Presented at the 14th Annual CMAS Conference, Chapel Hill, NC, October 5-7 2015

A PARALLEL SPARSE MATRIX IMPLEMENTATION OF THE GEAR SOLVER FOR THE GMI MODEL WITH RESULTS FOR NEW PARALLEL DEVICES

T. Clune, NASA Goddard Space Flight Center, MS 610, Greenbelt, MD 20771, Megan R. Damon, NASA Goddard Space Flight Center and Science Systems and Applications, Inc.MC 606, Greenbelt, MD 2077 and, George Delic* HiPERiSM Consulting, LLC, P.O. Box 569, Chapel Hill, NC 27514, USA

1. INTRODUCTION

The Global Modeling Initiative (GMI) [GMI] is part of the NASA Modeling Analysis and Prediction (MAP) program [MAP]. GMI investigations support the development and integration of a state-of-the-art modular 3-D chemistry and transport model (CTM) that includes full chemistry for both the troposphere and stratosphere. The GMI model is involved in assessment of anthropogenic impacts, such as those from aircraft, future changes in atmospheric composition, and the role of long-range transport of pollution.

The GMI model serves as a testbed for different meteorological fields, emissions, microphysical mechanisms, chemical mechanisms, deposition schemes, and other processes determining atmospheric composition, both gas-phase and aerosol. This enables GMI to work in close collaboration with the chemistry-climate modeling community. GMI seeks to understand and constrain the uncertainties in model results through inter-comparison of simulations and through comparison with observations.

To describe production and loss of chemical species in reaction mechanisms in the CTM, GMI implements the SMVGear solver algorithm for integration of a sparse stiff system of ordinary differential equations based on the work of Jacobson and Turco [Jacobson, 1994] for the JSparse method.

The HiPERiSM version replaces the legacy sparse matrix methodology of Jacobson and Turco by a more modern one (FSparse) described by Delic [2013, 2014]. This work reports on this replacement in the GMI CTM code with performance results on Intel's [INTEL] Ev5 CPU and Phi many integrated core (MIC) commodity architecture.

2. TEST BED ENVIRONMENT

2.1 Hardware

The hardware systems chosen were the platforms at HiPERiSM Consulting, LLC, shown in Table 2.1. Each node hosts two Intel E5v3 CPUs with 16 cores each. In addition each has four Intel Phi co-processor (MIC) cards with 61 and 60 cores for the respective models. With four MIC cards per node the total (usable) thread count is 960 and 944, respectively. This cluster is used for either MPI only, or hybrid thread-parallel OpenMP plus MPI execution. In this application, when the Phi cards are in use, each MPI process offloads its own OpenMP parallel region to a Phi co-processor. But this is not the only way to use Intel Phi architectures and other examples of successful utilization of such hybrid systems may be found in Reinders [Reinders, 2013, 2015].

The CPU and MIC architecture supports AVX 2 and FMA instructions and peak performance is only attainable if the full potential for FMA vector instructions is uncovered. Peak Gflop/s performance for either CPU, or Phi card, is calculated from the formula: FMA concurrency (=2) x number of cores x vector length (= 8 SP words) x processor speed. Respectively, this is

CPU:	2 x 16 x 8 x 2.3 = 589 Gflop/s
Phi 7120:	2 x 61 x 8 x 1.238 = 1208 Gflop/s

Thus, one MIC card has double the peak performance potential of a CPU. However, for performance to approach this value on a MIC card, with a maximum bandwidth of 352 GB/s (44 Gword/s), an algorithm must reach 1208/44=28 operations per word. This requires careful memory management and arithmetic optimizations (see Chapter 27 in [Reinders 2015]). These peak values are not reached in the work reported here, but further optimization work is in progress.

* *Corresponding author:* George Delic, george@hiperism.com.

2.2 Compilers

This report implemented the Intel compiler (release 15.0) for the hardware shown in Table 2.1. For MIC targets this compiler enables two important optimizations in the MIC environment: FMA vector and hardware gather/scatter instructions. Furthermore, specific compiler options and environment variables allow the selection of the number of threads per core, and MIC-specific optimizations for the offloaded OpenMP parallel region. Exploring these features is important in approaching peak Gflop/s rates.

