The purpose of this project is to re-implement vectorization in the GEM model, which has not been supported since the switch to scalar processors at the Meteorological Service. The vectorization and optimization tasks were undertaken using the NEC SX-6 ‘pharos2’ at Ouranos, running the SUPER-UX operating system. With the exception of the bug-fix outlined in the O7 optimization, the results from the vector modified code bit matches (using a binary diff) with the original GEM version 3.2.2 release. All tests were made using 6 steps (of 10 min each for a totally of 1 h of simulation time) on a 250x50 domain, taking advantage of the FFT package on a grid of similar extent to that used in the LACES project. The majority of the optimization steps are very simple, and the speed-up for each is shown using the -ftrace SX-6 compiler option. This is a standard compiler option, and does not incur any significant overhead. Since the -vopt optimizing option is standard for compiling, this option was also used during the vectorization and optimization processes.

Summarized performance statistics are provided for each step of the optimization, along with notes about the changes and recommendations about each optimizing step. None of these vectorizing or optimizing modifications should have any significant impact on scalar run-times; however a wall-clock comparison of the fully modified and clean version 3.2.2 codes should be undertaken on the IBM system. The maintenance requirements of the modifications should therefore be minimal, since there is only minimal splitting of the code between implementations for vector and scalar architectures. As noted in the detailed sections, the vast majority of the full optimization can be obtained without any additional maintenance.

### Basic Performance

The wallclock time and performance statistics for each of the GEM subprogram elements is derived from the traced output on the Ouranos NEC system. Only the main routines – from a wallclock consumption perspective – are included with this document, although full traces for all routines are available. This trace shows the out-of-the-box performance of the clean GEM model version 3.2.2.
### Bounds Check and Interpolation (O1)

The O1 optimizations are very basic in nature and are responsible for the majority of the enhancements made during the vectorization / optimization process. In the bounds checking routine for the LAM version of the model (`adw_ckbd_lam.ftn`), print statements in the loops prevented vectorization, so a set of flags were implemented instead. The functionality of the routine remains identical to that of the original version. The 3D interpolation routine (`adw_tricub_lag3d.ftn`) has a set of conditional statements that define vectors in only a single branch. This construct prevents vectorization and was modified to include null definitions in the opposite case. Although this appears to have no effect on the results since the (local) initialized values are never referenced in the routine, M. Valin expressed some concern from a memory perspective because of the branch guessing capability of the SX-6. The O1 optimizations are very high priority and are readily implemented.

### Horizontal Diffusion (O2)

The rationale for this optimization focuses on the small vector length noted in the `hzd_solmxma.ftn` routine. Reshaping the arrays in this subprogram should allow for an increase in the mean vector length; however, its wallclock timing did not change significantly. Since these modifications were did not enhance the vector length, op rate or wallclock timing, their inclusion is entirely optional.
Advection Interpolators (O3)

A set of routines is modified for this optimization (adw_interp2.ftn, adw_main_3_intlag.ftn and adw_comp.cdk). Additionally, a new routine is added (adw_tricub_lag3d_vec.ftn), the only vector-specific branch in the modified code. The main improvement in O3 was seen in the common namespace storage of array indexess, rather than a series of re-computations throughout the interpolator code. If this strategy is found to be effective on scalar machines, then the preprocessor branch isolating vector/scalar architectures in adw_interp2.ftn could be removed and adw_tricub_lag3d_vec.ftn used to replace the original adw_tricub_lag3d_vec.ftn. This would simplify maintenance and reduce code redundancy. The increased vector length may justify the inclusion of this set of modifications, however the overall flop rate seems to have dropped a bit.

Radiation Scheme (O4)

All indirect references in the radiation scheme (radir7.ftn) that occur more than once in a loop are replaced with scalar temporaries. This leads to the full optimization of the subprogram, but does not appear to improve either the wallclock or op-based performance indicators. As a result, the O4 optimization appears to be entirely optional, unless such a referencing scheme affects performance on scalar machines.
Convective Scheme (O5)

The Kain-Fritsch convective parameterization scheme (kfcp4.f.tn) required a number of modifications for vectorization and optimization. However, the main improvement was seen with the loop inversions that greatly enhanced the mean vector length and reduced the subprogram’s wallclock execution time. The changes made to this subprogram primarily mirror those implemented in version 3.7.2 of the physics package for the Kain-Fritsch scheme in use at that time. Additional modifications have been made based on compiler optimization and vectorization warnings, but do not appear to have a very significant effect on overall performance (although flop rate increases were noted). This is an important optimization that improves the overall vector length and wallclock times for the model.

