Geosci. Model Dev., 4, 835–844, 2011
www.geosci-model-dev.net/4/835/2011/

Geoscientific
Model Development

FAMOUS, faster: using parallel computing techniques to accelerate the FAMOUS/HadCM3 climate model with a focus on the radiative transfer algorithm

P. Hanappe[1], **A. Beurivé**[1], **F. Laguzet**[1,*], **L. Steels**[1], **N. Bellouin**[2], **O. Boucher**[2,**], **Y. H. Yamazaki**[3,***], **T. Aina**[3], **and M. Allen**[3]

[1]Sony Computer Science Laboratory, Paris, France
[2]Met Office, Exeter, UK
[3]University of Oxford, Oxford, UK
[*]now at: Laboratoire de Recherche en Informatique, Orsay, France
[**]now at: Laboratoire de Météorologie Dynamiqe, IPSL, CNRS/UPMC, Paris, France
[***]now at: School of Geography, Politics and Sociology, Newcastle University, Newcastle, UK

Received: 10 May 2011 – Published in Geosci. Model Dev. Discuss.: 17 June 2011
Revised: 10 September 2011 – Accepted: 12 September 2011 – Published: 27 September 2011

Abstract. We have optimised the atmospheric radiation algorithm of the FAMOUS climate model on several hardware platforms. The optimisation involved translating the Fortran code to *C* and restructuring the algorithm around the computation of a single air column. Instead of the existing MPI-based domain decomposition, we used a task queue and a thread pool to schedule the computation of individual columns on the available processors. Finally, four air columns are packed together in a single data structure and computed simultaneously using Single Instruction Multiple Data operations.

The modified algorithm runs more than 50 times faster on the CELL's *Synergistic Processing Elements* than on its main PowerPC processing element. On Intel-compatible processors, the new radiation code runs 4 times faster. On the tested graphics processor, using OpenCL, we find a speed-up of more than 2.5 times as compared to the original code on the main CPU. Because the radiation code takes more than 60 % of the total CPU time, FAMOUS executes more than twice as fast. Our version of the algorithm returns bit-wise identical results, which demonstrates the robustness of our approach. We estimate that this project required around two and a half man-years of work.

Correspondence to: P. Hanappe
(hanappe@csl.sony.fr)

1 Introduction

Our work is motivated by the need for faster climate models in order to increase model resolution on current and future computing platforms and/or to increase the size of ensemble simulations. We believe that significant speed improvements cannot simply be obtained through the use of compiler flags or the use of pre-processor instructions that are inserted into the code. Instead, the developers must become familiar with the algorithms and adapt them to take full advantage of the modern CPU architectures.

In order to illustrate our point, we studied the code of the FAMOUS climate model (Jones et al., 2005; Smith et al., 2008), a low-resolution version of the better known HadCM3 model developed by the UK Met Office, and used by the University of Oxford in the ClimatePrediction.net Millennium experiment. A short overview of FAMOUS is given in Sect. 2.

We initially decided to use the CELL processor as the target platform for this study (Gschwind, 2007). Positioned somewhat between a generic multi-core chip and a graphics processor, the CELL offers a good compromise between various hardware evolutions. It has a hybrid multi-core design that groups a generic PowerPC processor and several accelerators, the so called *Synergistic Processing Elements*, on a single chip. We give further details of this processor in Sect. 3.

Section 4 describes the changes we have made to the radiation algorithm of FAMOUS to exploit parallel computing

Published by Copernicus Publications on behalf of the European Geosciences Union.

techniques. Our revised code yields very large performance improvements on the CELL processor. The modifications are beneficial for other computing platforms as well, including general purpose CPUs with vector instructions, multi-core platforms, and Graphics Processing Units (GPUs). Details of the performance we achieved are given in Sect. 5.

Our results and approach are in line with the work of Zhou et al. (2009) who also used the CELL processor to accelerate the computation of the radiation of the NASA GEOS-5 climate model.

2 About FAMOUS

FAMOUS (FAst Met Office/UK Universities Simulator) is a low-resolution version of the better known HadCM3, one of the coupled atmosphere-ocean general circulation models used to prepare the IPCC Third Assessment Report, and is a particular configuration of the U.K. Met Office's *Unified Model*, which is used for both weather prediction and climate simulation. FAMOUS is designed as a fast test bed for evaluating new hypotheses quickly or for running a large ensemble of long simulations. It has been calibrated to produce the same climate statistics as the higher resolution HadCM3.

FAMOUS uses a rectangular longitude/latitude grid. The resolution of the atmospheric component is 48×36 ($7.5°$ longitude $\times$ $5°$ latitude or roughly 830 km $\times$ 550 km at the equator) with 11 vertical levels. It has a 1-h time-step for the atmosphere dynamics and a 3-h time-step for the radiation. The resolution of the ocean component is 98×72 ($3.75°$ longitude $\times$ $2.5°$ latitude) with 20 vertical levels and a 12-h time-step.

FAMOUS contains legacy code that has been optimised for previous hardware platform and that has been adapted continuously. It consists of about 475 000 lines of Fortran 77 with some extensions of Fortran 90.

The computation of the radiative fluxes in the atmosphere uses the algorithm developed by Edwards and Slingo (1996).

