http://dx.doi.org/10.1090/conm/218/03054

Contemporary Mathematics
Volume **218**, 1998
B 0-8218-0988-1-03054-5

Parallel Computing for Reacting Flows Using Adaptive Grid Refinement

Robbert L. Verweij, Aris Twerda, and Tim W.J. Peeters

1. Introduction

The paper reports on the parallelisation of the computational code for modelling turbulent combustion in large industrial 3D glass melting furnaces. Domain decomposition is used to perform the parallelisation. Performance optimisation is examined, to both speed up the code as well as to improve convergence. Local grid refinement is discussed using several different refinement criteria.

The numerical simulation of turbulent reacting flows requires advanced models of turbulence, combustion and radiation in conjunction with sufficiently fine numerical grids to resolve important small scale interactions in the areas of flame front, high shear and near solid walls. These simulations are very CPU- and memory-demanding. Currently, many of the numerical simulation have to be performed with relatively simple models, hampering an accurate prediction. Parallel processing is regarded nowadays as the promising route by which to achieve desired accuracy with acceptable turn-around time.

Domain decomposition is very suitable technique to achieve a parallel algorithm. Furthermore, it allows block-structured refinement quite easily. In this manner the number of grid points can be minimised, giving rise to a very efficient distribution of grid points over the domain.

2. Physical model

Figure 1 shows a typical furnace geometry, where the preheated air ($T = 1400$ K, $v = 9$ m/s) and the gas ($T = 300$ K, $v = 125$ m/s) enter the furnace separately. The turbulence, mainly occurring because of the high gas-inlet velocity, leads to good turbulent mixing. This mixing is essential for combustion of the initially non-premixed fuel and oxidiser into products, which exit the furnace at the opposite side.

The maximum time-averaged temperatures encountered in the furnace are typically 2200 K, at which temperature most of the heat transfer to the walls is by radiation.

The implementation involves the solving of the 3D incompressible (variable density) stationary conservation laws for mass, momentum, energy and species. The hydrodynamic and caloric equations of state are added to relate the pressure

---

1991 *Mathematics Subject Classification.* Primary 65N55; Secondary 65Y05, 76T05.

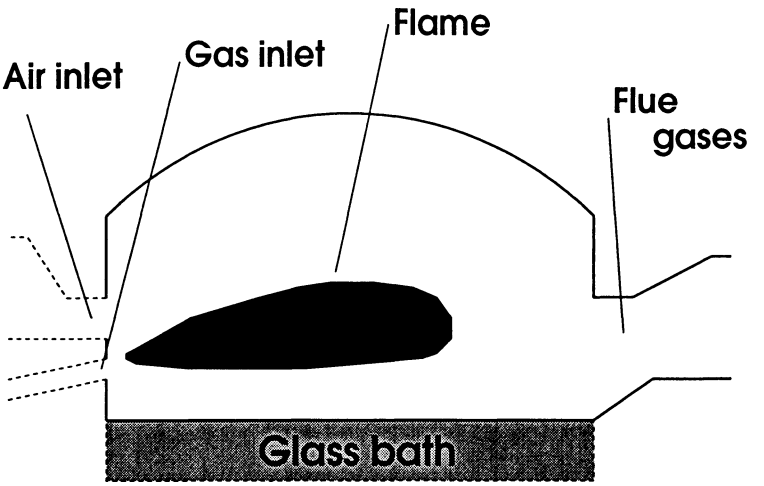

FIGURE 1. Artist's impression of a furnace geometry with flame.

to the density and the enthalpy to the temperature. The equations are averaged using Favre-averaging. The mean flow equations are closed using the standard high-Reynolds $k-\varepsilon$ turbulence model with standard values for all constants. Wall functions are used to bridge the low-Reynolds region near the walls. The heat transfer to the walls is not too much influenced by the velocity field in the near-wall region, justifying this assumption. The conserved-scalar approach is used to model the combustion. The chemistry is modelled with a constrained-equilibrium assumption. An assumed shape β Probability Density Functions (PDF) is used to obtain the mean values of the thermochemical quantities. Radiative heat transfer in the furnace is calculated using the Discrete Transfer Model. A more detailed description of turbulent combustion modelling for furnaces can be found in [**1, 6**].