2.3 Episode studied

For all GMI results reported here the model episode selected was for December 01, 2011, using data provided by NASA GSFC. This episode has 124 active chemical species. The episode was run for a full 24 hour simulation on a 144 X 91 x 72 global domain for a total of 0.94 million grid cells. The cells are partitioned amongst the number of MPI processes. In the original version of the GMI JSparse model these cells are processed in blocks of 20 by SMVGear in the CTM. In this work, for the "inlined" version of the original GMI and the FSparse version, each cell is processed individually in SMVGear.

Table 2.1. CPU platforms at HiPERiSM Consulting, LLC

Platform	Node20	Node21
Operating system	SuSE Linux 13.2	SuSE Linux 13.2
Processor	Intel™ IA32 (E5-2698v3)	Intel™ IA32 (E5-2698v3)
Coprocessor	4 x Intel Phi 7120	4 x Intel Phi 5110
Peak Gflop/s (SP) per processor	589	589
CPU power consumption	135 Watts	135 Watts
Cores per processor	16	16
Power per core	8.44 Watts	8.44 Watts
Processor count	2	2
Total core count	32	32
Clock	2.3 GHz	2.3 GHz
Bandwidth	68.0GB/sec	68GB/sec
Bus speed	2133 MHz	2133 MHz
L1 cache	16x32 KB	16x32 KB
L2 cache	16x256 MB	16x256 KB
L3 cache	40 MB	40 MB

3. GMI SMVGear MODEL

3.1 Original GMI version

For each block of 20 cells the chemistry-transport model (CTM) in in the original GMI version processes a nested loop structure shown schematically as follows:

```
!$omp parallel
!$omp do schedule( dynamic, my_chunk )
      do kblk  = 1, nblockuse
! each thread takes it own kblk value
! to perform SMVGEAR on a block of cells
       …………
       End do
```

Each MPI process performs this OpenMP parallel loop for both values of the day/night index (iday). As an example, with four MPI processes (NP=0-3), there are eight calls to SMVGear per simulation time step, or 8 x 24 =192 calls in a 24 hour simulation. An example with 1 OpenMP thread is shown in Table 2.2 with the loop range (nblockuse) and the corresponding processing time. A monitor of progress shows that some calls to SMVGear finish before others (wait status) while others continue executing (run status). Therefore completion time for the SMVGear call is determined by the longest running call. This results in a load imbalance due to the synchronization required at the end of the simulation time step.

Table 2.2. Typical CPU times for original GMI

NP	iday	nblockuse	clock tics(1)	Status
3	1	3773	229501	! wait
2	1	4439	296057	! wait
3	2	7892	282953	! run
2	2	7226	256024	! run
1	1	9000	546253	! run
0	1	9540	572869	! run
0	2	2384	72800	! wait
1	2	2924	113219	! wait
(1) Time unit returned with KIND=4 arguments in system_clock function				

3.2 FSparse GMI version

The FSparse version replaces the call to SMVGear over a block of cells with calls for each individual cell. This has two consequences. First some vectorization potential is relinquished, but with a sufficient number of threads in the OpenMP thread team, this loss in performance is recovered. The second consequence is that the precision of the SMVGear solution is greatly improved as is demonstrated in the next section.

3.3 Performance profile of GMI

Enabling the internal timing calls in the GMI shows the distribution of runtime spent in the various physical processes during execution. Fig. 3.1 shows the case of execution with offload to the Intel Phi devices for execution with MPI processes 4, 8, 16, and 32. The legend distinguishes these and in each case the parentheses indicates the total thread count and number of threads per core. In all cases some 60% of the wall clock time is accounted for by the CTM.

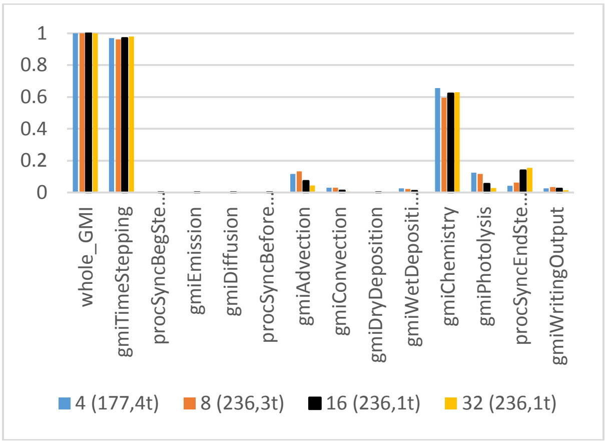

Fig 3.1. In the FSparse GMI version this shows the fraction of total average component wall clock time vs MPI-rank (MIC thread count, threads-per-core) in offload mode on 4 x 7120P + 4 x 5110 Phi coprocessors: with 2 x host E5-2698v3 CPUs each.