Explicit Precipitation Scheme (O6)

The Sundqvist explicit scheme (consun1.f.tn) was modified using compiler directives – specifically in loops with scalar temporaries whose final values are unimportant – and subprogram inlining. The presence of a short module subprogram call in an inner loop prevented vectorization for a small segment of the code, apparently justifying the manual inlining of the calculation. These modifications enable the compiler to vectorize the code more completely and may therefore slightly improve the performance of the model.
Surface Module Summation (O7)

The implementation of the urban land use scheme appears to have introduced a memory bug for the SX-6 platform. This is one of the five surface modules called by the aggregating routine agrege1.ftn, and was not accessed during the vectorization and optimization procedures; however, the removal of branches referencing the urban scheme yields a different result from their evaluation as false conditionals. This is clearly incorrect, but the source of the problem is not obvious. Extensive notes outline the problem have been added to the agrege1.ftn routine, and an preprocessor directive results in a fatal warning if a NEC user tries to access the urban land use scheme. The functionality of this code has been verified as identical to the unmodified version on scalar platforms, and given that the magnitude of the errors on the vector machine are not small (up to 20% in the maximum vertical motion after 1 h of simulation time), at least some form of this fix/warning should be included to alert users to possible problems with the scheme.

FFT Array Referencing (O8)

The modifications to the FFT preparation routine (qcfft8.ftn) are extensive, and focus on the restructuring of references to both argument and internal arrays. In the original code, 2D arrays are referenced using 1D vector pointers in the subprogram, with complicated indirect referencing computed using a statement function. In the modified code, 2D arrays remain 2D arrays, and references are direct, although irregular strides will likely reduce the effectiveness of vectorization. Although the individual performance enhancement is relatively small for each time this subprogram is called, the frequency with which it is called makes these modifications relatively important. In addition, the 2D referencing dramatically improves code readability since the gridded location of each
element is easily determined.

<table>
<thead>
<tr>
<th>PROG.UNIT</th>
<th>FREQUENCY</th>
<th>EXCLUSIVE TIME[sec] (%)</th>
<th>AVER.TIME [msec]</th>
<th>MOPS</th>
<th>MFLOPS</th>
<th>V.OP</th>
<th>AVER. RATIO V.LEN</th>
<th>VECTOR TIM</th>
<th>I-CACHE MISS</th>
<th>O-CACHE MISS</th>
<th>BANK CONF</th>
</tr>
</thead>
<tbody>
<tr>
<td>adw_tricub_lag3d</td>
<td>72</td>
<td>6.037 (19.3)</td>
<td>83.850</td>
<td>5661.7</td>
<td>99.95</td>
<td>248.1</td>
<td>6.037</td>
<td>0.0002</td>
<td>0.003</td>
<td>0.1699</td>
<td></td>
</tr>
<tr>
<td>radir8</td>
<td>126</td>
<td>2.867 (9.2)</td>
<td>22.754</td>
<td>3582.1</td>
<td>99.35</td>
<td>246.1</td>
<td>2.823</td>
<td>0.0014</td>
<td>0.0005</td>
<td>0.0902</td>
<td></td>
</tr>
<tr>
<td>adw_trilin_turbo$1</td>
<td>114</td>
<td>2.360 (7.5)</td>
<td>20.702</td>
<td>4752.4</td>
<td>99.85</td>
<td>248.2</td>
<td>2.355</td>
<td>0.0003</td>
<td>0.0005</td>
<td>0.0467</td>
<td></td>
</tr>
<tr>
<td>qcfitt8</td>
<td>4560</td>
<td>2.255 (7.2)</td>
<td>0.494</td>
<td>1475.6</td>
<td>98.71</td>
<td>248.2</td>
<td>0.537</td>
<td>0.0052</td>
<td>0.0688</td>
<td>0.0194</td>
<td></td>
</tr>
<tr>
<td>adw_trajsp$1</td>
<td>38</td>
<td>1.492 (4.8)</td>
<td>39.272</td>
<td>7872.7</td>
<td>99.50</td>
<td>250.9</td>
<td>1.446</td>
<td>0.0005</td>
<td>0.0011</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>vco2inf2</td>
<td>126</td>
<td>1.109 (3.5)</td>
<td>8.802</td>
<td>6894.9</td>
<td>98.71</td>
<td>246.3</td>
<td>1.066</td>
<td>0.0007</td>
<td>0.0003</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>hzd_solfft_lam$1</td>
<td>24</td>
<td>0.851 (2.7)</td>
<td>35.439</td>
<td>3081.8</td>
<td>88.88</td>
<td>61.0</td>
<td>0.831</td>
<td>0.0018</td>
<td>0.0032</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>consun1</td>
<td>294</td>
<td>0.758 (2.4)</td>
<td>2.578</td>
<td>8713.3</td>
<td>99.38</td>
<td>246.1</td>
<td>0.730</td>
<td>0.0036</td>
<td>0.0045</td>
<td>0.0001</td>
<td></td>
</tr>
<tr>
<td>flxsurf3.fmi</td>
<td>217431</td>
<td>0.753 (2.4)</td>
<td>0.003</td>
<td>307.2</td>
<td>78.7</td>
<td>0.00</td>
<td>0.000</td>
<td>0.0016</td>
<td>0.1016</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>flxsurf3.fhi</td>
<td>217431</td>
<td>0.491 (1.6)</td>
<td>0.002</td>
<td>312.3</td>
<td>72.6</td>
<td>0.00</td>
<td>0.000</td>
<td>0.0007</td>
<td>0.0050</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>724168</td>
<td>31.304 (100.0)</td>
<td>0.043</td>
<td>4825.1</td>
<td>97.80</td>
<td>219.7</td>
<td>24.487</td>
<td>0.3455</td>
<td>1.1958</td>
<td>0.3776</td>
<td></td>
</tr>
</tbody>
</table>