3 About the CELL processor

The CELL processor was jointly developed by Sony, IBM, and Toshiba. It is used mainly in Sony's PlayStation 3 game console but also in supercomputers such as the Roadrunner at the Los Alamos National Laboratory[1].

The architecture of the chip is more generally known as the CELL Broadband Engine Architecture (CBEA). One of the design goals behind this architecture was to reduce the *von Neumann bottleneck*: the slowing down of the computation due to the latency of the data transfer to and from memory (Backus, 1978). Generic CPUs use a variety of techniques to reduce this latency, most notably the use of memory caches (Patterson and Hennesy, 1997). To reduce this bottleneck on the CELL, the choice was made to simplify the logic of the main CPU and use the freed-up space to incorporate additional small processors that have direct access to low-latency, on-die memory (Gschwind, 2007).

The resulting multi-core chip consists of two types of processors: the PowerPC Processing Element (PPE) and the so-called Synergistic Processing Element (SPE). The PPE is a general-purpose processor that is compliant with the PowerPC specifications. The SPE is a RISC processor that is optimised for vector operations. Each SPE has at its disposal a private Local Storage (LS) that is located on the chip. A schematic view of the CELL processor can be found in Fig. 1.

The CELL processor in the PlayStation 3 has one PPE and eight SPEs, of which six are available to programmers.

[1] At the time of writing, it is unclear whether the CELL product line will be further developed.

Table 1. The CPU time used by the main sub-components of FAMOUS. The first column shows the absolute time measured on the CELL's PPE for a one month simulation (720 atmosphere time-steps, 60 ocean time-steps). We used both the `gettimeofday` function and the PowerPC's hardware instruction counter to measure the intervals. The second column shows the relative CPU time.

Subroutine	Computation time (s)	Computation time (%)
Ocean sub-model	142.66	10.04
Atmosphere sub-model	1278.43	89.96
↪ Atmosphere physics	1120.47	78.85
↪ Radiation	950.34	66.87
↪ Long-wave radiation	572.84	40.31
↪ Short-wave radiation	314.76	22.15
↪ Convection	46.01	3.24
↪ Boundary layers	38.86	2.73
↪ Atmosphere dynamics	109.84	7.73
↪ Adjustment	49.10	3.46
↪ Advection	29.08	2.05
↪ Diffusion	10.52	0.74

4 Porting the FAMOUS radiation code

4.1 Profiling

The first step we took in this case study was to analyse which sub-components of FAMOUS consume most of the CPU time. Our analysis is summarised in Table 1. We obtained the run-time profile by inserting timers into the code.

The computation of the short-wave and long-wave radiation in the atmosphere stand out as the most interesting targets for parallelisation. Together they consume more than 60 % of the CPU time in spite of being called only every three hours of the atmosphere simulation.

The radiation code amounts to about 10 000 lines of Fortran code, forty times less than the total code size. The implementation, much of which is shared between the two types of

radiation, does not depend on other sub-components of FAMOUS, so it was a good candidate for a modular improvement.

4.2 Methodology

The restructuring of the radiation algorithm for the CELL processor proceeded in several steps that resulted in the following intermediate versions:

translated: We rewrote the original Fortran code in the *C* programming language because of technical constraints, as discussed in Sect. 4.3.

column: The radiation can be evaluated for each air column independently and we reorganised the code to make this data parallelism explicit. Further motivation for this reorganisation is given in Sect. 4.4.

simd: Two types of optimisations can be applied to the *column* version. The first is the use of "SIMD" vector instructions (Sect. 4.5).

multi-threaded (*mt*): The second optimisation that we applied to the *column* version is the use of multiple processors (Sect. 4.6).

spe: For the CELL processor, we produced the *spe* version in which the computation of the radiation is delegated to SPEs (Sect. 4.7).

opencl: For graphics processors, we translated the *column* version to the OpenCL language (Sect. 4.8).

We validated our changes in two ways. First, the binary output generated by our modified versions of FAMOUS is bit-wise identical with the binary output generated by the original version, when the code is generated without compiler optimisations. Bit reproducibility is a strong validation for code changes on the same computing platform (Easterbrook and Johns, 2009). This test is not feasible for the *spe* version because we cannot run the original code on the SPEs and because the SPEs have a different floating-point implementation than generic CPUs. We therefore introduced the second test. We ran a 120 yr simulation and compared the statistical properties of the results against a reference run (see discussion in Sect. 5.2).

4.3 Translating the code to *C*

The initial hardware platform that we targeted in this project was the commercial version of the PlayStation 3 game console. Because no Fortran compiler existed for the SPEs, we were compelled to translate the radiation code to *C*. An additional motivation for this translation was the good support that most *C* compilers provide for the vector data types and