3. Numerical model

The modelled equations are discretised using the Finite Volume Method on a colocated, Cartesian grid [**4**]. The SIMPLE-algorithm is applied to couple the pressure and velocity fields and satisfy mass- and momentum conservation. The linearised systems are solved using the SIP-algorithm or the GMRES-algorithm. The convective terms are discretised using central discretisation, as suggested by [**4**], but other convective schemes (upwind, tvd) are also available. Full multi-grid [**4**] is used to improve the convergence behaviour of the multi-block code.

For the domain decomposition the grid-embedding technique [**3**] is used. This means that one global (coarse) grid is defined, and the domains are defined as subdomains of this coarse grid, where minimal overlap between the blocks is used. Every subdomain can be refined with respect to the coarse grid, with the restriction that adjacent blocks can only be equally fine, twice as fine or twice as coarse as the neighbouring blocks. This is not really a restriction, since the truncation error over the interface is of the order of the size of the coarser block, so patching a very fine block to a very coarse block would not lead to better numerical accuracy. A

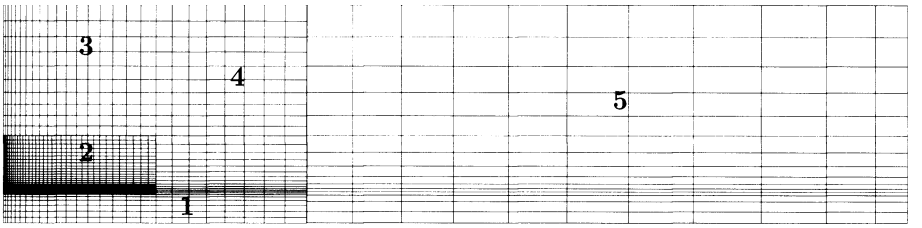

FIGURE 2. Two-dimensional view of a domain decomposition into five blocks with three different levels of grid refinement. For clarity the overlap areas between the blocks have not been drawn.

2D slice of a typical domain decomposition with local grid refinement is shown in Figure 2.

To couple the domains, one layer of halo-cells (auxiliary control volumes) was used [7]. The values on the internal boundaries are copied into the halo-cells of the neighbouring domains. This way of coupling renders the need for explicit boundary conditions on the internal boundaries superfluous and guarantees that the converged solution is independent of the block decomposition, if all blocks are equally fine.

The local grid refinement was implemented by computing the fluxes on the fine grid side of the interfaces only, and then sending them to the coarser grids and adding them there. This yields a flux conservative scheme over the block-interface. For the other quantities, tri-linear interpolation and splines are used. This also uniquely defines the value of the gradients on the interfaces.

4. Parallel implementation

The domain decomposition was used as a basis for parallelisation. Static load balancing is performed; every domain contains (approximately) the same amount of grid-points and the amount of work per grid point was assumed to be constant for all points. All processors were assumed to be equally fast. In combination with local grid refinement some load-imbalance was accepted to minimise the number of blocks and optimise the number of grid points needed.

Exactly one domain is computed per processor, so that the (old) sequential single-block program could easily be used for multi-block computations. The SPMD (Single Program, Multiple data) programming model was used. Typically 4 to 32 domains were used to perform the calculations.

In this study, PVM and MPI were adopted on clustered workstations and the machine specific message passing tool SHMEM on the CRAY T3E. Changing from one message passing interface to another proved easy, since all communication was hidden in a few generic subroutines, named for example *parsend* or *parrecv*. In domain decomposition the number and size of messages is relatively small compared to the computations done. Hence switching from MPI to SHMEM didn't yield significantly better results.

5. Results

5.1. Performance optimisation. First the speedup of the code was examined. The 3D version of the code, called FURNACE is shown in Figure 3 on the left for two grid-sizes (20^3 and 50^3). On the right the speedup for a 2D version of the

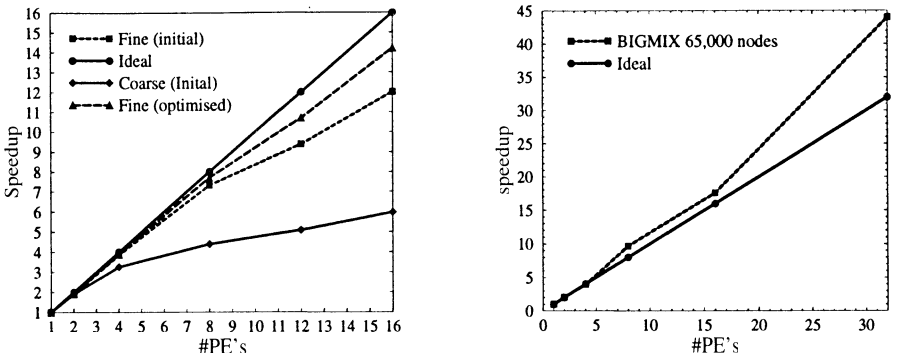

FIGURE 3. Speedup for several turbulent combustion codes **left**: 3D **right**: 2D

code, called BIGMIX, with more elaborate chemistry [**6**] is shown. All test-runs have been performed on a CRAY T3E AC80-128.

