, we can see that at the low process count MPI isn't as dominant, and we see computational routines associated with calculating the fields update and linear timestep appearing in the profile. MPI broadcast is still the dominant MPI call, with allreduce the second, and one further investigation, as with the *yxles* profiles, all the allreduce time, and a portion of the broadcast time can is attributable to the fields calculations.

When moving to 4096 processes we can see the MPI becomes completely dominant, the computational part of the code now only accounts for 10% of the runtime, and the initialisation of the fields is not appearing as a costly feature. This matches well with the improved scaling of the initialisation (as compared to *yxles*) that we see in Figures 3-5.

We can see that broadcast takes a much higher proportion of the runtime, and we can attribute this to the fields calculations. Furthermore, the MPI allgatherv routine also appears in the profile for 4096 cores, which is called from the fields calculation routines.

2 fields, 512 processes

```
  Samp% |      Samp |     Imb. |   Imb. |Group
        |           |     Samp |  Samp% | Function
        |           |          |        |  PE=HIDE

 100.0% | 31,556.2 |       -- |     -- |Total
|---------------------------------------------------------------------------
|  63.1% | 19,904.8 |       -- |     -- |USER
||--------------------------------------------------------------------------
||  51.3% | 16,173.6 |    479.4 |   2.9% |mat_inv_mp_inverse_gj_
||   2.0% |    637.9 |     79.1 |  11.0% |dist_fn_mp_invert_rhs_1_
||   1.6% |    519.1 |     87.9 |  14.5% |dist_fn_mp_get_source_term_
||   1.3% |    418.9 |     74.1 |  15.1% |dist_fnget_source_term_mp_set_source_
||   1.3% |    398.2 |     85.8 |  17.8% |dist_fn_mp_getan_nogath_
||   1.1% |    343.4 |    106.6 |  23.7% |le_grids_mp_integrate_species_master_
||==========================================================================
|  32.6% | 10,287.6 |       -- |     -- |MPI
||--------------------------------------------------------------------------
||  16.9% |  5,335.4 | 1,350.6 |  20.2% |mpi_bcast
||   9.1% |  2,868.3 | 7,014.7 |  71.1% |MPI_ALLREDUCE
||   5.4% |  1,707.3 | 1,076.7 |  38.7% |MPI_REDUCE
||==========================================================================
|   4.3% |  1,342.8 |       -- |     -- |ETC
||--------------------------------------------------------------------------
||   2.8% |    872.9 |    188.1 |  17.8% |__intel_memset
||   1.2% |    366.9 |    131.1 |  26.4% |__intel_ssse3_rep_memcpy
|===========================================================================
```

2 fields, 4096 processes

```
  Samp% |      Samp |     Imb. |   Imb. |Group
        |           |     Samp |  Samp% | Function
        |           |          |        |  PE=HIDE

 100.0% | 14,803.9 |       -- |     -- |Total
|------------------------------------------------------------------------------
|  87.5% | 12,959.2 |       -- |     -- |MPI
||-----------------------------------------------------------------------------
||  40.0% |  5,917.4 | 1,179.6 |  16.6% |mpi_bcast
||  23.5% |  3,479.2 |    537.8 |  13.4% |MPI_ALLGATHERV
||  13.2% |  1,954.5 |    717.5 |  26.9% |MPI_ALLREDUCE
||   7.3% |  1,083.2 | 1,849.8 |  63.1% |MPI_REDUCE
||   2.0% |    292.7 |     42.3 |  12.6% |mpi_waitall
||   1.3% |    191.0 |     15.0 |   7.3% |mpi_comm_split
||=============================================================================
|   9.8% |  1,447.3 |       -- |     -- |USER
||-----------------------------------------------------------------------------
||   3.0% |    450.7 | 3,305.3 |  88.0% |mat_inv_mp_inverse_gj_
||   1.5% |    217.4 |     63.6 |  22.6% |redistribute_mp_c_redist_33_mpi_copy_nonblock_
||   1.0% |    145.8 |    113.2 |  43.7% |fields_local_mp_advance_local_
||=============================================================================
|   2.6% |    380.1 |       -- |     -- |ETC
||-----------------------------------------------------------------------------
||   1.1% |    167.1 |     63.9 |  27.6% |__intel_memset
|==============================================================================
```

Moving to 2 fields, we can see the trends apparent become more extreme with more fields. We can see that at 4096 cores the allgatherv becomes much more important in terms of the runtime, and this is a direct consequence of the fields calculation.

We can see that the fields initialisation (`mat_inv_mp_inverse_gj_`) is still important at 512 processes, but much less so at 4096 cores.

3 fields, 512 processes

```
  Samp% |      Samp |      Imb. |   Imb. |Group
        |           |      Samp |  Samp% | Function
        |           |           |        |  PE=HIDE
```

```
 100.0% | 79,161.2 |       -- |    -- |Total
|------------------------------------------------------------
|  78.7% | 62,285.4 |       -- |    -- |USER
||-----------------------------------------------------------
||  72.4% | 57,324.6 |  1,300.4 |  2.2% |mat_inv_mp_inverse_gj_
||===========================================================
|  19.1% | 15,141.8 |       -- |    -- |MPI
||-----------------------------------------------------------
||   8.3% |  6,601.7 | 21,817.3 | 76.9% |MPI_ALLREDUCE
||   7.3% |  5,770.3 |  1,374.7 | 19.3% |mpi_bcast
||   3.1% |  2,446.4 |  1,459.6 | 37.4% |MPI_REDUCE
||===========================================================
|   2.2% |  1,712.7 |       -- |    -- |ETC
||-----------------------------------------------------------
||   1.4% |  1,119.5 |    177.5 | 13.7% |__intel_memset
|============================================================
```

3 fields, 4096 processes

```
  Samp% |     Samp |     Imb. |  Imb. |Group
        |          |     Samp | Samp% | Function
        |          |          |       |  PE=HIDE

 100.0% | 28,895.4 |       -- |    -- |Total
|------------------------------------------------------------
|  88.3% | 25,508.1 |       -- |    -- |MPI
||-----------------------------------------------------------
||  43.4% | 12,548.4 |  1,981.6 | 13.6% |MPI_ALLGATHERV
||  24.6% |  7,122.6 |  1,416.4 | 16.6% |mpi_bcast
||  13.6% |  3,941.9 |  2,774.1 | 41.3% |MPI_ALLREDUCE
||   5.0% |  1,433.1 |  2,258.9 | 61.2% |MPI_REDUCE
||===========================================================
|   9.9% |  2,870.7 |       -- |    -- |USER
||-----------------------------------------------------------
||   5.7% |  1,644.8 | 11,949.2 | 87.9% |mat_inv_mp_inverse_gj_
||===========================================================
|   1.7% |    499.4 |       -- |    -- |ETC
|============================================================
```

Finally, using 3 fields it is apparent that the field communication code is dominating MPI performance at both 512 and 4096 processes (with allreduce and allgatherv being the most costly MPI functions, both only used by the fields calculations).

Similar performance features can be seen when investigating the $lexy$s layout, profiles are available in Appendix B.

We can see from the profiles and scaling graphs that have been presented that the performance of GS2 does not improve by increase the processes used, once we go beyond a certain number of processes, and that the scaling issues are associated with MPI costs dominating performance.

5 WP1 Improving the performance of the velocity space integration

The current code calculates the velocity space integration by performing a pre-calculation on the g array for each field of the following form (or similar form, the actual calculate varies by field):

```
do iglo = g_lo%llim_proc, g_lo%ulim_proc
  do isgn = 1, 2
    g0(:,isgn,iglo) = aj0(:,iglo)*gnew(:,isgn,iglo)
  end do
end do
```

Where `g0` is a temporary array used to accumulate the modified g data prior to the integration `g_lo` is the g data decomposition structure, and `iglo` iterates through all the points this process has in the g decomposition.