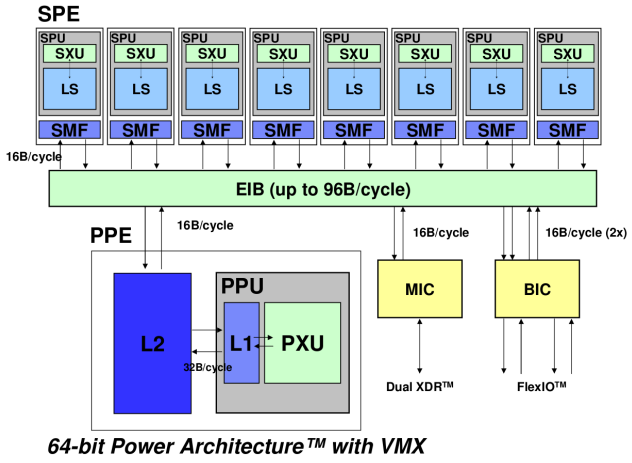

Fig. 1. A schematic overview of the CELL processor. The PowerPC Processing Element (PPE) and eight Synergistic Processing Elements (SPE) are connected on the Element Interconnection Bus (EIB). Beside the functional units (PXU), the PPU contains level 1 and 2 caches (L1 and L2). An SPE consists of two main components, the computational unit called Synergistic Processing Unit (SPU) and the Synergistic Memory Flow Controller, (SMF) which is in charge of the data transfers. In the SPU, one can further distinguish the functional units (SXU) and the Local Storage (LS). MIC stands for the memory controller. The BIC is an I/O controller. From Gschwind (2007, Fig. 1), with kind permission of Springer Science and Business Media.

SIMD instructions, as discussed in Sect. 4.5. Some support for SIMD instructions is provided by commercial Fortran compilers on other platforms but this adaptation would similarly require significant code changes.

We initially tested the *f2c* conversion program, which converts Fortran 77 code to *C*, to translate the two top-level entry functions of the radiation and their descendants. We modified *f2c* to recognise the Fortran 90 features but found that the produced *C* code was not satisfying. The array indexing of the *C* code reflected the Fortran indexing and did not help us for the subsequent restructuring of the algorithm. More importantly, because we did not obtain bit-wise identical results, we lost an indispensable code validation method. We therefore embarked upon a gradual manual translation process in combination with constant testing.

The main difficulty of this translation stems from the differences in the memory layout and in the indexing of the arrays between Fortran and *C*. To detect errors, we set up a testing environment that compared the subroutines' input and output arguments between the translated and the original version.

During this conversion process, we deleted unused code sections and a fair number of *if-then-else* statements in low-level computation routines that select which version of the algorithm is used. This results only in a minor loss

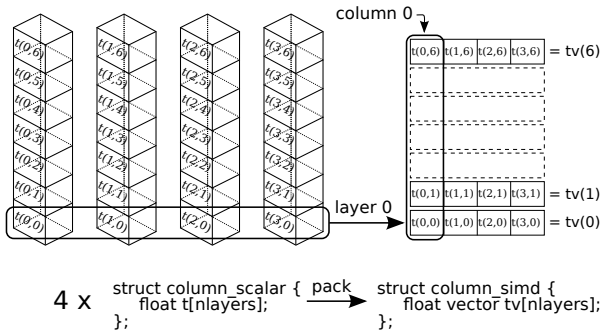

Fig. 2. Four instances of the array variable *t*, which extends over all layers of the column, are packed together into a single array of vectors.

in the flexibility of the radiation code because FAMOUS' configuration is not expected to be changed.

4.4 Computing the radiation per column

At low spatial resolutions, the net radiative fluxes across the boundaries of neighbouring columns are negligible compared to the fluxes across the layers of the atmosphere. Most climate models, including FAMOUS, therefore calculate the radiation in one air column independently from the other columns. We restructured the algorithm taking the column as the guiding principle because this organisation is more advantageous for the CELL processor, as discussed below. We call the resulting code the *column* version.

The original code does not explicitly use this data parallelism, although it does handle domain decomposition (the division of the global surface into several large sub-domains and simulating these sub-domains concurrently). The reason it does not compute one column at a time stems, in part, from its Cray heritage, the machine on which the code was developed. Vector machines can efficiently *chain* together subsequent operations on variable size vectors. The original code therefore stored a variable of the algorithm in a long array that spawns the 1728 horizontal grid cells of a layer. These long arrays are fed as often as possible to the vector processor. Most subroutines in the original code repeatedly execute the same two nested loops: the outer loop traverses the atmospheric layers and the inner loop iterates over the grid cells in the layer. Each subroutine thus touches upon a large memory area of about 74 KB per variable. The following code extract (simplified for clarity) gives an idea of the structure of the original algorithm.

```
function trans_src_coef(lambda,tau,gamma,trans)
  real xlamtau(n_cells, n_layers);
  loop i=1, n_layers:
    loop l=1,n_cells:
      xlamtau(l,i) = -lambda(l,i)*tau(l,i);
  loop i=1, n_layers:
    loop l=1, n_cells:
      xlamtau(l,i) = exp(xlamtau(l,i));
  loop i=1, n_layers:
    loop l=1, n_cells:
      trans(l,i)= xlamtau(l,i)*(1-gamma^2);
```

Modern CPUs have a considerable amount of logic to keep the functional units of the CPU busy, such as large memory caches, super-scalar execution, branch prediction, and deep instruction pipelines. The original algorithm is not well suited, however, for the CELL's SPUs, with their 256 KB of local storage. Processors with small memory caches or with simplified logic, such as embedded processors, are also likely to suffer.

The SPE and graphics processors are better adapted for *stream processing*: they can efficiently apply the same (small) algorithm to many (small) data structures. Our changes to the radiation code reflect that architecture. We store all the data for one column in a single data structure, group all column data structures into one big array, and then apply the modified radiation algorithm to all array elements. With these changes, each subroutine accesses a much smaller memory area of approximately 44 bytes per variable, compared to 74 KB in the original version.

