Analysis of parallelisation of 3D-CEMBS model using technologies like OpenACC and OpenMP

Analiza możliwości zrównoleglenia modelu 3D-CEMBS z wykorzystaniem technologii typu OpenACC i OpenMP

Piotr Piotrowski

Maritime Institute in Gdańsk, Poland

ABSTRACT: Oceanographic models utilise parallel computing techniques to increase their performance. Computer hardware constantly evolves and software should follow to better utilise modern hardware potential. The number of CPU cores with access to shared memory increases with hardware evolution. To fully utilise the possibilities new hardware presents, parallelisation techniques employed in oceanographic models, which were designed with distributed memory systems in mind, have to be revised. This research focuses on analysing the 3D-CEMBS model to assess the feasibility of using OpenMP and OpenACC technologies to increase performance. This was done through static code analysis and profiling. The findings show that the main performance problems are attributed to task decomposition that was designed with distributed memory systems in mind. To fully utilise modern shared memory systems, other task decomposition strategies need to be employed.
The presented 3D-CEMBS model analysis is a first stage in wider research of oceanographic models as a specific class of parallel applications. In the long term the research will result in proposing design patterns tailored for oceanographic models that would exploit their characteristics to achieve better hardware utilisation on evolving hardware architectures.

KEYWORDS: 3D-CEMBS • parallel computing • MPI • OpenMP • OpenACC • distributed memory • shared memory

STRESZCZENIE: Modele oceanograficzne wykorzystują przetwarzanie równoległe dla zwiększenia wydajności. Sprzęt komputerowy ciągle ewoluuje, więc oprogramowanie powinno zmieniać się razem z nim, aby w pełni wykorzystać potencjał współczesnego sprzętu. Wraz z rozwojem sprzętu komputerowego zwiększa się liczba rdzeni procesorów, które mają dostęp do pamięci współdzielonej. Aby w pełni wykorzystać możliwości nowego sprzętu, techniki zrównoleglania wykorzystywane w modelach oceanograficznych muszą zostać zrewidowane. Modele oceanograficzne były często projektowane z myślą o systemach z pamięcią rozproszoną. Niniejsze badania skupiają się na analizie modelu 3D-CEMBS pod kątem możliwości wykorzystania technologii OpenMP i OpenACC w celu podniesienia wydajności modelu. W tym celu została przeprowadzona statyczna analiza kodu modelu oraz profilowanie. Wyniki badań pokazują, że główny problem wydajnościowy modelu jest wynikiem zastosowania dekompozycji zadań przewidzianej dla systemów z pamięcią rozproszoną. Aby w pełni wykorzystać współczesne komputery z pamięcią współdzieloną należy wprowadzić inne strategie dekompozycji zadań.

SŁOWA KLUCZOWE: 3D-CEMBS • przetwarzanie równoległe • MPI • OpenMP • OpenACC • pamięć rozproszona • pamięć dzielona

INTRODUCTION

Oceanographic models play an important role in operational oceanography. Apart from the accuracy and the like attributes of the model forecasts, model performance is also crucial. Model performance is important for several reasons. First of all, if a forecast is to be used for operational purposes it has to be computed within a prescribed amount of time. This time limit can vary depending on the particular purpose or the forecast length, however 2 hours is a common time limit to finish a forecast. Moreover, the faster the model is, the longer forecast can be computed within the same time limit. This is important if the forecast is used for planning purposes like flood prevention, etc. Model performance also has an economic aspect – the more efficient the model is, the less power it consumes. Also if the model is more efficient, cheaper hardware can be used to perform the computations.

This research focuses on model parallelisation as a performance improvement method. Nowadays, Message Passing Interface (MPI) [9] is commonly used for model parallelisation. This research analysis OpenMP [12] and OpenACC [11] as technologies that could potentially improve model performance.

PARALLELISATION TECHNOLOGIES

There are many technologies that can be used to facilitate parallel programming paradigm. This research focuses on the usage of three of them: MPI, OpenMP and OpenACC. These three technologies represent three different parallelisation models and the conclusions can be easily generalised to other technologies corresponding to those models.