Once this has been performed then each process then undertakes the velocity space integration by iterating through all points they have in the g decomposition and incorporating it into its velocity space integration total, then finally all processes that have this x, y point in the g decomposition combine their data to calculate the final integration for those x, y points; as follows:

```
do iglo = g_lo%llim_proc, g_lo%ulim_proc
   ik = ik_idx(g_lo,iglo)
   it = it_idx(g_lo,iglo)
   ie = ie_idx(g_lo,iglo)
   is = is_idx(g_lo,iglo)
   il = il_idx(g_lo,iglo)

   total(:, it, ik) = total(:, it, ik) +
weights(is)*w(ie)*wl(:,il)*(g(:,1,iglo)+g(:,2,iglo))
end do

call sum_allreduce_sub(total,g_lo%xyblock_comm)
```

This code relies on the g decomposition being regular so that x, y points are common to groups of processes, so that the code can construct MPI sub-communicators to match those x, y points and enable the allreduce in the above code to ensure every process with that set of x, y points gets the correct final result.

The above code will perform an allreduce for each field in the calculation, and for every block of x, y points (every sub-communicator that has been constructed across x, y points).

Two approaches to optimising the velocity space integration functionality outlined above were considered. The first approach (localised velocity integration) redistributes the g data prior to the integration calculation to enable the integration to be calculated locally. This process involves creating a new data distribution in GS2, mapping the g data to that decomposition, and then performing integration over the re-mapped data.

The second approach (localised field calculation) extends this work beyond the velocity space integration to take advantage of this new data decomposition in the field calculations, implements field calculations based on this layout to re-use the redistribution undertaken for the fields calculation as well (and thus removing the requirement to redistribute the result of the integration).

5.1 Localised velocity integration

As can be seen from the code outlined above, the current velocity space integration operates on the g data object. It undertakes the velocity space integration locally on g

and then send that partial velocity space integration to all other processes who have part of those x, y points in g. This has the potential to be a very efficient method where all of x and y are local to a process (for a given part of l, e, s), as no communications would be necessary to perform the integration, it would simply require local calculations.

However, when using large process counts are used for a simulation x and y are very likely to be split across processes, with multiple processes having parts of the data for a given x and y point.

An alternative approach would be to redistribute the g data so that all the data for a given x, y point is guaranteed to be on a single process, regardless of the decomposition of that data in g.

The benefits of this approach is that, although communication would have to be undertaken prior to the integration to get the data from g to the new distribution, this communication can be done with point-to-point communications (between the processes who have the data in g and those that will have it in the new decomposition) rather than collective routines, as it currently the case for the integration; and that the data only needs to be redistributed once regardless of the number of fields used (so the same amount of data needs to be redistributed to calculated 1 fields as it does for 3 fields).

There is a further step that would be required to enable such a velocity space integration, the communication of the final integral back to the processes that need it after the calculation has taken place. However, as with the g redistribute, this can be done with point-to-point communications, rather than collective operations, and the data for all the fields can be communicated at the same time, meaning you don't need such a communication for each of the fields.

Algorithmically, the proposal is to move from this approach:

```
for each field
  calculate local velocity space integration
  communicate to calculate global velocity space
  integration with all processes involved in that section
  of data
end for
```

To this approach:

```
redistributed the data so that only one process has all
the data for the velocity space calculation of a given
point
for each field
  calculate global velocity space integration for all the
  points I own
end for
```

```
communicate global velocity space to all process that
have part of that space in the original data
decomposition
```

5.1.1 g-fields Data Decomposition

GS2 has a framework for moving data between different data distributions or decompositions. It has redistribution routine, which use point-to-point MPI messages to send and receive data, and layout objects that specify how data is decomposed across processes and how to convert one layout into another.

Storing the g data for linear calculations uses the `g_lo` layout which, as has already been discussed, decomposes data across processes using the layout string passed in the input file for the simulation (i.e. $xyles$).

There are other layouts in GS2, ones for transforming the data between normal space (g) to Fourier space for the non-linear calculations; and for transforming from g into a format that makes calculating collisions simpler.

To enable local calculation of velocity space integration we are going to define a new layout, `gf_lo`. This layout parallelises over x and y, with everything else local to a process (i.e. for a given x and y point all the associated data will be on a single process).

To enable the redistribution between `g_lo` and `gf_lo` data decompositions we need to create some new data redistribution routines to be able send data from a 6-D layout to a 3-D layout, and vice versa. This involved adding the routines `c_redist_36`, `c_redist_36_inv`, and associated blocking and non-blocking communication routines called from these two redistribute functions.
To construct the `gf_lo` data decomposition we implemented two different options:

1. Simple distribution that takes all x, y (i.e. $x * y$ points) and assigns them to the first M processes. This will mean that processes with IDs from 0 to M-1 get points in the `gf_lo` data domain, and processes M to N (where N is the total number of processes running the simulation) get no data in `gf_lo`.
2. Scatter distribution that takes all x, y (i.e. $x * y$ points) and attempts to assign them evenly across the distribution of processes running the simulation. In practice this means that if you have N processes and P points, and P=N/2, then every other process would get a point in the `gf_lo` data domain. This approach is attempting to utilise all computational nodes being used to run the simulation, but not all the processes (as using all the nodes will enable using all the network connections, processor caches, etc… available to the simulation).

Which `gf_lo` data decomposition is used can be set by a new GS2 input parameter (in the `layouts_knobs` input list):

- `simple_gf_decomposition`

Where `simple_gf_decomposition = .true.` uses decomposition technique 1 above, and `simple_gf_decomposition = .false.` uses decomposition technique 2 above (scatter decomposition).

The final functionality required to fully implement the `gf_lo` data decomposition is the code that will specify how to convert between `g_lo` and `gf_lo` and vice versa. This is implemented in a new routine in the `le_grids` module, `init_g2gf_redistribute`.

With all these features implemented it is possible to convert between `g_lo` and `gf_lo` using the code:

```
    call gather(g2gf, g, gf)
```

and do the reverse conversion using:

```
     call scatter(g2gf, gf, g)
```

where `g2gf` is the layout data structure that specifies how data can be converted between the two data decompositions, and `gf` and `g` are data arrays that are structured to sort the data in the required format (i.e. either 3-D or 6-D).

5.1.2 local integration

The gather routine that converts data from `g_lo` to `gf_lo` can then be utilised to convert the integrate functionality from:

```
do iglo = g_lo%llim_proc, g_lo%ulim_proc
  do isgn = 1, 2
    g0(:,isgn,iglo) = aj0(:,iglo)*gnew(:,isgn,iglo)
  end do
end do

do iglo = g_lo%llim_proc, g_lo%ulim_proc
   ik = ik_idx(g_lo,iglo)
   it = it_idx(g_lo,iglo)
   ie = ie_idx(g_lo,iglo)
   is = is_idx(g_lo,iglo)
   il = il_idx(g_lo,iglo)

   total(:, it, ik) = total(:, it, ik) +
weights(is)*w(ie)*wl(:,il)*(g(:,1,iglo)+g(:,2,iglo))
end do

call sum_allreduce_sub(total,g_lo%xyblock_comm)
```

To:

```
do iglo = g_lo%llim_proc, g_lo%ulim_proc
```

```
  do isgn = 1, 2
    g0(:,isgn,iglo) = aj0(:,iglo)*gnew(:,isgn,iglo)
  end do
end do

call gather(g2gf, g, gf)

do igf = gf_lo%llim_proc,gf_lo%ulim_proc
   it = it_idx(gf_lo,igf)
   ik = ik_idx(gf_lo,igf)
   do il = 1,gf_lo%nlambda
      do ie = 1,gf_lo%negrid
         do is = 1,gf_lo%nspec
          total(:,it,ik) = total(:,it,ik) +
weights(is)*w(ie)*wl(:,il)*(gf(:,1,is,ie,il,igf)+gf(:,2,i
s,ie,il,igf))
          end do
      end do
   end do
end do
```

However, this is only a partial solution, as after this integration the data is complete on the process that owns it in `gf_lo`, but not necessarily on all the processes that need it for the matrix-vector calculation to complete the field update (the processes that are part of the supercell that the `gf_lo` point belongs to). Therefore, we needed to create some communication code to redistribute the final result of the velocity space integration to the process that needed it.