In practice, the inputs and the outputs are stored in two separate data structures. We also introduced a third data structure that groups all the information about the spectral bands and the radiative properties of the trace gases. This spectral data is initialised once during the start-up of the model and subsequently reused.

When the top-level radiation subroutines are called, the input data is reorganised into the single-column data structures. This reorganisation costs some CPU time, as we will see later. After the reorganisation, a single top-level loop remains that iterates over all the columns and calls our modified radiation algorithm for each, as indicated in the following pseudo-code:

```
function sw_radiation(args)
  if (first_call)
    init_spectral_data(args, spectrum);
  loop l=1, n_columns:
    copy_arguments_to_input(args, in);
    swrad_one_column(spectrum, in, out);
    copy_results_to_output(out, args);
```

4.5 Using SIMD instructions

SIMD stands for Single Instruction Multiple Data and, in general, denotes a set of CPU instructions that apply the same operation to all elements of the vectors passed as arguments. Well-known examples include the PowerPC AltiVec instructions and the Streaming SIMD Extensions (SSE) found in Intel-compatible processors.

Vector machines, such as the Cray supercomputers, could efficiently apply several operations in series to variable

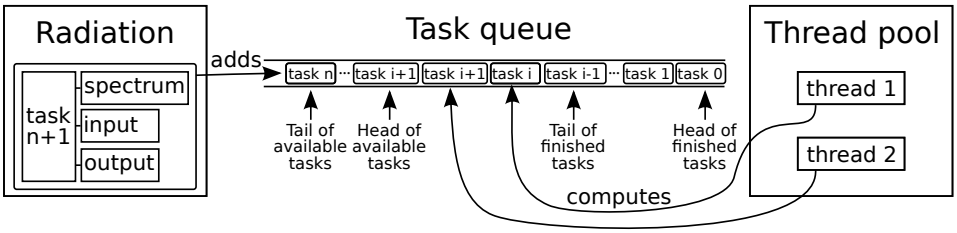

Fig. 3. The exchange of tasks between the top-level radiation function and the threads in the thread pool through the task queue.

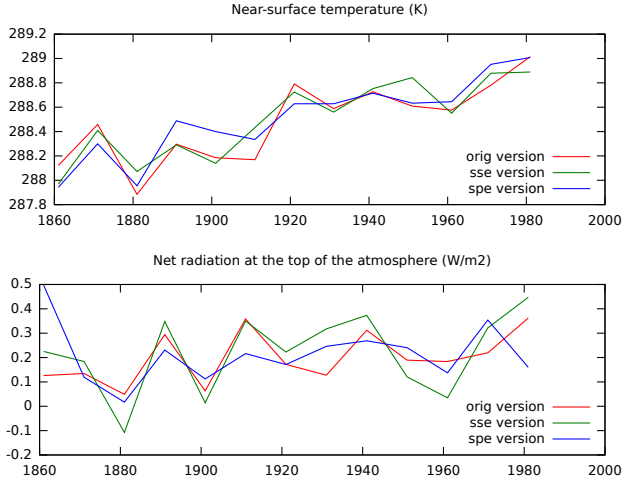

Fig. 4. Graphs showing the effects of rounding-errors on the decadal means of a 120 yr simulation using three different implementations of FAMOUS using: the Intel standard floating-point unit (original version), the Intel SSE extensions and libsimdmath (sse version), the CELL SPEs (spe version).

length vectors. The underlying implementation uses the notion of pipeline parallelism in which the mathematical operations on the vectors are chained together through an efficient pipeline. Most current commercial implementations of SIMD are based on the notion of data parallelism and use multiple arithmetical units to execute the operations. These implementations introduce fixed-size vector data types and incorporate new registers to operate on them. For scientific computing, vectors of four single precision floating-points numbers are mostly used although the SSE instructions on Intel can also operate on vectors of two double precision numbers.

To use the SIMD instructions for the radiation code, the simplest solution is to pack the values of the same layer, but of different columns, into the adjacent vector slots (see Fig. 2). This data layout is generally known as *structure of arrays*. This approach requires minimal changes to the code and computes four columns at once. The alternative organisation, called *arrays of structures*, would have been much more cumbersome because the radiation algorithm contains recursive loops that are hard to express using SIMD instructions.

The top-level functions must pack together the input data of four columns into a single data structure. The algorithm itself remains largely unchanged thanks to the compiler extensions for SIMD vectors. Most compilers recognise vector data types and translate the common mathematical operators to the appropriate SIMD instructions. Mathematical functions, such as the logarithmic or exponential functions, must be replaced with their vectorised versions, however, and we used the libsimdmath library (Dersch, 2008).

Conditional expressions require some special care. The *if-then-else* expression below will not return correct results when the variable x_1 is a SIMD vector:

```
if (x1 != 0)
    x3 = x2 / x1;
else x3 = abs(x2);
```

A correct approach is to replace the conditional with a *predication* through the use of a *select* instruction:

```
mask = compare_not_equal(x1, 0);
x3 = select(mask, x2 / x1, abs(x2));
```

A bit mask is first computed using a *vector compare* instruction. The mask is then used to select the requested values. Note that this technique computes both results first (x_2/x_1, and $abs(x_2)$) and then picks the correct value (Fisher and Dietz, 1998). For processors without branch prediction, such as the SPE, the select construct will offer better performance than a conditional expression when the code of the two branches is relatively small.