Note that FURNACE scales better for finer grids, as is expected. The super-linear speedup of BIGMIX can be explained by assuming better cache-use, because of the two dimensionality of the arrays. Triggered by the sensitivity of the code performance on the cache-use, the original code was slightly rewritten, and optimised to enable better vector-length controlling. Compiler-options were investigated and it was found out that `-O3,Unroll2,nojump` together with the enabling of the CRAY T3E data-stream buffers significantly improved the singe PE performance as well as the speedup, since there was less load-imbalance. This is shown in Figure 3 (left) too.

The effect of load imbalance can also be seen in the weird bend in the speedup curve in the left figure around 8 and 12 PE's, where speedup seems to decrease and then increase again, especially for the coarse grid. This bend been analysed by plotting the time spent in different parts of FURNACE. It can be seen from Figure 4 that when using more blocks the amount of time spent in updating the internal boundaries (which requires communication) becomes the bottleneck. This is probably due to the fact that from if there are more than 8 blocks in the decomposition, some blocks have more neighbours than others, which implies that some blocks will updating more boundaries than others. This assumption is asserted in the next paragraph.

Further time-reduction was obtained by minimising the amount of I/O requests. I/O is very expensive and still highly sequential on parallel machines (although recently MPI-2 defines a standard to optimise this, which has not been used here). In general it was found that if a data-block needs to be read in by all processors, it is the quickest to let 1 PE read the data and then global scatter it to all other PE's.

TABLE 1. Load imbalance for complete problem

Time needed (s)	sliced decomp.		rectangular decomp.	
Total program time	2150		1700	
Parallel working time	1140	(53%)	1241	(73%)
Total barrier waiting time	606	(28%)	106	(6%)
Total communication time	85	(4%)	77	(5%)

TABLE 2. Relative change of some variables with respect to their finest grid values

Grid size	heat flux	flue temp.	flue O_2 conc.	flue NO_x conc.
$16 \times 24 \times 20$	+ 17%	+ 2%	+ 2%	-33%
$24 \times 36 \times 30$	+ 12%	+ 3%	+ 1%	-25%
$32 \times 48 \times 40$	+ 8%	+ 3%	+ 1%	-12%
$32 \times 72 \times 60$	+ 0%	+ 0%	+ 0%	+ 0%

In the original code every variable could be read from file independently, causing much I/O interrupts. In the modified code, all data was available in a single data-file, and the amount of file checking was minimised. The total amount of I/O time for both methods is depicted in Figure 4 for a run on 16 PE's, the total amount of data to be read from disk is $16 \times 5.4Mb$ (this includes the lookup table for chemistry) The amount of data written to disk is 16×1.4Mb. The differences per PE are much smaller in the modified code, and the mean time spent in I/O was brought down from 197.1 seconds to 15.4 seconds.

The assumption that load imbalance was caused by the fact that some blocks have more neighbours than others was asserted next. 200 iterations were done on a coarse ($16 \times 24 \times 20$) grid, used in previous studies [**1, 6**]. This grid was split into four blocks in two different configurations, shown in Figure 5. In the left decomposition all blocks have exactly two neighbours (the rectangular decomposition), in the right one the middle two blocks have two neighbours (the sliced decomposition), and the outer blocks one. Every block contains the same amount of points. Both decompositions lead to exactly the same converged solution, but the timings are quite different, as shown in Table 1.

The rectangular decomposition yields much better timing results than the sliced decomposition. This confirms the assumption that in the sliced decomposition, the two middle blocks have much more communications to do, creating an huge load-imbalance in that part of the code. This effect becomes smaller if the number of points per blocks becomes bigger. Note that the total communication time is approximately equal in both cases, and is small compared to the entire program time. The smaller parallel working time in the rectangular decomposition is because of better cache use, we assume. The load imbalance becomes smaller if the number of points per block become larger, as could be seen in Figure 4.