5.1.3 Redistributing the velocity space integral

As discussed in section 3, the decomposition of the matrix-vector calculation for a given supercell is done in the fields code, there is a fields data decomposition constructed and used for all the fields calculation. Therefore, if we are required to send data from the processes that owns it in `gf_lo` to the processes that need it in the fields supercell decomposition, we need to be able to map between `gf_lo` and the fields data decomposition.

This functionality was implemented in a new routine in the `fields_local` module, `reduce_an`. `reduce_an` uses information from the fields supercell decomposition, including information on which process is the *supercell head*, to work out where `gf_lo` data should be sent to and then how that is redistributed.

The fields decomposition allocates each supercell a master process called the supercell head that can be responsible for co-ordinating communication between processes that require data in the supercell. We utilise supercell heads to be the receiving processes for those processes in `gf_lo` sending data appropriate for that supercell.

Field supercells consist of a single, unique point in y with one or more associated (linked) points in x, which means any given `gf_lo` data point will only map to one supercell. Therefore, we can easily identify the process to receive the velocity space integration result from the process that calculated in it `gf_lo` (if the sending process and the receiving process are the same then the data is simply copied rather than being sent by MPI).

Each field component of the velocity space integration is sent separately to the supercell head (this could be combined into a single message but that would require copying the data into a temporary buffer), and sent using non-blocking MPI point-to-point communications.

Once the supercell head has received the relevant data it then broadcasts that data to all the other processes in its supercell, using a sub-communicator that contains all the processes that own that data point.

As this new velocity space integration functionality is design to improve the performance of GS2 at large process counts, we only expect this method of performance velocity space integrations to be efficient when we are using more processes than we have x and y points (i.e. when x and y are likely to be split in the g data decomposition). Below this threshold we would expect the existing velocity space integration to be more efficient as it would require too much data movement to redistributed `g_lo` to `gf_lo` in this scenario and the integrations are not the dominant part of the code.

To enable the `gf_lo` velocity space integration a new input parameter has been defined in the `dist_fn` name list: `gf_lo_integrate`. By default this is false, but if it set to true then the `gf_lo` integrate is utilised.

5.1.4 Performance

Evaluating the performance of this new, local, velocity space integration method we ran a number of tests using the $yxles$ data layout, and measured the time to perform the original velocity space integration and the time required to complete the new local integration. The times presented in the table below are the maximum time measured across all processes for those operations:

Number of processes	Original integrate (minutes)	Local integrate (minutes)	Integrate redistribution (minutes)
4096	1.39	0.96	4.11
8192	1.23	0.73	3.11

The original integrate column above is simply the time to complete the original integration code (including the associated allreduce). The local integrate column contains the time to do the redistribution from `g_lo` to `gf_lo` and perform the velocity space calculation. The integrate redistribution column has the time required to redistribute the calculated velocity space integral back to the processes in the supercell that require the data for the matrix-vector calculation.

We can see from the data in the table above that our approach to the velocity space integration does reduce the cost of that calculation. Redistributing the data from `g_lo` to `gf_lo` and then performing the calculation locally is cheaper than performing the distributed calculation. However, the cost of then sending the calculated result to the processes that require it far outweighs the benefits we have gained in performing the calculations locally.

Primarily, this is because we are performing the velocity space calculation in a data distribution that does not match the data distribution being used for the fields calculation (i.e. the matrix-vector functionality).

Therefore, for us to be able to benefit from the local integration code we need to implement a new fields decomposition that matches our `gf_lo` data decomposition, thereby removing the need to before the redistribution after the integration, and maintaining our performance benefits from the local integration.

5.2 Localised field calculation

Given the `gf_lo` functionality, already described in the previous section, we designed a new data decomposition for the fields calculation, in a new fields module called `fields_gf_local`. This new module follows the same structure as the existing `fields_local` module, but significantly alters the data decomposition, and therefore the matrix-vector calculation.

5.2.1 gf_lo field decomposition

As previously discussed, the supercell data decomposition for the fields calculation is calculated as follows (for each supercell):

1. Calculates a balanced blocksize for all the processes in the supercell.
2. Implements an actual blocksize which is Max(balanced blocksize, user_defined_value), which can introduce a load imbalance to reduce communication costs.
3. Records allocated work, and assigns blocks of work to available processes, starting with those processes with the least amount of work already assigned.

Membership of the supercell is decided by whether a process owns any of the supercell in g, i.e. are any of the x and y points it has in the g decomposition part of this supercell (remembering that a supercell is a single y point with one or more associated x points).

If we consider membership of a supercell with reference to `gf_lo` instead of `g_lo`, we can follow the same procedure, but as a single x and y point in `gf_lo` is only every owned by one process we no longer need to worry about distributing work between processes, our membership of the supercell is simply the processes that own the x and y points in that supercell and also in `gf_lo`.

As each process in the supercell will own the whole data for one (or more) x and y point(s) we need not worry about splitting blocks or block sizes, each process should have the same amount of data/work in the supercell (assuming we are using more processes than we have x and y points and therefore every process in the supercell will only have a single point).

Therefore, the decomposition for our new fields module is calculated simply as follows:

1. Iterate through all supercells
2. For each supercell iterate through each cell (a cell is a single x and y point)
3. If I am the process that owns that cell in `gf_lo` then I have the work to do in the fields calculation for that cell
4. Otherwise I ignore that cell

5.2.2 Communication routines

Whilst the new fields module will undertake all its calculations in the `gf_lo` data layout, the rest of the GS2 code (i.e. the core linear calculations) still expect the result to be in the `g_lo` data layout. Furthermore, the initialisation of the fields response matrix (i.e. the setup of the initial fields) uses both the fields calculation code, and the linear simulation code, to obtain the final result.

This means that, unless we change the whole code to use our `gf_lo` data layout, which would limit the ultimate scalability of our simulations to using at most $x * y$ processes and therefore is not desirable, we need to able to convert the final and initial field data from `gf_lo` to `g_lo` and vice versa.

To perform this task we created two new communication routines, `fm_scatter` and `fm_gather`, which send the data between `gf_lo` and `g_lo` data layouts. Whilst this is a communication overhead, the scatter is only performed once per simulation (after initialisation) and gather is only performed once per iteration of the solver, rather than many times per field or per supercell, so it should not significantly impact the performance.

To perform the scatter operation each process simply loops through the field data it has, packs all the fields it has into a single temporary array (so if we are calculating with more than one field we are still only sending a single message for this communication) and sends that data to the owning process in `gf_lo`. The `g_lo` field data is iterated through in a loop over x and y points, so each `gf_lo` point is send individually to the owning process.