Code that uses look-up tables, such as interpolation tables, also requires special care because vector values cannot be used as an index into an array.

We applied the changes described above on our *column* version, leading to new code, called the *simd* version.

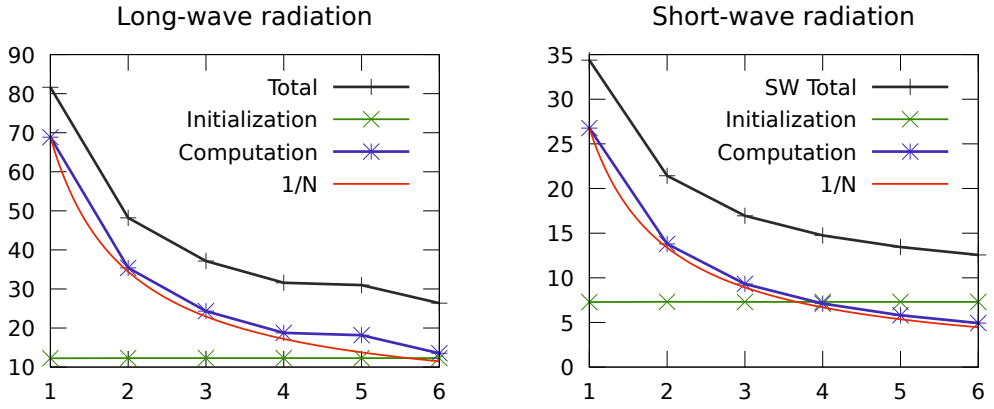

Fig. 5. The computation time of the long-wave and short-wave radiation as a function of the number of SPEs used.

4.6 Distributing the computation

We developed the *multi-threaded* version of the code in order to facilitate the distribution of the computation on the SPEs.

Ordinarily, FAMOUS uses MPI-based domain decomposition to distribute the computation of the radiation. In this approach, it is important to distinguish between the technique of domain decomposition, used to divide the data set by the number of available computing nodes, and the MPI technology, used to distribute the computation to the computing nodes.

Domain decomposition is not well suited for the CELL processor, because, with only six SPEs, the resulting data sets are too large to be stored in the SPE's Local Store. The MPI standard would be an appropriate choice to distribute the computation to the SPEs, but no freely available implementation of MPI for the SPEs was available (see also the discussion in the next section).

These two constraints lead us to use a *thread pool* and a *task queue* to distribute the computation over several processors. For each column, the top-level function prepares a task data structure. This task holds a reference to spectral data, the input, and the output data. The tasks are then inserted into the queue of available tasks. As soon as a task is available, one of the computing threads in the thread pool removes the task from the queue and computes the output. After the computation, the task is marked as finished and handed back to the main application (see Fig. 3).

We only implemented the parallel computation on shared-memory multi-processor systems, but we expect that a version for distributed memory systems, such as computing clusters, could easily be written with the help of the MPI standard.

4.7 Using the SPEs

We studied the existing software solutions available for the CELL to distribute the code to the SPEs. We found that most of the higher-level development libraries for the SPEs were either unstable, or more complex and slower than our solution (Laguzet, 2009). To distribute, the computation we decided to extend the multi-threaded version with the functionality provided in IBM's libspe2 library. The resulting binary code requires less than 2.3 KB of SPE storage.

For each SPE, one pool thread is created on the PPE. Whenever a pool thread obtains a task, it signals its associated SPE that new work is available using a *notification signal*. The SPE transfers the spectral data and the input to its local storage and runs the radiation algorithm. After the computation, the SPE copies the output to main memory and wakes up the PPE thread using the *interrupt mailbox*.

The memory space needed to execute the radiation algorithm, including the binary code, the data heap, and the execution stack, must fit within local storage of the SPE, which is limited to 256 KB. The binary code consumes around 60 KB. The data heap requires approximately $3.5+8.8\times B+2.5\times L$ kilobytes, where B is the number of wave-bands used to describe the spectral properties of the trace gases and L is the number of layers in the atmosphere. FAMOUS has eleven layers and uses up to eight spectral bands, resulting in roughly 100 KB of data storage. We measured that the maximum depth of the execution stack is close to 11 KB. The limited size of the local storage has thus not been an issue for FAMOUS, but could become a concern for models that use many spectral wave-bands or that have a high vertical resolution.

4.8 The OpenCL version for graphics processors

To show that the re-structured code is well adapted to the architecture of graphics processors, we decided to translate

Table 2. The computation times (in seconds) and speed-up factors (columns marked with '×') of FAMOUS, the short-wave radiation (SW), and the long-wave radiation (LW) for the different code versions. The time indicated for FAMOUS excludes the work done during the start-up of the program. The model was run for 720 atmosphere time-steps. For the spe version, the number between parentheses indicate how many columns were computed simultaneously using SIMD, and how many SPEs were used concurrently.