4. NUMERICAL PRECISION COMPARISON

4.1 FSparse and inlined JSparse on host

Of the 124 species in the data set, concentration values of the 14 species in this list: `CH2O, CH4, CO, HNO3, HO2, H2O2, MP, NO, NO2, N2O5, O3, INO2 PAN, SYNOZ`
have been compared for FSparse and an inlined JSparse on the node20+node21 hosts. The inlined version is a modification of the original to force processing in SMVGear on individual grid cells to bring the convergence criterion in conformance with that in FSparse. For all 24 simulation time steps, the comparison is for all ilat x ilong x 24 = 314496 values in the first layer of the grid in the form of GNU plot graphics with a sort from smallest to largest concentration value. Typical results are those in Fig. 4.1 for O3. When comparing the same results for FSparse executing with offload to the Intel Phi the results are identical. The observed differences are of the order of the error tolerances used in the SMVGear convergence criteria and may be assumed to be negligible.

4.2 Inlined and original JSparse on host

When making the same species concentration comparison for JSparse in two versions: inlined and original, the comparison is typically like that shown in Fig. 4.2.

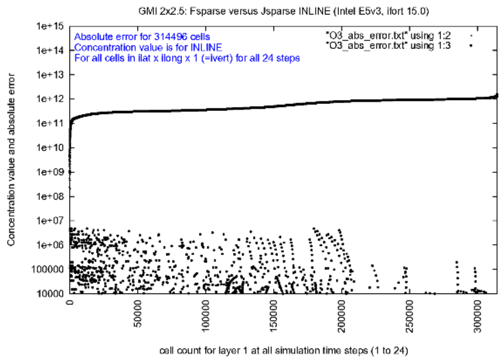

Fig 4.1. On a log scale this shows the absolute error in comparing O3 species concentration predictions of FSparse and inlined JSparse for 314496 values. The curve is the concentration value sorted on increasing size.

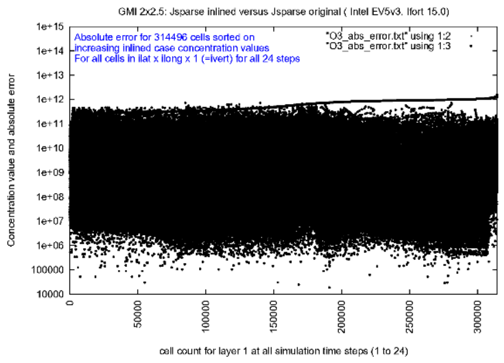

Fig 4.2. On a log scale this shows the absolute error in comparing O3 species concentration predictions of JSparse in inlined and original versions for 314496 values. The curve is the concentration value sorted on increasing size.

The results of the comparison in Fig. 4.2 show observed differences that are large and often exceed the value of the species concentrations. In view of the sensible values for the comparisons of Fig. 4.1, it has to be assumed that these divergences in precision originate in the original (legacy) version of JSparse and must be related to the number of grid cells per block. The original (legacy) version of JSparse in the GMI model uses 20 cells in a block. This means that the error tolerance criterion calculation in the SMVGear algorithm uses the RMS error over this block of 20 cells. Improvement in accuracy was confirmed by reducing the number of cells in a block and repeating the comparison to see a reduction in the absolute error. On this basis either the inlined JSparse or FSparse versions of GMI are considered definitive.

5. PERFORMANCE OF GMI

5.1 CPU core demand in hybrid mode

This section present results for a combination of runs in a 24 hour simulation with differing MPI rank and OpenMP thread counts. Table 5.1 summarized the core count demanded in each combination for host CPUs. Those cases where the core demand exceeds the available core count of 64 (across both nodes) are color coded.

Table 5.1: Core demand on the host nodes for combinations of MPI rank (column) and OpenMP thread count (row). Color coding indicates where the core count is oversubscribed (i.e. above the total of 64 available)

	4	**8**	**16**	**24**	**32**	**40**	**48**	**64**
1	4	8	16	24	32	40	48	64
2	8	16	32	48	64	80	96	**128**
4	16	32	64	96	**128**	160	192	256
6	24	48	96	144	192	240	288	384
8	32	64	**128**	192	256	320	384	512

Fig. 5.1 shows the FSparse scaling results for all combinations in Table 5.1 and there is a peak at 128. This corresponds to thread x MPI process counts of: 8 x 16, 4 x 32, and 2 x 64. These results suggest that it is possible to oversubscribe the available core count by as much as a factor of 2 and still have an improvement in performance.