Whilst we are undertaking this scatter process we also send all the field data to process 0 (the process MPI rank 0) as currently the full field data is required on this process for diagnostic calculations.

The gather is the reverse operation, although we have integrated the communication functionality required to ensure all processes in a supercell has the field data for that supercell into the gather as well as the functionality to distribute the field data back to the processes that require it in `g_lo`.

To understand the gather operation we need to understand that the matrix-vector calculation we undertake in the fields functionality first does a local matrix-vector on each process in the supercell. Once that has been calculated the global matrix-vector for the supercell is calculated by reducing the local matrix-vector results to the supercell head. At the end of that process the supercell head has the correct data for the matrix-vector calculation in the supercell.

Gather then takes the result on the supercell head and both sends it back to all processes in the supercell (so they have the correct final result) and sends it to process 0 for diagnostic usage.

Once that process has been completed then the process that owns the field data point in `gf_lo` then sends the data back to the processes that own the same point in `g_lo`. At the end of this process all the processes that own a given x and y point in `gf_lo` and `g_lo` have the field data for that point. This means we can update the fields in `gf_lo` and `g_lo` and continue with the linear calculations (and any other operations required by the simulation, i.e. non-linear and collisions).

This does mean that with this new fields functionality we store the field data in two different forms, `gf_lo` and `g_lo`, requiring twice as much data storage for the fields data (although fields data is not large compared to the g data array).

However, whilst this is inefficient in terms of storage, it does mean that we do not need scatter the fields from `g_lo` to `gf_lo` every timestep, we simple do it once after initialisation, then we update both sets of fields (which is a simple operation) each time we calculate the fields, and simply have to covert (gather) from `gf_lo` to `g_lo` each time step, not both ways.

5.2.3 Input parameters

To utilise the new fields module a number of input flags need to be set:

- fields_knobs: `field_option='gf_local'`
- dist_fn_knobs: `gf_lo_integrate= .true.`
- layout_knobs: `gf_local_fields = .true.`

It could that these can be rationalise to a single input variable with a small amount of code refactoring.

5.2.4 Performance

As has already been discussed, this functionality (fields_gf_local) is designed to be used when we are using large process counts for a given simulation, i.e. when we have significantly more processes than x and y points in the simulation. For the simulation we have been using for benchmarking there are 2016 x and y points. Therefore, we evaluated the performance of the new fields at 4032/4096 processes and higher.

The table below outlines the performance of the original code and the new fields code for different layouts and process counts, timings are in minutes. We have colour coded the timings for the new functionality, red where it is slower than the original code, and green where it is fast. This data was collected using the simple fields decomposition, the scatter fields decomposition was also benchmarked but did not give as good a performance as the simple decomposition):

Number of processes	Layout	Number of fields	Original initialise	`gf_lo` initialise	Original advance	`gf_lo` advance
4032	*lexys*	1	0.2	0.28	0.31	0.3
		2	1.15	0.78	0.45	0.35
		3	3.71	2.08	0.63	0.43
4096	*yxles*	1	0.37	0.32	0.42	0.49
		2	3.1	0.95	0.79	0.54
		3	9.63	2.23	0.99	0.61
4096	*xyles*	1	0.17	0.33	0.4	0.54
		2	0.92	1.53	0.94	0.54
		3	2.79	4.01	1.19	0.64
8064	*lexys*	1	0.21	0.37	0.56	0.3
		2	1.15	1.36	0.7	0.37
		3	3.62	3.85	0.86	0.46
8192	*yxles*	1	0.45	0.36	0.49	0.44
		2	1.91	0.69	0.73	0.5
		3	9.3	1.56	1.03	0.58
8192	*xyles*	1	0.14	0.42	0.40	0.40
		2	0.69	1.45	0.57	0.49
		3	1.98	3.97	0.75	0.58

0.21 0.56 1.15 0.7 3.62 0.86

The total improvement column records the improvement in the initialise + advance time (i.e. the time to complete the whole simulation). Note, for these benchmark we were only running for 1000 steps, which is much shorter than normal simulation, and will mean the initialisation time has a bigger impact, proportionally, that it normally would. Finally, the Total(1,000,000 steps) column estimates the impact on a normal, long running, simulation by multiplying the measured advanced time for 1000 steps by 1000 to give an estimate of the advanced runtime for 1,000,000 steps.:

Number of processes	Layout	Number of fields	Initialise	Advance	Total	Total (1,000,000 steps)
4032	*lexys*	1	0.71	1.03	0.87	1.03
		2	1.47	1.28	1.41	1.28
		3	1.78	1.46	1.72	1.46
4096	*yxles*	1	1.15	0.86	0.97	0.85
		2	3.26	1.46	2.61	1.46
		3	4.31	1.62	3.71	1.63
4096	*xyles*	1	0.51	0.74	0.65	0.74
		2	0.60	1.75	0.90	1.73
		3	0.69	1.86	0.85	1.85

8064	*lexys*	1	0.57	1.87	1.15	1.86
		2	0.84	1.89	1.07	1.89
		3	0.94	1.87	1.04	1.86
8192	*yxles*	1	1.28	1.11	1.17	1.11
		2	2.77	1.46	2.22	1.46
		3	5.96	1.77	4.83	1.79
8192	*xyles*	1	0.33	1.00	0.66	0.99
		2	0.47	1.16	0.65	1.16
		3	0.50	1.29	0.60	1.29

We can see that for the short simulation, where initialisation can have a big impact, our new functionality can give a near 5x performance improvement for the whole simulation (8192 processes, 3 fields, $yxles$).

However, the impact of reduced initialisation performance for the $xyles$ simulations means that on short simulations the overall cost of the new functionality is higher than the original functionality, although at higher process counts the new field functionality can make the advance ~29% faster than the original code.

When considering a longer simulation, we can see that the big performance improvements, or impacts, of the new functionality on the initialisation code have a much smaller impact, and the performance improvement due to the new fields functionality is much closer to the performance improvement of the advance simulation functionality.

However, that is not to say that the performance improvements in the initialisation should be ignored as for non-linear simulation it is sometime necessary occasionally need to reset the time step when simulation accuracy calculations exceed a certain limit and this requires recalculating the response matrix (i.e. a full initialisation). Therefore, optimised initialisation means that this is a smaller performance penalty for needing to reset, and therefore means the simulations can be run using stricter accuracy tolerances.

WP2 Improving the distributed matrix-vector product through optimised work decomposition

As previously discussed, the fields supercell decomposition is constructed using the following process:

1. Calculates a balanced blocksize for all the processes in the supercell.
2. Implements an actual blocksize which is max(balanced blocksize, user_defined_value), which can introduce a load imbalance to reduce communication costs.
3. Records allocated work, and assigns blocks of work to available processes, starting with those processes with the least amount of work already assigned.

However, the blocksize is fixed for all supercells in the simulation, either as a pre-defined value in the code, or as an input parameter.

We know that supercells vary in size, with some supercells containing a single (x, y) point, and others being multiple x points associated to a single y point. This means that using a single, fixed, blocksize to determine how to choose which processes participate in the matrix-vector calculation associated with a given supercell may not be optimal.

We implemented an auto-tuning framework in the original `fields_local` module (our new fields module has a fixed decomposition for all points in the fields so there is no scope for tuning the decomposition in this way).

The auto-tuning framework alters the blocksize for the decomposition, constructs the data decomposition, and times a set of matrix-vector operations to evaluate the performance of that particular blocksize. It then resets the decomposition, chooses a new blocksize and does the whole process again.