Message Passing Interface

Message Passing Interface (MPI) [9] is a parallelisation technology commonly used in cluster environments. It is a de facto standard for oceanographic models. Its design comes from the early nineties and was tailored for computer systems with distributed memory. MPI applications were originally run on many connected nodes, each of which had only few CPU cores and little RAM memory. Even if a node has several CPU cores, MPI application ignores this fact. MPI application consists of a set of processes. Each process communicates with others as if they were on separate nodes. This is convenient as processes do not have to know if their siblings are on the same node or not. Thanks to this design all communication from the process point of view is done in a uniform manner and it is the role of the MPI implementation to decide what medium to use for communication: TCP/IP network, other network protocols if appropriate hardware is available or shared memory if processes are on the same node.

However, since the nineties computer hardware changed. Nowadays computer nodes in clusters have several dozens of CPU cores ranging from 24 to even 64 cores. A single node can feature even 1 TB of shared RAM, with 128 or 256 GB of RAM being quite common. These changes in hardware availability should be taken into account when writing parallel applications like oceanographic models. Especially important is the availability of large memory shared among many CPU cores.

MPI was designed for systems with distributed memory and is still important if models require multiple nodes to be computed within a satisfactory time frame. However, there are more efficient ways to parallelise software if it fits a single shared memory system.

OpenMP

OpenMP [12] is a thread based parallelisation technology. This makes it complementary to MPI. MPI was designed to facilitate communication between nodes, while OpenMP is designed to parallelise work within a single node using shared memory for communication. Even though MPI implementation can also use shared memory as a communication medium, memory copying is performed between private memory spaces of two or more processes. In OpenMP communication is a zero-copy communication: that is a thread can have direct access to other thread's memory without the need for copying.

There is a number of thread based technologies. Practically every modern programming language has its own thread based API, while native threading is usually done using Pthreads in POSIX compliant environments. Fortran lags behind in this respect with some additions concerning parallel computing in Fortran 2008 that resemble MPI computation model.

OpenMP was chosen in this research for its ease of use achieved thanks to declarative programming paradigm. Appropriately annotating source code indicating what parts can be computed in parallel is sufficient to achieve a significant parallelisation level in many typical cases. Moreover, OpenMP gained wide adoption. It is supported by commercial compilers like the PGI compiler as well as open source compilers like GCC.

OpenACC

The third technology considered in this research is OpenACC [11]. This technology allows the usage of GPUs for numerical computations – general-purpose computing on graphics processing units (GPGPU). Again there are other GPGPU technologies like OpenCL or CUDA, but OpenACC has been chosen due to its simplicity and therefore easier assessment of feasibility of using GPGPU for oceanographic models. Its adoption by compilers also increases and it is implemented by commercial compilers like the PGI compiler, but also by open source compilers, for example GCC starting from version 5.

The design of OpenACC is similar to that of OpenMP – it also conforms to declarative programming paradigm. Similarly to OpenMP, with OpenACC one annotates source code fragments to indicate that it can be computed in parallel on a GPU. There are also additional annotations that can help with memory management, since CPU and GPU usually have separate memory

```
!$OMP PARALLEL DO PRIVATE(iblock)
do iblock = 1,nblocks_clinic
   UBTROP(:,:,oldtime,iblock) = UBTROP(:,:,curtime,iblock)
   VBTROP(:,:,oldtime,iblock) = VBTROP(:,:,curtime,iblock)
   UVEL(:,:,:,oldtime,iblock) = UVEL(:,:,:,curtime,iblock)
   VVEL(:,:,:,oldtime,iblock) = VVEL(:,:,:,curtime,iblock)
   RHO (:,:,:,oldtime,iblock) = RHO (:,:,:,curtime,iblock)
   TRACER(:,:,:,:,oldtime,iblock) = &
        TRACER(:,:,:,:,curtime,iblock)
end do
!$OMP END PARALLEL DO
```

Figure 1. Memory copying using OpenMP

pools. This can change however with the adoption of heterogeneous system architecture (HSA) [6] that facilitates unified memory model. HSA can potentially improve performance of OpenACC annotated code, but HSA supporting hardware is still not common and HSA as a technology is still in its development state.

MODEL ANALYSIS

To analyse parallelisation of the 3D-CEMBS model, [2, 3, 4, 10] static code analysis and model profiling were performed. Code analysis was conducted to see what technologies and parallelisation methods are already used in the model. Moreover, static code analysis reveals fragments of code that adhere to code patterns that are inherently prone to parallelisation. Profiling reveals which fragments of code are the bottlenecks and allows an estimation of potential benefits of improvements of particular code fragments.