5.2 Performance on host CPUs

For execution on the host CPUs Fig. 5.2 shows the wall clock times for FSparse as a function of increasing MPI process count. For MPI ranks upto to 40 there is a steady improvement as thread count increases, but is bounded by the limits in Table 5.1.

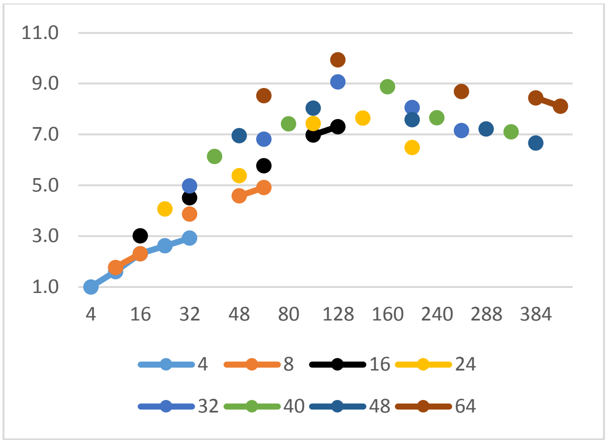

Fig 5.1. This shows FSparse scaling (over the case of the 4 MPI ranks and 1 OpenMP thread) versus the number of cores demanded (see Table 5.1), with 1 to 8 OpenMP threads and MPI ranks 4 to 64 (see legend).

For the original JSparse version Fig. 5.3 shows the corresponding results (with changes in vertical and horizontal scales). This improvement in wallclock times is in contrast to either the FSparse or inlined JSparse versions which use a single cell per block in both cases. For an easier comparison two cases with MPI ranks of 4 and 8 from Figs. 5.2 and 5.3 are shown in Fig. 5.4 as a function of the thread count. The FSparse execution is in no-offload mode, i.e. on the host CPUs only. At the highest thread count (8) JSparse and FSparse results approach each other asymptotically. However, The effects of reducing the cell block count from 20 to 2 in the original JSparse version shows a steady increase in wall clock time as is demonstrated in in Fig. 5.5 where the dilation in wall clock time is ~1.3 (rank 4) and ~1.5 (rank 8), respectively as the cell count per block reduces from 20 to 2.

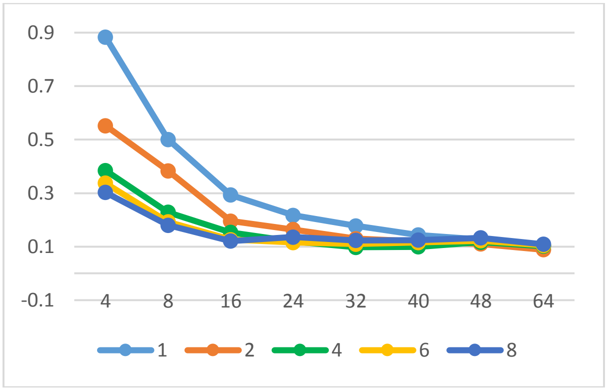

Fig 5.2. Wall clock time (hours) versus MPI process count for FSparse on host CPUs for OpenMP thread counts from 1 to 8 as shown in the legend.

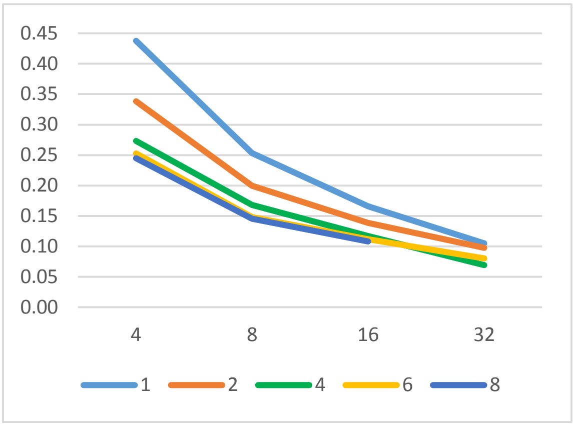

Fig 5.3. Wall clock time (hours) versus MPI process count for the original JSparse version on host CPUs for OpenMP thread counts from 1 to 8 as shown in the legend.