Code version	FAMOUS		LW		SW	
	s	×	s	×	s	×
original	1446.5	1	630.8	1	321.8	1
column	1394.1	1.04	657.3	0.96	256.3	1.26
mt (2 thr.)	1167.8	1.24	495.9	1.27	193.4	1.66
spe (1/1)	844.9	1.71	274.7	2.30	107.9	2.98
spe (1/6)	531.3	2.72	50.2	12.56	20.4	15.78
spe (4/6)	490.7	**2.95**	22.3	**28.24**	10.8	**29.69**

the algorithm to the Open Computing Language (OpenCL, version 1.0), which is a C-like language to program graphics processors.

We worked with the scalar column version of the algorithm, not the SIMD version. The data structure for all the columns are stored sequentially in a single, large, array that is transfered to the graphics card. The GPU applies the radiation algorithm to every column and the results are copied back to main memory.

5 Stability test and performance benchmarks

Before we take a look at the results of our benchmarks, we will evaluate the impact of the differences in the floating-point computation on the stability of the climate simulations.

5.1 Testing platform

All tests were performed on PlayStation 3 hardware running GNU/Linux, Fedora release 8. The code was compiled using the GNU compiler suite, gcc and gfortran version 4.1.2, for the 32-bit PowerPC architecture. When compiler optimisations were enabled, we used the -O3 flag. The radiation was computed using single-precision floating points[2].

The tests for the Intel compatible platforms were performed on a Sony VAIO VPC-F11S1E equipped with an Intel Core i7 Q720 at 1.6 GHz. All code was compiled using gcc and gfortran version 4.4.3. We used the same laptop for the benchmarks on graphics processors, a Nvidia GeForce GT 330M with 48 CUDA cores running at 1265 MHz. All OpenCL code was developed using nVidia's development kit.

[2]ClimatePrediction.net also uses a single-precision version of FAMOUS.

Table 3. Comparison of the computation time versus the time to initialise the column data structure and copy the results back (finalisation). The third column shows the computation's speed improvement.

Code version		Initialisation & finalisation (s)	Computation (s)	×
SW	original	4.63	317.08	1
	column	5.23	251.08	1.26
	spe (6)	10.39	4.17	**76.0**
LW	original	7.78	622.65	1
	column	8.65	648.64	0.96
	spe (6)	16.30	10.97	**56.8**

5.2 The effects of rounding errors on the SPEs

The single-precision floating point calculations on the SPEs are not fully compliant with the IEEE 754 standard. In particular, the rounding mode of floating-point operations is always truncation, while CPUs typically round the intermediate results to the nearest value. To evaluate the effects of the truncation on the stability of the climate model, a 120 yr simulation was performed and the result compared to a reference run. This simulation was forced by historical changes in greenhouse gas concentrations, solar forcing, volcanic aerosols, and a time-varying climatology of sulphate aerosols.

As can be seen in Fig. 4, the decadal mean of the global average surface temperature computed by the *spe* version (blue line) evolves differently than the output of the reference simulation (red line). However, the results did not show any instability or bias and the statistical differences between the versions are comparable to running the unmodified model on different platforms or with different compiler configurations (see also Knight et al. (2007) for a discussion on how the hardware variation effects the model behavior). The green line in the figure shows the results obtained with the *simd* version using Intel's SSE.

5.3 The benchmark tests on the CELL processor

The performance numbers discussed in this section were obtained by running FAMOUS on the CELL processor for one simulated month, or 720 atmospheric time-steps. The computation times on the PPE and the SPE were determined using the CPU clock tick counters (using the *mftb* instruction on the PPE and the hardware *decrementer* on the SPE). The standard *gettimeofday* function was used for verification.

In Table 2, we see that the *column* version offers little performance improvements over the original code version. The multi-threaded (mt) version, however, yields a speed improvement of 1.24 because the PPE has hardware support to execute two threads simultaneously. The real improvements

Table 4. Comparison of the computation time versus the time needed to transfer the data to/from main memory and the SPE's local storage.

	LW (s)	SW (s)
Copy input	0.0147	0.0115
Computation	10.97	4.17
Copy output	0.0021	0.0020

Table 5. The computation time (in seconds) and the speed improvement ('×' column) for FAMOUS, the long-wave (LW), and short-wave (SW) radiation for the different versions of the radiation algorithm on the Intel test platform.

Code version	FAMOUS		LW		SW	
	s	×	s	×	s	×
original	275.80	1	125.26	1	60.73	1
column	338.14	0.82	169.35	0.74	82.92	0.73
simd	132.09	**2.09**	31.00	**4.04**	14.88	**4.08**

come when the computation is executed using an SPE. When we apply 6 SPEs to the task, the computation time for the radiation is further reduced more than 5-fold. Finally, when four columns are packed together and computed simultaneously (as described in Sect. 4.5) we observe the fastest computation time.

If we do not take into account the time to reorganise the data and we consider only the time to compute the radiation, we see a speed improvement of 76 and 56 for the SW and LW radiation, respectively. This reveals the full potential of our method because the initialisation overhead could be avoided if we had fully adapted the FAMOUS code and the proposed column data structures were used throughout.

In Table 4, we see that the time needed to transfer the data back and forth between the main memory and the SPE's local stores is more than two orders smaller than the computation time and there is little risk of saturating the communication bus. This positive result is likely to be reproducible for other column-based radiation algorithms. We did not implement a double-buffering scheme to overlap the data transfers with the computation because the small gain in performance does not justify the additional code complexity.