Code analysis

3D-CEMBS utilises MPI as the main parallelisation technology, but also uses OpenMP. OpenMP is used in a way similar to MPI, that is the task decomposition is block based [7, 8]. This way OpenMP does not utilise the full potential of the shared memory architecture, since such task decomposition was initially designed for distributed memory systems. This requires synchronisation between each block and the more blocks there are the more effort is put into data synchronisation. Moreover, the blocks representing individual tasks are intentionally small to improve load balancing and to remove more land representing grid cells from computations. However, this is done at a price – increased communication time.

Static code analysis also revealed some code that could be refactored to improve sequential performance. The code shown in Figure 1 copies large amounts of memory. This code is parallelised using OpenMP, however such code in general should not be necessary at all. It does not perform any computations, but moves memory from one place to another. Such code should be substituted by reference swapping or a similar technique – it does not matter where some value resides in memory as long as we have a reference or a pointer to the correct place. Memory in computer systems is much slower than the CPU, therefore memory copying is relatively a very slow task. In some cases time spent on memory access can dominate over time spent on floating point computations.

Code similar to that in Figure 1 can also be found in other oceanographic models, for example in WAM wave model [14]. This coding pattern might have come from the lack of reference or pointer like features in early Fortran language versions. This shows that the choice of programming language might influence coding techniques that can influence performance. Unfortunately once implemented, such patterns are hard to remove from code. This is partly a consequence of using global variables. Direct usage of global variables makes it harder to just swap references to memory. The use of global variables also makes the code harder to refactor in order to remove such coding patterns, because a change of a global variable would require changing all its usages, also indirect usages, throughout the code. Although dominant in oceanographic models, Fortran might not be the best choice of language to write high quality code with, since it encourages obsolete programming techniques [1].

Profiling

Profiling of the model is necessary to conduct performance analysis, because it shows where the real bottlenecks are. Profiling of the model was performed on a setup running on 16 CPU cores communicating using MPI. 16 CPU cores is not a large number, but was enough to reveal poor scalability being the result of communication overhead.

Statistically, computer applications spend most of their time in a relatively small amount of code. Time spent executing the

Table I CEMBS model profiling results

Procedure	Time %
pop_haloupdate3dr8	8%
lw_lim	6%
step	5%
pop_haloupdate2dr8	4%
ice_haloupdate3dr8	4%
...	...

prepared model setup was distributed among many procedures, without an apparent bottleneck (see Table I). In such a situation it is hard to make performance improvements, because any single change has minimal impact on the overall execution time. However, more detailed analysis of the top five time consuming procedures revealed that in fact three of them are procedures handling synchronisation between individual blocks. Another procedure is the one from which the code in Figure 1 was taken. Only one of the top five procedures performs actual model computations.

Optimisation of any of the procedures would have negligible impact on the overall performance. However, the three procedures responsible for synchronisation between individual blocks sum up to 16% of the overall execution time. 16% is a significant amount of time. This time can be reduced or even eliminated by changing task decomposition from the block based.

The procedure that performs extensive memory copying uses 5% of the overall execution time. This adds up to 21% of the overall execution time that can be potentially optimised, by changing task decomposition and refactoring the code not to use global variables and then removing unnecessary memory copying. Often the main optimisation efforts are focused on floating point computation performance. Profiling of this model shows that task decomposition and memory management issues can become dominant problems over the raw computational performance. This has yet another consequence: OpenACC, which focuses on floating point computation performance, cannot improve performance of the model without changing the model architecture and task decomposition first.

RECOMMENDATIONS

To improve the 3D-CEMBS model performance a major change in task decomposition has to be made. Moreover, to fully utilise the full potential of systems with shared memory, OpenMP cannot be used as a substitute to MPI following the same patterns. Different design assumptions of OpenMP require a different approach than that from MPI. Otherwise the capabilities of OpenMP are not fully exploited. These are, however, major architectural changes and require significant effort to implement. The required work is programming, but also research on possible task decompositions.