To enable this functionality we had to alter the decomposition code in the fields module to make sure that the decomposition could be reset without causing problems or losing data. Once this adaption was made the auto-tuning code was straight forward to implement:

```
current_best_size = rowsize
do ik = 1,fieldmat%naky
  current_best = -1
  rowsize = 2
  MinNRow = rowsize
  do i = 1,max_tuning_size
    MinNRow(ik)=rowsize
    call init_fields_matrixlocal(tuning_in=.true.)

    start_time = timer_local()
    call getfield_local(phinew,aparnew,bparnew,
do_gather,do_update)
    end_time = timer_local()

    call finish_fields_local()

    temp_best = end_time - start_time
    max_best = temp_best
    call max_reduce(max_best, 0)
    min_best = temp_best
    call min_reduce(min_best, 0)
    av_best = temp_best
    call sum_reduce(av_best, 0)

    if(iproc .eq. 0) then
      av_best = av_best/nproc
      if(current_best .lt. 0) then
        current_best = max_best
        current_best_size(ik) = rowsize
      else
```

```
        if(max_best .lt. current_best) then
          current_best = max_best
          current_best_size(ik) = rowsize
        end if
      end if
    end if
    rowsize = rowsize*2
  end do
end do

call broadcast(current_best_size)
MinNRow = current_best_size
```

The auto-tuning is performed in the initialisation step, prior to the calculation of the field response matrix, which means the optimal blocksize will be used for that, expensive, calculation.

To enable the user to choose whether to use auto-tuning, and new input parameter has been added to the fields_knobs name list: `field_local_tuneminnrow`. By default this is false, but if `field_local_tuneminnrow = .true.` in the input file the auto-tuning functionality will be used.

5.3 Performance

We evaluated the performance of the auto-tuning using the standard linear benchmark we have used previous. Figures 9 and 10 should the initialisation time for $xyles$ and $yxles$ ($lexys$ was benchmarked but showed no performance improvement). The original lines show the performance of the original code, prior to any of the work we have done in this project. The other lines are timings from the simulation using the auto-tuning functionality.

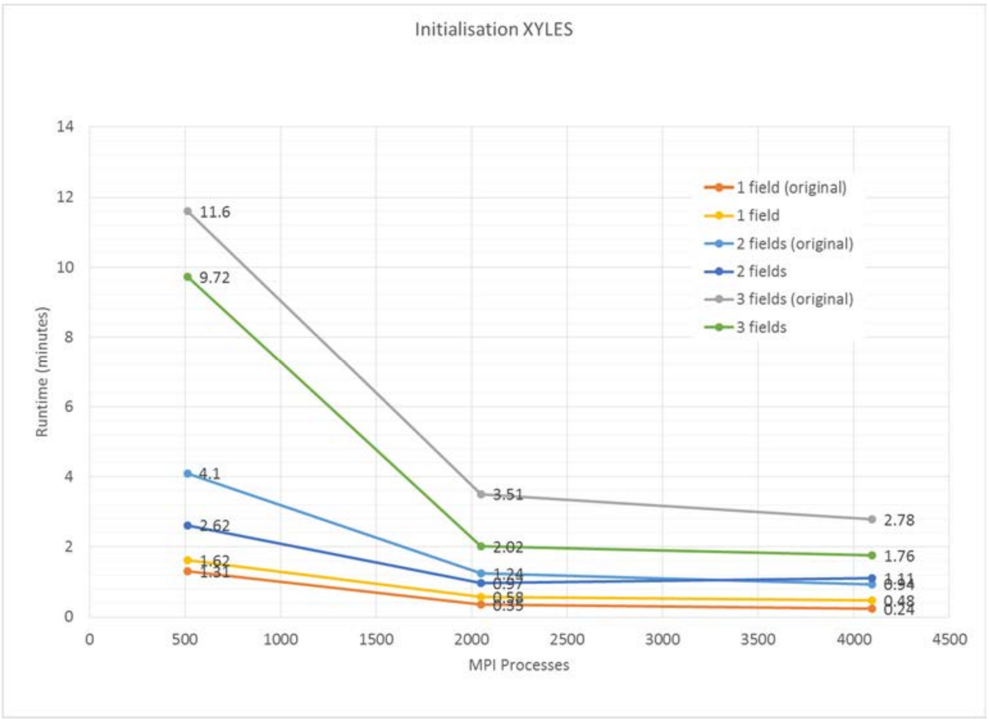

Figure 9: Initialisation performance for YXLES using auto-tuning

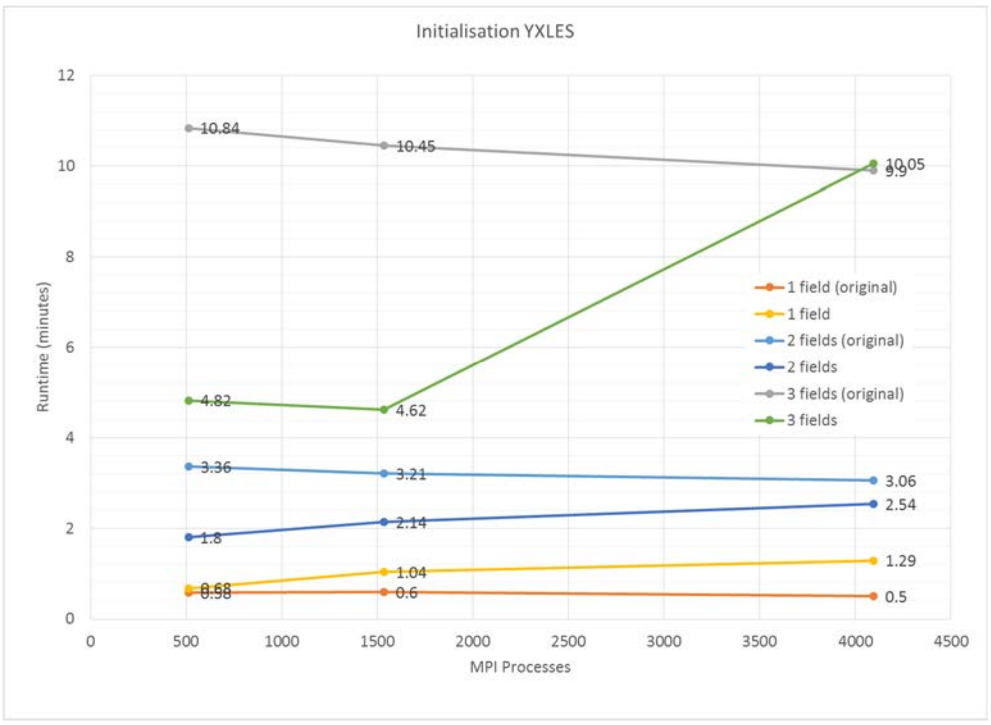

Figure 10: Initialisation performance for YXLES using auto-tuning

It is evident that the blocksize tuning can benefit the initialisation for these two layouts, with significant improvements for higher numbers of fields and lower core counts. In this scenario the blocksize tunes itself to have large blocks, thereby reducing the number of processes involved in the supercell calculation, and performing a similar optimisation to the new fields functionality we have created (i.e. restricting the number of processes involved in the matrix-vector calculations).

The best performance improvement is ~2.25x faster for $yxles$ initialisation at low core counts, and ~73% faster for $xyles$. These improvements are in spite of the fact that auto-tuning the blocksize has associated costs (undertaking a number of matrix-vector calculations).