5.2. Local grid refinement. The block-structured approach allows for local grid refinement in each block, as was already explained in the previous section. The need for local grid refinement becomes clear from Table 2, where the relative change of some variables with respect to the value of the variable on the finest grid is printed, if the global grid is varied from relatively coarse to very fine.

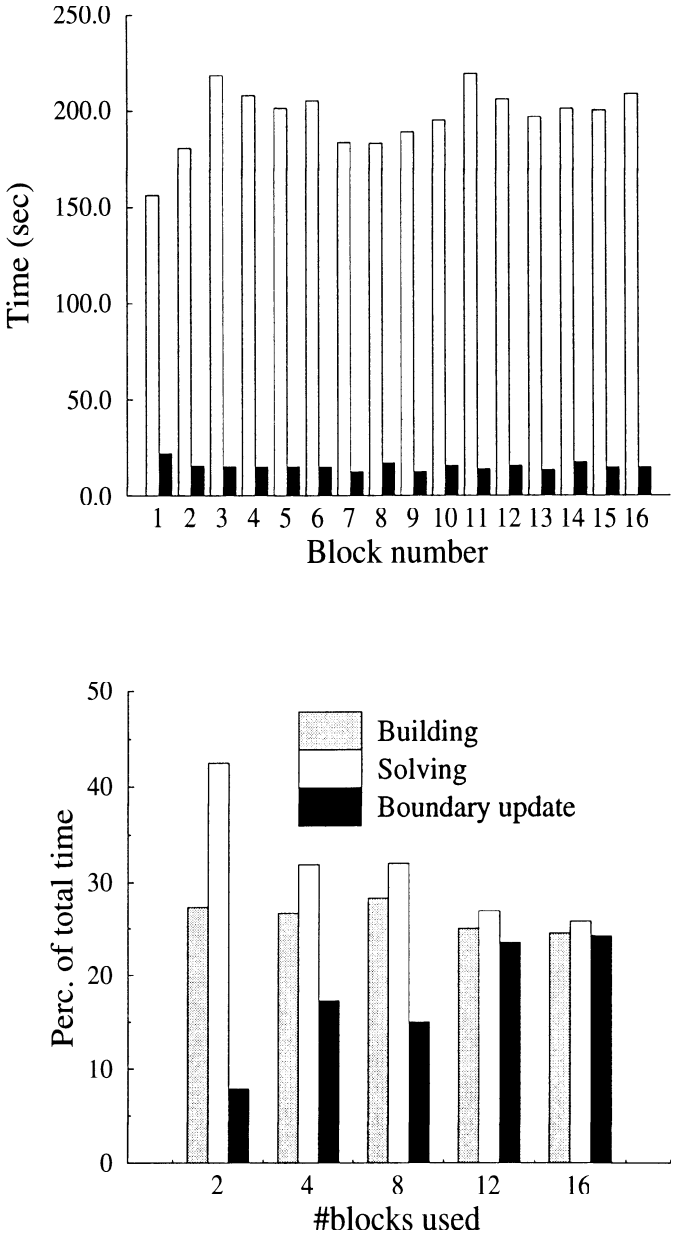

FIGURE 4. **top**: Total time spent in I/O for each PE **bottom**: Percentage of time spent in different parts of the code for different number of blocks

Although the temperature and the heat flux are not so much influenced by finer grids, the predictions for the concentrations show a great grid sensitivity, mainly because they depend heavily on the mixture fraction concentration prediction, which also varies considerably.

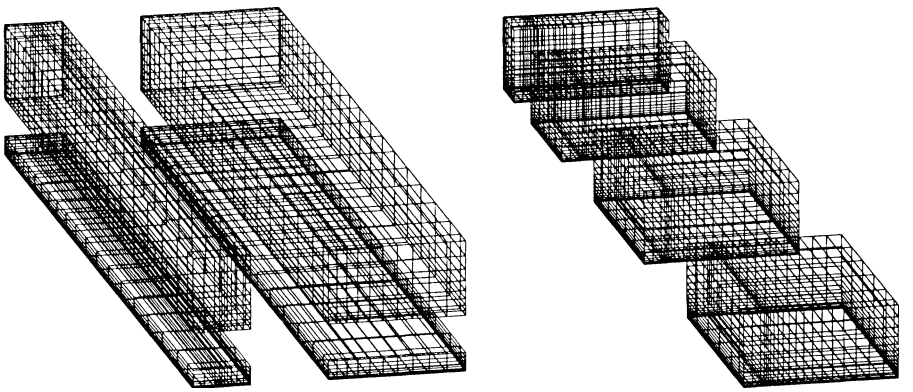

FIGURE 5. Two block decompositions of the same grid. **left**: Rectangular decomposition **right**: Sliced decomposition.