PBL Stability Calculation (O9)

The stability of the boundary layer is used to compute surface fluxes in several subprograms, including diasurf2.ftn, flxsurf3.ftn, slfun_tq.ftn, and slfun_uv.ftn. Functions for calculating various stability-based parameters are found in stabfunc.cdk. These functions are called in loops from the flux subprograms and inhibit vectorization. The usual solution for this is to inline the (short) function using a compiler directive and the -pi compiler option. However, these functions access the namespace of the calling program directly, a bizarre behaviour that prevents compile-time inlining. The interfaces of the stabfunc.cdk functions have been modified to pass all array variables required in the functions themselves, and compiler directives have been added to ensure inlining. This leads to full vectorization of the loops that contain the function calls, and a significant overall speedup.

It seems to me that some more extensive recoding than simple vectorization and optimization is required to correct problems with these calculations. As far as I know, all code was supposed to have been removed from “comdec” files. This is clearly not the case for stabfunc.cdk, a comdec that contains three separate non-trivial functions. This comdec should be replace with a set of separate functions or a module. As currently written, the stabfunc.cdk code and all subprograms that call functions within it are essentially un-maintainable. Because the functions inherit the namespace of the parent program (both scalar and array variables from the calling program are referenced in the comdec’s functions without passing through the interface argument list), the naming scheme in the calling programs and the comdec must be identical. Clearly, this is all a complete violation of modular programming and virtually ensures that the collective code is essentially unmanageable. Furthermore, undeclared variables exist in the comdec subprograms that must be declared by the parent, even if they are not defined or referenced in the parent’s code (see the “X” variable in slfun_tq.ftn for example). Since there is no documentation in the comdec, it is virtually impossible for a developer to know what these variables should be, and what values they should contain (if any). And it gets worse; the functions actually make use of their access to the parent’s namespace to modify values within it.
For example, the “X” variable is declared in \texttt{flxsurf3.ftn}, defined internally in calls to the comdec functions (although it is not passed through the interface or returned), and then referenced later in \texttt{flxsurf3.ftn}. This is a horrendous violation of privacy and variable scope that appears to be completely unnecessary. During the vectorization and optimization process, I fixed only the aspect of this programming nightmare that were required to satisfy the optimizing compiler; however, this gross violation of basic programming principles should be addressed as soon as possible for the sake of modularity and maintainability.

\begin{verbatim}
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
TIME[sec](  % )    [msec]               RATIO V.LEN    TIME   MISS    MISS      CONF
adw_tricub_lag3d  72     6.074( 20.7)    84.355 5627.8 1300.8 99.95 248.1    6.073  0.0001  0.0003  0.1699
radir8           126     2.867(  9.8)    22.756 3581.7 1086.6 99.35 246.1    2.823  0.0015  0.0005  0.0904
adw_trilin_turbo$1  114     2.371(  8.1)    20.794 4731.3  838.7 99.85 248.2    2.366  0.0003  0.0004  0.0467
qcf8t8           4560     2.275(  7.8)    0.499 1462.8  514.6 81.37 129.8    0.544  0.0564  0.6780  0.0195
adw_trajsp$1    38     1.500(  5.1)    39.486 7830.1 3024.4 99.50 250.9    1.454  0.0003  0.0010  0.0000
vco2inf2         126     1.109(  3.8)     8.801 6896.3 2148.9 98.71 246.3    1.066  0.0007  0.0003  0.0002
hzd_solfft_lam$1  24     0.874(  3.0)    36.425 2998.4  729.3 88.88  61.0    0.855  0.0015  0.0034  0.0001
consun1          294     0.763(  2.6)     2.596 8652.6 3090.7 99.38 246.1    0.735  0.0039  0.0045  0.0001
wflux          17766     0.485(  1.7)     0.027 4967.6 1919.1 97.61 245.7    0.265  0.0272  0.0607  0.0001
kfcp4            294     0.475(  1.6)    1.617 6361.0 2009.1 98.60 229.6    0.411  0.0078  0.0073  0.0006
----------------------------------------------------------------------------------------------------------
total         130896    29.313(100.0)     0.224 5139.6 1572.2 98.26 219.5    24.696  0.3413  1.0536  0.3774
\end{verbatim}