Figures 11 and 12 demonstrate the performance of the advanced time from the same simulation:

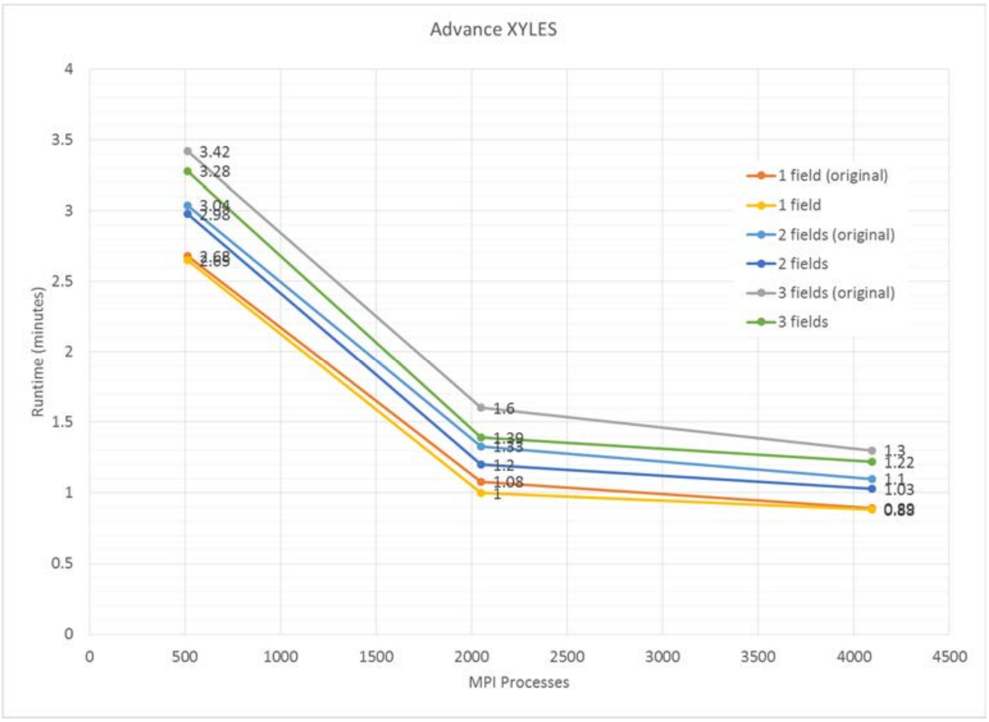

Figure 11: Advance performance for XYLES using auto-tuning

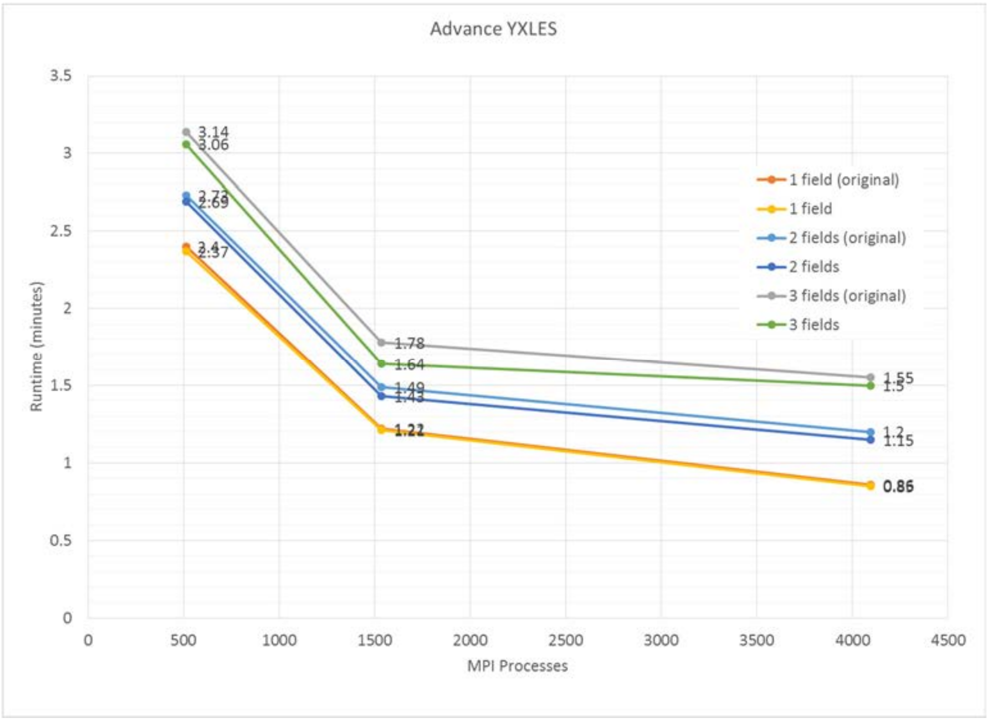

Figure 12: Advance performance for YXLES using auto-tuning

The advanced time shows performance improvements from auto-tuning as well, particularly at high field numbers and lower core counts. The best performance improvement for the advance time is ~15% faster for $xyles$ and ~8% for $yxles$.

5.4 Further functionality

We also implemented a more refined auto-tuning framework with the ability to select a different blocksize for each supercell in a simulation, rather than choosing a single blocksize for the whole simulation.

In theory this could be more efficient as supercells can have different sizes so it is possible that different blocksizes will be optimal for different supercells.

However, it does significantly increase the search space for the auto-tuning, as instead of searching through 10 or 15 different blocksizes it is necessary to seach through 10 or 15 times the number of supercells in the simulation, so can increase the auto-tuning cost by 30-50 times for an average simulation.

This cost significantly increases the initialisation time (for example, from 2 minutes to 25 minutes), and the advance time does not improve significantly compared to the simple blocksize tuning approach, so this functionality has been removed from GS2.

6 WP3 Improving communications involved in matrix-vector product through non-blocking collectives

The original fields code uses a range of collective communication, and new versions of the MPI standard defines a number of non-blocking collective operations.

As the field matrix-vector calculations have a range of supercells to work on, and each supercell has calculations and communications associated with it, there is potential for overlapping the computation of the local matrix-vector products with the communication of that calculated data to supercell members. Specifically, sending supercell data for one supercell whilst calculating the local matrix-vector product for the following supercell.

To utilise the non-blocking collective communication, new versions of the MPI wrapper routines used in GS2 were written, which provided the range of non-blocking collectives required in the fields simulations. They were written in such as way as to default to the standard collective communication routines if GS2 is being used on a machine with an MPI library without MPI-3.0 functionality.

A new input parameter, `field_local_nonblocking_collectives`, has been added to the fields namelist to enable turning the non-blocking collectives on and off (by default it is not enabled).

The non-blocking collectives have been implemented in the original fields code, however they provided little performance improvement (around ~1% in the total run time). Following profiling it is apparent that current there is not enough work to overall the communication with in the current code. However, there are other areas of the code where non-blocking communications may be of use, and the non-blocking functionality is now available for use in GS2.

7 Conclusions

We have implemented a new fields module that uses a new data layout that optimises both the fields update and velocity space integration at large process counts by sacrificing an even data distribution between processes for a reduced amount of communication required to calculate the field update.

This new field solve can provide up to ~5 times performance improvements for short simulations, and approaching 2 times performance improvement for longer running simulations with a range of process counts.

The optimisation generally improves both the initialisation and advance stages of the simulation, and it enables data layouts that were too costly previously (in terms of initialisation) to be used for production simulations.

We have also investigated auto-tuning of the matrix-vector operation the fields update with the previous fields data decomposition, and found that it can improve performance (~8-~15% faster for the advance and ~.7-~2.25 times faster for the initialisation) for some data layouts and types of simulation. This can be used to improve the performance of GS2 at lower process counts (where the new fields functionality is not designed to work).