The algorithm scales very well with the number of SPEs, even though the test was limited to six processors. In Fig. 5, the initialisation time appears as a constant. The computation time, including the data transfers, is almost inversely proportional to the number of deployed SPEs. This was to be expected because the data transfers have a small cost.

The time that is needed to convert the original Fortran arrays to the new column data structures is relatively costly. For the SW radiation, this reorganisation requires more time than the computation. Because of the reorganisation overhead, the speed improvements of the drop-in replacements for the SW and LW subroutines is less, but still approximately 30 times faster than the original subroutines.

Table 6. The computation time (in seconds) and the speed improvement ('×' column) for FAMOUS, the long-wave (LW), and short-wave (SW) radiation for the original version of the radiation algorithm on the Intel Core i7 Q720 and the OpenCL version running on the Nvidia GeForce GT 330M.

Code version	FAMOUS		LW		SW	
	s	×	s	×	s	×
original	275.80	1	125.26	1	60.73	1
column	168.48	**1.64**	45.38	**2.76**	21.50	**2.82**

The overall speed of FAMOUS increases by a factor of three. The *spe* version simulates 15 yr per wall-clock day, up from 5 yr day^{-1} for the original version.

We could have applied additional optimisation techniques in addition to using SIMD operations and replacing conditionals with predications. Other techniques to consider include the use of software pipelines, faster synchronisation techniques between the PPE and the SPE, and double-buffering the data transfers (Eichenberger et al., 2005). However, these improvements would have a marginal impact on the overall computation time of FAMOUS: the new performance is now determined by the serial code running on the PPE (e.g. the dynamics of the atmosphere and oceans).

5.4 Intel-compatible processors

The results in Table 5 show that the SIMD version of FAMOUS on Intel-compatible CPUs runs more than two times faster than the original version. The radiation algorithm is executed more than four times faster. The *simd* version simulates roughly 54 yr in one wall-clock day, compared to 26 yr day^{-1} for the original version.

5.5 Graphics processors

The benchmark results on the graphics processor are displayed in Table 6. We see a 2.5 times reduction in the computing time. It is likely that further speed improvements can be obtained when SIMD vector instructions are used on the GPU. The benchmark data should not be considered representative of the performance of GPUs versus CPUs in general. The work spent on porting the code to OpenCL, approximately one week, and the performance results above are a strong indication, however, that the organisation of the radiation code in columns is also appropriate for graphics processors.

As shown in Table 7, the time for the computation largely outweighs the time for the data transfer from/to the GPU's memory, similar to the results obtained on the CELL processor.

Table 7. Comparison of the computation time on the GPU versus the time needed to transfer the data to/from main memory and the GPU's local memory.

	LW (s)	SW (s)
Copy input	0.70	0.54
Computation	41.40	19.54
Copy output	0.14	0.12

6 Conclusions

It is now generally understood that future performance improvements of computing hardware will come mainly from increased use of parallel computing (Asanovic et al., 2009). The rise of graphics processing units (GPU) for general-purpose computation (Owens et al., 2007) has also led to the use of hybrid CPU-GPU platforms for scientific computing. In addition, the high power consumption and the stagnating performance of generic CPUs has made the use of cheaper embedded processors more attractive for scientific computing (Wehner et al., 2008).

In this paper, we believe we have made a clear case that a re-evaluation of the structure of existing algorithms for these novel computing platforms can yield a high return on investment. Our reorganisation of the FAMOUS radiative transfer algorithm resulted in a 30-fold speed improvement on the CELL processor, a 4-fold speed improvement on the Intel processor, and a 2.5 speed improvement on GPUs.

The effort for recoding may be significant, though. The programming of the CELL and graphics processors remains, in general, a difficult technical task. We were fortunate, however, that the radiation algorithm exhibits a form of data parallelism that is suitable for SIMD instructions.

The achieved reduction in computing time not only means faster results. It may now be reasonable to call the fast radiation code at every atmosphere time-step, instead of every three time-steps, to improve the sampling of interaction of the clouds with the radiation.

After our code changes, the performance of FAMOUS is determined mainly by the other components, in particular, by the dynamics of the atmosphere and oceans (see Table 1). This reflects Amdahl's law, which states that the performance of parallel applications is largely determined by the performance of the serial code segments. It is unlikely that the same speed-up factors can be obtained for all components of FAMOUS. Continuing the optimisation of FAMOUS beyond the radiation code would require a continued effort with decreasing gains.

It is worth raising the question as to whether the optimisations that we have applied could be done automatically. In essence, we converted code that is well-adapted for the Cray vector machines into code that is well-adapted for the CELL processors. This work was done manually and required about two and a half man-years of work. Research projects exist that try to tackle the re-organisation of code using automatic optimisation tools and that target whole programs instead of well contained algorithms (Liao et al., 2009). It would be an interesting exercise to analyse the results of these automatic conversion tools. Our manually optimised code may be a useful reference in such a study.

Another interesting long-term approach would involve abstracting away the hardware platform on which the climate model will ultimately be executed. Climate models like FAMOUS will be run on increasingly heterogeneous hardware platforms that evolve incessantly. Targeting a specific architecture for model development may not be the best choice. It is worth investigating how this hardware independence could be achieved. Recent work on language virtualisation may be a valuable starting point for this inquiry (Chafi et al., 2010).