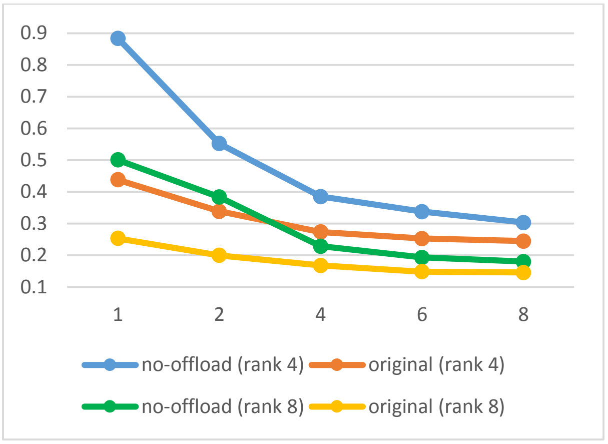

Fig 5.4. For the FSparse (no-offload) and original JSparse versions this shows the wall clock time (hours) versus the number of OpenMP threads. Each pair of curves corresponds to a different choice for the number of MPI ranks (4 or 8) as identified in the legend.

5.3 Performance with Phi cards

When offload to the Intel Phi is enabled the results for FSparse are those shown in Fig. 5.6 as a function of MIC thread count and again in Fig. 5.7 as a function of MPI process count. The latter demonstrates that the optimal utilization of the Phi architecture in offload mode occurs when each MPI rank has its own dedicated MIC card. This occurs when the MPI rank is 8, with 4 ranks per node. The combined results of Figs. 5.6 and 5.7 suggest that if two MPI ranks could utilize 120 threads on each card, double the number of MPI ranks (i.e. 16) could still deliver optimal performance. Testing of this option is in progress.

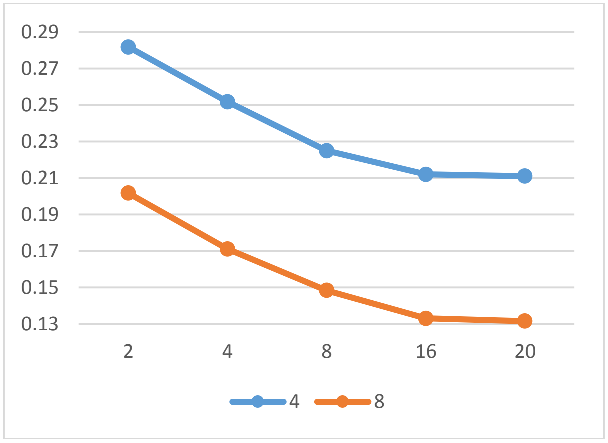

Fig 5.5. For the original JSparse version this shows the wall clock time (hours) versus the number of cells in a block. Each curve corresponds to a different choice for the number of MPI ranks (4 or 8) as identified in the legend.

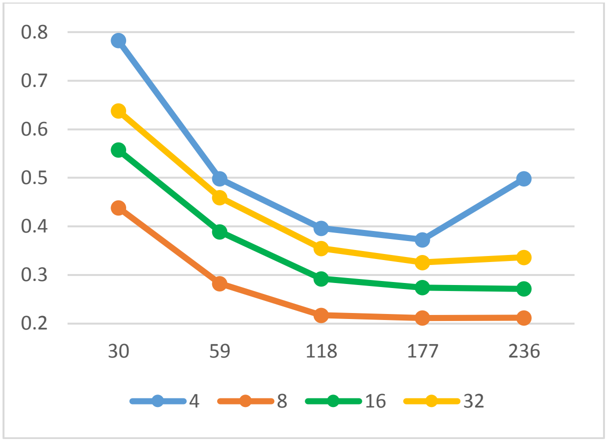

Fig 5.6. This shows the wall clock time versus the number of OpenMP threads for FSparse in offload mode to the Intel Phi architecture. Each curve corresponds to a different choice for the number of MPI ranks (4 to 32) as identified in the legend.

In a final example, results of four runtime modes for FSparse and the legacy JSparse versions are compared in Fig. 5.8. The two modes for FSparse are those with and without offload (offload/no-offload), and the two modes for JSparse are original and inlined. In Fig. 5.8 the legacy JSparse version delivers the lowest values on wall clock time, and the inlined JSparse version the highest. The two cases in between are for FSparse with and without offload, and the best times of these two modes are for the latter, with the highest thread count (8). However, it should be noted that the synchronization of thread count scales in Fig. 5.8 between these two FSparse modes is arbitrary.