Finally, we also investigated non-blocking collective communications but found them not to give significant performance improvements, and to be unnecessary in the new fields functionality we have created.

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

8 Future Work

There are a number of areas where further optimisation or functionality is could be added to GS2.

The first area would be to add redistribution functionality to able data in `gf_lo` layout to be redistributed to collision layouts. Collisions are currently very expensive in GS2 simulations, and involve large amounts of communication. However, collisions can use a data layout, `le_lo`, which has similarities to `gf_lo`.

As previously mentioned, the current code performs calculations in the following order: NLCFLC (where F is field calculations and C is collision calculations). Given this sequence, if we could map from collision layout (`le_lo`) directly to our new field layout (`gf_lo`) this could significantly reduce the communications associated with collisions in GS2.

Furthermore, if such a redistribute was implemented, it could be possible to re-order the calculations in GS2, from NLCFLC to NLCFCL (i.e. keep data in similar layouts as much as possible). This would require some careful consideration to ensure it does not adversely affect the numerical accuracy and stability of the simulation (as we are changing for order of the operator splitting scheme being used) but if it is deemed possible without significantly affecting the scientific integrity of the simulations it could have big benefits (i.e. collisional simulations could have very similar computational costs to non-linear simulations).

The diagnostic code in GS2 currently takes all the data needed for diagnostics and sends it to process 0, which performs the diagnostic calculations and outputs the result. If this could be done in a distributed fashion, rather than by a single process this could have significant communication savings.

The decompositions implemented in the new fields functionality is not currently optimal, and could be further improved by taking into account the current distribution in `g_lo` prior to the redistribute. It should also be extended to group x points with a single y when used with less processes than x, y points. This could, potentially, enable the new fields code to be efficient even with low process counts.

Finally, the code we use to gather and scatter fields data in the new fields routines does not take into account the fact that currently process 0 requires all the data. The communications could be adapted to take this into account.

9 Appendix A

GS2 input file used for this project

```
&theta_grid_knobs
 equilibrium_option='eik'
/

&theta_grid_parameters
 rhoc = 0.4
 ntheta = 30
 nperiod= 1
/

&parameters
 beta = 0.04948
 zeff =   1.0
 TiTe = 1.0
/

&collisions_knobs
 collision_model = 'none'
/

&theta_grid_eik_knobs
 itor = 1
 iflux = 1
 irho = 3
 ppl_eq = .false.
 gen_eq =  .false.
 efit_eq = .true.
 gs2d_eq = .true.
 local_eq = .false.
 eqfile = 'equilibrium.dat'
 equal_arc = .false.
 bishop = 1
 s_hat_input = 0.29
  beta_prime_input = -0.5
 delrho = 1.e-3
 isym = 0
 writelots = .false.
/

&fields_knobs
 field_option='local'
 field_subgath = .false.
/

&gs2_diagnostics_knobs
 write_ascii = .false.
 print_flux_line = .true.
 write_flux_line = .false.
 write_nl_flux = .false.
 write_omega = .false.
 write_omavg = .false.
 write_final_moments = .false.
 write_final_fields=.false.
 print_line=.false.
 write_line=.false.

 save_for_restart=.false.
 nsave=          1000

 nwrite=         2000
 navg=           200

 omegatol=  1.0e-5
 omegatinst = 500.0
/

&le_grids_knobs
 ngauss = 8
 negrid = 8
/

&dist_fn_knobs
 boundary_option= "linked"
 gridfac=   1.0
 gf_lo_integrate= .false.
/
```

```
&init_g_knobs
 !restart_file= "nc/input.nc"
 ginit_option= "noise"
 phiinit=   1.e-6
 chop_side = .false.
/

&kt_grids_knobs
 grid_option='box'
/

&kt_grids_box_parameters
 y0 = 10
 ny = 96
 nx = 96
 jtwist = 2
/

&knobs
 fphi= 1.0
 fapar= 1.0
 fbpar= 1.0
 faperp= 0.0
 delt= 1.0e-4
 nstep= 1000
 wstar_units = .false.
/

&species_knobs
 nspec=  2
/

&species_parameters_1
 type  = 'ion'
 z     = 1.0
 mass  = 1.0
 dens  = 1.0
 temp  = 1.0
 tprim = 2.04
 fprim = 0.0
 vnewk = 1.0
 uprim = 0.0
/

&dist_fn_species_knobs_1
 fexpr  = 0.45
 bakdif = 0.05
/

&species_parameters_2
 type  = 'electron'
 z      = -1.0
 mass  = 0.01
 dens  = 1.0
 temp  = 1.0
 tprim = 2.04
 fprim = 0.0
 vnewk = 1.0
 uprim = 0.0
/

&dist_fn_species_knobs_2
 fexpr= 0.45
 bakdif=  0.05
/

&theta_grid_file_knobs
 gridout_file='grid.out'
/

&theta_grid_gridgen_knobs
 npadd = 0
 alknob = 0.0
 epsknob = 1.e-5
 extrknob = 0.0
 tension = 1.0
 thetamax = 0.0
 deltaw = 0.0
 widthw = 1.0
/

&source_knobs
/

&nonlinear_terms_knobs
nonlinear_mode='off'
cfl = 0.5
/

&additional_linear_terms_knobs
/

&reinit_knobs
 delt_adj = 2.0
 delt_minimum = 1.e-8
/

&theta_grid_salpha_knobs
/
&hyper_knobs
/
```

```
&layouts_knobs
 layout = 'xyles'
 local_field_solve = .true.
 unbalanced_xxf = .true.
 max_unbalanced_xxf = 0.5
 unbalanced_yxf = .true.
 max_unbalanced_yxf = 0.5
 opt_local_copy = .true.
 opt_redist_init = .true.
 opt_redist_nbk = .true.
 intmom_sub = .true.
 intspec_sub = .true.
/
```

10 Appendix B

Linear profiling result for *lexys*:

1 field, 448 processes

```
  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  PE=HIDE

 100.0% | 10,103.4 |      -- |    -- |Total
|----------------------------------------------------------------------------
|  65.0% |  6,563.3 |      -- |    -- |MPI
||---------------------------------------------------------------------------
||  37.1% |  3,748.8 | 1,065.2 | 22.2% |mpi_bcast
||  17.8% |  1,793.9 |   287.1 | 13.8% |MPI_ALLGATHERV
||   6.8% |    686.4 | 1,281.6 | 65.3% |MPI_REDUCE
||   2.2% |    221.5 |   724.5 | 76.8% |MPI_ALLREDUCE
||===========================================================================
|  25.6% |  2,588.0 |      -- |    -- |USER
||---------------------------------------------------------------------------
||   6.2% |    631.0 |   277.0 | 30.6% |dist_fn_mp_invert_rhs_1_
||   4.7% |    470.4 |   199.6 | 29.9% |dist_fn_mp_get_source_term_
||   4.3% |    438.9 |   191.1 | 30.4% |dist_fnget_source_term_mp_set_source_
||   2.3% |    227.6 |   280.4 | 55.3% |dist_fn_mp_invert_rhs_linked_
||   1.6% |    159.5 |    53.5 | 25.2% |dist_fn_mp_getan_nogath_
||   1.1% |    109.6 |   590.4 | 84.5% |redistribute_mp_c_redist_33_mpi_copy_nonblock_
||===========================================================================
|   9.2% |    933.5 |      -- |    -- |ETC
||---------------------------------------------------------------------------
||   6.4% |    646.0 |   133.0 | 17.1% |__intel_memset
||   2.2% |    226.6 |   107.4 | 32.2% |__intel_ssse3_rep_memcpy
|============================================================================
```