Also another, standard, testcase was performed: The laminar lid-driven cavity at $Re = 100$. This testcase was also used by [**5**] to test his dynamic local grid refinement. Also here grid independent results were only obtained at a 256^2 grid, in agreement with [**8**], showing the amount of grid points needed without local refinement.

Statical grid refinement was applied first; the blocks where manually generated, based on experience and intuition rather than some fundamental refinement criterion. This did yield satisfactory results, in the sense that first results yielded reduction of grid points from 38 % for the entire combustion computations [**1**] to 80 % for the cavity.

Next, dynamical grid refinement was implemented. Some error estimator should provide information about the refinement. In literature, the Richardson extrapolation is often used as criterion for refinement. However, for combustion problems, where geometry and physics are complex, the coarsest reliable mesh contains so many grid points that a uniform refinement, just to estimate the error, cannot be afforded. Muzaferia [**5**] developed a method, similar to Richardson extrapolation, which is based on the difference of higher- and lower order approximation of the fluxes to approximate the truncation error. Apart from this method, a more physical criterion was build in, to reflect our believe that in complex flows steep gradients of relevant quantities also mark regions which should be refined [**2**]. Hence the gradient of an 'interesting quantity' is used as an error indication. In the furnace-simulation another interesting quantity like temperature or density could be used. However, in most flows it is not the gradient, but the curvature (the second derivative) which yields the areas of high shear. These three criteria have been implemented and tested on several flows.

To test the different criteria the laminar lid-driven cavity was computed, to compare the results to [**5**], and since this problem converges very quickly at low Reynolds numbers. Grid independent results were only obtained at a 256^2 grid, in agreement with [**8**], showing the amount of grid points that are needed without local refinement. The results are in excellent agreement with those of [**5**] and [**8**]. However, switching to other criteria gave different results. Figure 6 shows the

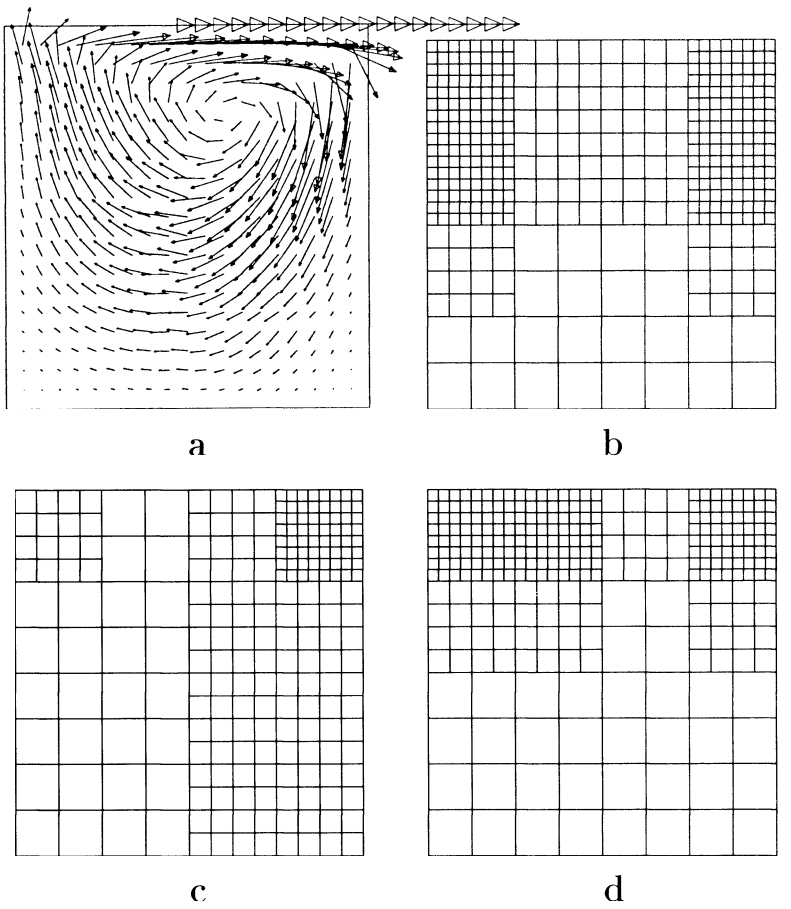

FIGURE 6. Grids, obtained with different local refinement criteria **a** Vectorplot of field **b** Maximum curvature **c** Method of [**5**] and present method **d** Maximum velocity-gradient criterion

different grid refinements after 2 levels of refinement starting with a uniform 8^2 grid, and allowing 4 blocks in both directions.