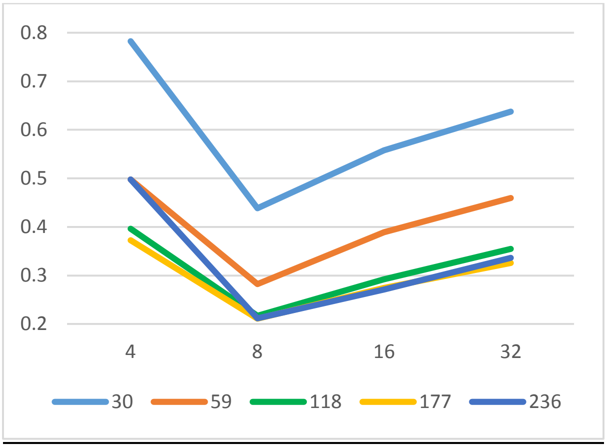

Fig 5.7. This shows the wall clock time versus the number of MPI ranks for FSparse in offload mode to the Intel Phi architecture. Each curve corresponds to a different choice for the number of OpenMP threads (30 to 236) as identified in the legend.

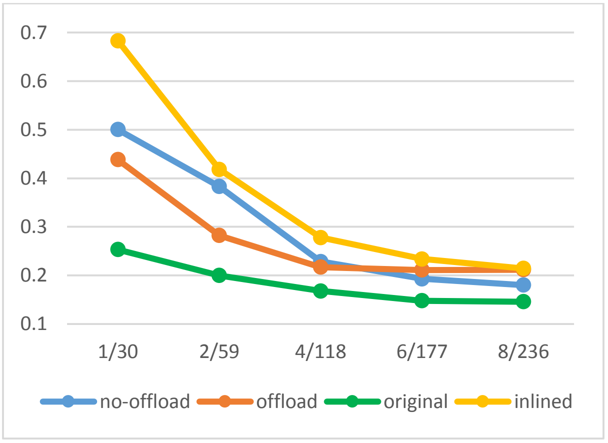

Fig 5.8. This shows the wall clock time versus the number of OpenMP threads for FSparse in no-offload (host only), offload mode (host and Intel Phi), and JSparse in the original and inlined versions (on the host only). Each curve corresponds to these respective choices of the mode. The horizontal scale corresponds to a thread count on either the host (1 to 8 threads), or on the Intel Phi (30 to 236 threads).

6. CONCLUSIONS

6.1 Benefits of the FSPARSE method

A comparison of GMI in original and FSparse versions showed these benefits for FSparse:

- Easy porting to either host CPU or attached MIC devices with the same code.
- Good performance scaling with MPI rank or OpenMP thread count.
- Superior numerical precision.

6.2 Numerical precision issues

A comparison of numerical precision for GMI in JSparse and FSparse versions suggests that:

- The blocking of cells into groups required in JSparse results in inferior precision.
- Application to individual cells in FSparse enhances precision by many orders of magnitude.
- For the same precision FSparse runtime is less than (inlined) JSparse.

6.3 Future work

Further opportunities remain for thread parallelism by:

- Enhancing MIC vector performance.
- Enabling nested thread parallelism.

ACKNOWLEDGEMENT

Work effort performed by HiPERiSM Consulting, LLC, as Subcontractor/Consultant under a Task Order issued by the National Aeronautics and Space Administration, Goddard Space Flight Center, MD.

REFERENCES

GMI. http://gmi.gsfc.nasa.gov/
MAP, http://map.nasa.gov/

Jacobson, M. and Turco, R.P., (1994), Atmos. Environ. 28, 273-284.

Delic, G., 2013: contribution to 12th Annual CMAS Conference, Chapel Hill, NC, October 28-30, 2013, https://www.cmascenter.org/conference/2013/agenda.cfm)
Delic, G., 2014: presented at the 8th International Workshop on Parallel Matrix Algorithms and Applications (PMAA14), July 2-4, Università della Svizzera italiana // Lugano, Switzerland.

INTEL: Intel Corporation, http://www.intel.com.
Reinders, **2013**, Intel Xeon® Phi™ Coprocessor High-Performance Programming, James Reinders and Jim Jeffers,Morgan Kaufmann/Elsevier, Waltham, MA, 2013.
Reinders 2015, High-Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, James Reinders and Jim Jeffers,Morgan Kaufmann/Elsevier, Waltham, MA, 2015.