1 field, 4032 processes

```
  Samp% |    Samp |    Imb. |  Imb. |Group
        |         |    Samp | Samp% | Function
        |         |         |       |  PE=HIDE

 100.0% | 7,151.3 |      -- |    -- |Total
|----------------------------------------------------------------------------
|  86.2% | 6,164.0 |      -- |    -- |MPI
||---------------------------------------------------------------------------
||  59.4% | 4,248.4 |   789.6 | 15.7% |mpi_bcast
||  13.1% |   933.9 |   125.1 | 11.8% |MPI_ALLGATHERV
||   9.3% |   662.0 | 1,162.0 | 63.7% |MPI_REDUCE
||   2.8% |   203.6 |    88.4 | 30.3% |MPI_ALLREDUCE
||===========================================================================
|  10.1% |   724.6 |      -- |    -- |USER
||---------------------------------------------------------------------------
||   2.8% |   200.4 |   102.6 | 33.9% |redistribute_mp_c_redist_33_mpi_copy_nonblock_
||   1.0% |    74.1 |     9.9 | 11.8% |le_grids_mp_legendre_transform_
||   1.0% |    72.8 |    61.2 | 45.7% |dist_fn_mp_invert_rhs_1_
||=============================================================================
|   3.4% |   246.0 |       -- |    -- |ETC
||-----------------------------------------------------------------------------
||   1.9% |   133.8 |    59.2 | 30.7% |__intel_memset
|==============================================================================
```

2 field, 448 processes

```
  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  PE=HIDE

 100.0% | 20,730.1 |      -- |    -- |Total
|---------------------------------------------------------------------------
|  77.0% | 15,960.7 |      -- |    -- |MPI
||--------------------------------------------------------------------------
||  45.7% |  9,469.0 |   932.0 |  9.0% |MPI_ALLGATHERV
||  23.5% |  4,881.8 | 1,308.2 | 21.2% |mpi_bcast
||   4.3% |    886.8 | 1,497.2 | 62.9% |MPI_REDUCE
||   2.6% |    544.8 | 3,480.2 | 86.7% |MPI_ALLREDUCE
||==========================================================================
|  17.0% |  3,518.6 |      -- |    -- |USER
||--------------------------------------------------------------------------
||   3.2% |    669.1 |   594.9 | 47.2% |dist_fn_mp_invert_rhs_1_
||   2.4% |    494.8 |   464.2 | 48.5% |dist_fn_mp_get_source_term_
||   2.2% |    463.7 |   383.3 | 45.4% |dist_fnget_source_term_mp_set_source_
||   1.7% |    349.3 |   224.7 | 39.2% |dist_fn_mp_getan_nogath_
||   1.4% |    290.0 |   452.0 | 61.0% |dist_fn_mp_invert_rhs_linked_
||   1.4% |    284.0 | 6,721.0 | 96.2% |mat_inv_mp_inverse_gj_
||==========================================================================
|   5.9% |  1,231.0 |      -- |    -- |ETC
||--------------------------------------------------------------------------
||   4.1% |    839.6 |   164.4 | 16.4% |__intel_memset
||   1.5% |    320.1 |   189.9 | 37.3% |__intel_ssse3_rep_memcpy
|===========================================================================
```

2 field, 4032 processes

```
  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  PE=HIDE

 100.0% | 14,727.1 |      -- |    -- |Total
|------------------------------------------------------------------------------
|  89.4% | 13,171.3 |      -- |    -- |MPI
||-----------------------------------------------------------------------------
||  43.6% |  6,426.6 |   360.4 |  5.3% |MPI_ALLGATHERV
||  35.5% |  5,229.2 |   900.8 | 14.7% |mpi_bcast
||   5.8% |    852.7 | 1,528.3 | 64.2% |MPI_REDUCE
||   3.3% |    486.1 |   174.9 | 26.5% |MPI_ALLREDUCE
||=============================================================================
|   7.9% |  1,169.2 |      -- |    -- |USER
||-----------------------------------------------------------------------------
||   1.8% |    264.2 |   113.8 | 30.1% |redistribute_mp_c_redist_33_mpi_copy_nonblock_
||   1.3% |    196.7 | 5,929.3 | 96.8% |mat_inv_mp_inverse_gj_
||   1.0% |    140.9 |    77.1 | 35.4% |fields_local_mp_advance_local_
||=============================================================================
|   2.5% |    368.9 |      -- |    -- |ETC
||-----------------------------------------------------------------------------
||   1.2% |    174.4 |    94.6 | 35.2% |__intel_memset
||   1.0% |    140.0 |   166.0 | 54.3% |__intel_ssse3_rep_memcpy
|==============================================================================
```

3 field, 448 processes

```
  Samp% |     Samp |     Imb. |  Imb. |Group
        |          |     Samp | Samp% | Function
        |          |          |       |  PE=HIDE

 100.0% | 43,289.6 |       -- |    -- |Total
|----------------------------------------------------------------------------
|  84.9% | 36,754.2 |       -- |    -- |MPI
||---------------------------------------------------------------------------
||  64.9% | 28,075.6 |  2,212.4 |  7.3% |MPI_ALLGATHERV
||  14.7% |  6,383.0 |  1,660.0 | 20.7% |mpi_bcast
||   2.7% |  1,176.1 |  2,102.9 | 64.3% |MPI_REDUCE
||   2.0% |    858.7 | 10,658.3 | 92.8% |MPI_ALLREDUCE
```

```
||===========================================================================
|  11.6% |  5,016.3 |        -- |     -- |USER
||---------------------------------------------------------------------------
||   2.5% |  1,072.8 | 23,108.2 | 95.8% |mat_inv_mp_inverse_gj_
||   1.6% |    712.8 |    985.2 | 58.2% |dist_fn_mp_invert_rhs_1_
||   1.3% |    544.4 |    767.6 | 58.6% |dist_fn_mp_getan_nogath_
||   1.2% |    526.6 |    671.4 | 56.2% |dist_fn_mp_get_source_term_
||   1.1% |    490.2 |    542.8 | 52.7% |dist_fnget_source_term_mp_set_source_
||===========================================================================
|   3.5% |  1,498.5 |        -- |     -- |ETC
||---------------------------------------------------------------------------
||   2.4% |  1,020.9 |    203.1 | 16.6% |__intel_memset
|============================================================================
```

3 field, 4032 processes

```
  Samp% |      Samp |      Imb. |  Imb. |Group
        |           |      Samp | Samp% | Function
        |           |           |       |  PE=HIDE

 100.0% | 31,630.8 |        -- |     -- |Total
|-----------------------------------------------------------------------------
|  92.4% | 29,242.0 |        -- |     -- |MPI
||----------------------------------------------------------------------------
||  65.7% | 20,797.2 |     877.8 |  4.1% |MPI_ALLGATHERV
||  21.3% |  6,740.0 |  1,249.0 | 15.6% |mpi_bcast
||   3.1% |    989.7 |  1,943.3 | 66.3% |MPI_REDUCE
||   1.8% |    559.6 |    356.4 | 38.9% |MPI_ALLREDUCE
||============================================================================
|   5.9% |  1,876.7 |        -- |     -- |USER
||----------------------------------------------------------------------------
||   2.2% |    680.7 | 19,973.3 | 96.7% |mat_inv_mp_inverse_gj_
||   1.0% |    313.3 |    141.7 | 31.1%
|redistribute_mp_c_redist_33_mpi_copy_nonblock_
||============================================================================
|   1.6% |    494.2 |        -- |     -- |ETC
|=============================================================================
```