The results for refinement criterion c (the Muzaferia method) are in excellent agreement with those of [**5**] and [**8**]. All three criteria yield plausible refinement regions at first view and comparable minimum values for the streamlines. More testcases need to be considered to determine which criterion yields the most accurate results by comparing them to the 'true' solution, ie. the grid independent solution.

A backdraw of the block-structured refinement is that it leads to severe load imbalance, as shown in Table 3. This could be overcome by using a different way of defining the new domain decomposition. Currently the decomposition is based on a minimal number of blocks. Other options might be to use many very small blocks (although this will probably influence the convergence behaviour) or to run several blocks on 1 PE, which is currently not possible.

TABLE 3. Load imbalance due to local grid refinement.

		grid points		
Criterion	# blocks	Max	Min	Total
Muzaferia	6	128	4	220
Gradient	7	256	4	292
Curvature	7	128	8	376

6. Conclusions

Load imbalance is a underestimated problem in most CFD applications. Parallelisation by domain decomposition is a straightforward and efficient way to parallelise combustion codes. When combined with multigrid, convergence becomes independent of the number of blocks used. Good tuning of a program significantly improves the performance on cache-based machines like the CRAY T3E.

Dynamical grid refinement yields significant reduction of grid points with the same accuracy. This is needed since tests showed the need of huge grids for realistic applications. Different criteria can be applied to determine the refinement, yielding different regions of refinement. The choice for the 'best' criterion is not obvious yet, and might even depend on the application.

References

1. G.P. Boerstoel, *Modelling of gas-fired furnaces*, Ph.D. thesis, TU Delft, may 1997.
2. W.L. Chen, F.S. Lien, and M.A. Leschziner, *Local mesh refinement within a multi-block structured grid scheme for general flows*, Comp. Meth. Appl. Mech. Eng. **144** (1997), no. 3, 327–327.
3. P. Coelho, J.C.F. Pereira, and M.G. Carvalho, *Calculation of laminar recirculating flows using a local non-staggered grid refinement system.*, Int. J. Num. Meth. Fl. **12** (1991), 535–557.
4. J.H. Ferziger and M Perić, *Computational methods for fluid dynamics*, Springer-Verlag, Berlin, 1996.
5. S. Muzaferia and D. Gosman, *An adaptive finite-volume discretisation technique for unstructured grids*, Tech. Report Ms. No. G0353, Dept. of Mech. Eng., Imperial College of Science, Technology and Medicine, 1997.
6. T.W.J. Peeters, *Numerical modeling of turbulent natural-gas diffusion flames.*, Ph.D. thesis, TU Delft, September 1995.
7. M. Perić, M. Schäfer, and E. Schreck, *Computation of fluid flow with a parallel multigrid solver.*, Parallel Computational Fluid Dynamics 1991, Elsevier Science Publishers. B.V., 1992, pp. 297–312.
8. M.C. Thompson and J.H. Ferziger, *An adaptive multigrid technique for the incompressible navier Stokes equations*, Journ. Comp. Phys. **82** (1989), 94–121.

DEPARTMENT OF APPLIED PHYSICS, DELFT UNIVERSITY OF TECHNOLOGY, LORENTZWEG 1, 2628 CJ DELFT, THE NETHERLANDS
E-mail address: R.Verweij@tn.tudelft.nl

DEPARTMENT OF APPLIED PHYSICS, DELFT UNIVERSITY OF TECHNOLOGY, LORENTZWEG 1, 2628 CJ DELFT, THE NETHERLANDS
E-mail address: A.Twerda@tn.tudelft.nl

DEPARTMENT OF APPLIED PHYSICS, DELFT UNIVERSITY OF TECHNOLOGY, LORENTZWEG 1, 2628 CJ DELFT, THE NETHERLANDS
E-mail address: T.W.J.Peeters@tn.tudelft.nl