Acknowledgements. Sony CSL would like to thank the UK Met Office for providing us with a Vendor Benchmarking License, and the Sony Computer Entertainment R&D teams for their support. This work was also supported by EU FP6 project European Climate of the Last Millennium (MILLENNIUM 017008). O. B. and N. B. acknowledge support from the DECC/Defra Hadley Centre Climate Programme (GA01101). We thank Simon Colton and the reviewers for their constructive comments.

Edited by: D. Ham

References

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., and Yelick, K.: A view of the parallel computing landscape, Communications of the ACM, 52, 56–67, doi:10.1145/1562764.1562783, 2009.

Backus, J.: Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs, Communications of the ACM, 21, 613–641, doi:10.1145/1283920.1283933, 1978.

Chafi, H., DeVito, Z., Moors, A., Rompf, T., Sujeeth, A. K., Hanrahan, P., Odersky, M., and Olukotun, K.: Language virtualization for heterogeneous parallel computing, in: OOPSLA '10 Proceedings of the ACM international conference on Object oriented programming systems languages and applications, pp. 835–847, ACM, New York, NY, USA, doi:10.1145/1869459.1869527, 2010.

Dersch, H.: Universal SIMD-Mathlibrary, Tech. rep., Furtwangen University of Applied Sciences, http://webuser.fh-furtwangen.de/~dersch/libsimdmath.pdf, 2008.

Easterbrook, S. M. and Johns, T.: Engineering the Software for Understanding Climate Change, IEEE Comput. Sci. Eng, 11, 65–74, doi:10.1109/MCSE.2009.193, 2009.

Edwards, J. and Slingo, A.: Studies with a flexible new radiation code. 1: Choosing a configuration for a large-scale model, Quart. J. Roy. Meteor. Soc., 122, 689–719, doi:10.1002/qj.49712253107, 1996.

Eichenberger, A. E., O'Brien, K., O'Brien, K., Wu, P., Chen, T., Oden, P. H., Prener, D. A., Shepherd, J. C., So, B., Sura, Z.,

Wang, A., Zhang, T., Zhao, P., and Gschwind, M.: Optimizing Compiler for the CELL Processor, in: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques, 161–172, doi:10.1109/PACT.2005.33, 2005.

Fisher, R. J. and Dietz, H. G.: Compiling For SIMD Within A Register, in: 11th Annual Workshop on Languages and Compilers for Parallel Computing, 290–304, Springer Verlag, Chapel Hill, 1998.

Gschwind, M.: The Cell Broadband Engine: Exploiting multiple levels of parallelism in a chip multiprocessor, Int. J. Parallel Prog., 35, 233–262, doi:10.1007/s10766-007-0035-4, 2007.

Jones, C., Gregory, J., Thorpe, R., Cox, P., Murphy, J., Sexton, D., and Valdes, P.: Systematic optimisation and climate simulation of FAMOUS, a fast version of HadCM3, Clim. Dynam., 25, 189–204, doi:10.1007/s00382-005-0027-2, 2005.

Knight, C. G., Knight, S. H. E., Massey, N., Aina, T., Christensen, C., Frame, D. J., Kettleborough, J. A., Martin, A., Pascoe, S., Sanderson, B., Stainforth, D. A., and Allen, M. R.: Association of parameter, software and hardware variation with large scale behavior across 57,000 climate models, Proceedings of the National Academy of Sciences, 104, 12259–12264, doi:10.1073/pnas.0608144104, 2007.

Laguzet, F.: Analyse des performances du processeur CELL, Master's thesis, Institut d'Electronique Fondamentale, 2009.

Liao, C., Quinlan, D. J., Vuduc, R., and Panas, T.: Effective Source-to-Source Outlining to Support Whole Program Empirical Optimization, The 22nd International Workshop on Languages and Compilers for Parallel Computing, Newark, Delaware, USA, http://www.springerlink.com/content/85w42v4122x1164p/, 2009.

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krger, J., Lefohn, A., and Purcell, T. J.: A Survey of General-Purpose Computation on Graphics Hardware, Comput. Graph. Forum, 26, 80–113, doi:10.1111/j.1467-8659.2007.01012.x, 2007.

Patterson, D. A. and Hennesy, J. L.: Computer Organisation & Design: The Hardware/Software Interface, Morgan Kaufmann Publishers, second edn., 1997.

Smith, R. S., Gregory, J. M., and Osprey, A.: A description of the FAMOUS (version XDBUA) climate model and control run, Geosci. Model Dev., 1, 53–68, doi:10.5194/gmd-1-53-2008, 2008.

Wehner, M., Oliker, L., and Shalf, J.: Towards Ultra-High Resolution Models of Climate and Weather, Int. J. High Perform C., 22, 149–165, doi:10.1177/1094342007085023, 2008.

Zhou, S., Duffy, D., Clune, T., Suarez, M., Williams, S., and Halem, M.: The impact of IBM Cell technology on the programming paradigm in the context of computer systems for climate and weather models, Concurr. Comp.-Pract. E., 21, 2176–2186, doi:10.1002/cpe.1482, 2009.