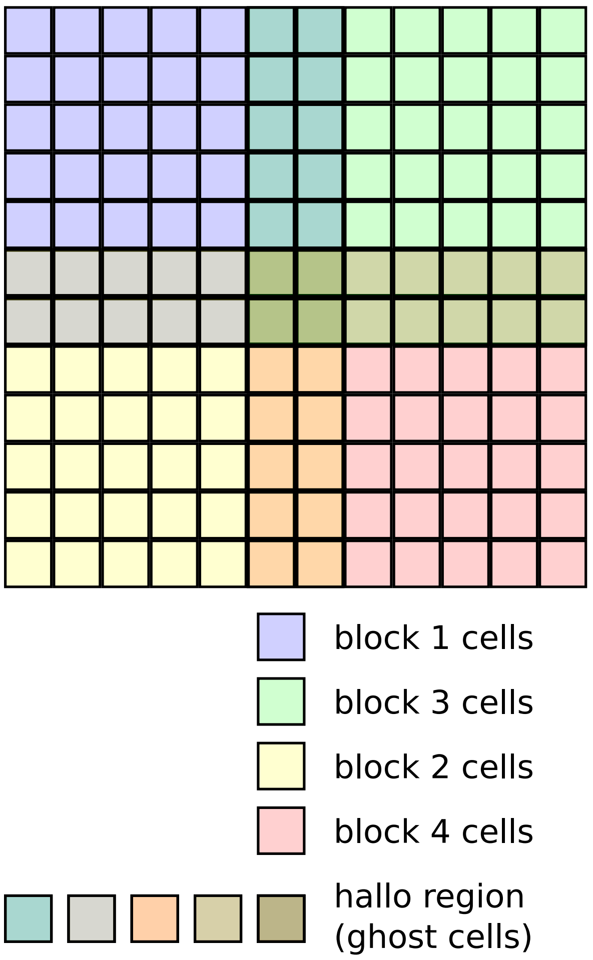

Figure 2. Block decomposition with 1 cell wide halo region

Task decomposition

The block based task decomposition used in the 3D-CEMBS model [7, 8] was designed for systems with distributed memory. Even though assumptions valid for distributed memory architectures can be adhered to in a shared memory system, they are nonetheless more restrictive than they need to be. This causes the potential of shared memory to be wasted.

Much of the computations in oceanographic models is solving systems of linear equations. Solving systems of linear equations can be reduced to matrix multiplication, which is inherently well suited for parallelisation. However, depending on the particular numerical scheme used, other task decompositions might be better suited. For example in the alternate direction implicit method, (ADI) [13] each row or column of the domain (depending on the current algorithm step) can be computed independently. This constitutes a good task decomposition, but requires shared memory due to the algorithm's row/column alternating nature. Other numerical schemes might have some other good decompositions.

In shared memory systems task decomposition can be accomplished in such a variety of ways that it has to be considered on a case by case basis – there is no silver bullet. The block based decomposition that was justified for distributed memory systems causes a number of problems that have to be handled by sacrificing performance. These problems are mainly communication, memory and computation overheads.

The block decomposition used in the 3D-CEMBS model and many other oceanographic models [5, 15] requires the so called halo region (see Figure 2). The values in the cells in the halo region need to be exchanged between neighbouring blocks – this is an obvious overhead. However, that is not the only overhead. The cells in the halo region need to be stored in memory – that means more memory is required after decomposition than for the undivided task. The model manual [16] suggests blocks of 20-40 cells in each dimension. With 2 cells wide halo region that makes 10-20% more points to be stored in memory. Those cells are not only stored, but also need to be computed, so if the blocks are small, the overhead becomes significant.

The domain decomposition to small blocks also has another disadvantage. It makes it harder to use GPGPU technologies like OpenACC. GPUs are excellent in matrix based computations, but if the matrix is divided into relatively small blocks, then more effort is put into block management than into floating point computations, making the use of GPGPU infeasible.

Block based task decomposition and distribution on distributed memory systems is static. The distribution is done once at the beginning of the model run. With the use of shared memory the task distribution can be dynamic between threads running on the same node. This can help to better utilise the hardware, thanks to better load balancing. With static load balancing all decisions need to be made a priori based on some assumptions on the runtime environment – assumptions concerning hardware parameters as well as resource utilisation by other software. However, the runtime environment is not static, but changes with time. Even if the parallel tasks were evenly distributed among processes or threads, the tasks might finish at different times. This can be caused by a number of reasons. For example if threads read or write data to a persistent storage, the storage access times might not be deterministic, which in effect might lead to different execution times of theoretically identical tasks. Another example might be when tasks are executed on hardware with technologies like hyper-threading, where two processes or threads executed on a single CPU core compete for CPU resources. Such examples are numerous, therefore employing dynamic task distribution between CPU cores should lead to better hardware utilisation and shorter execution times.

On a shared memory hardware architecture dynamic load balancing is fairly easy. A typical approach is to establish a thread pool and a task queue. As soon as a thread finishes one task it gets a new task from the queue. The greater the number of parallel tasks in comparison with the thread pool size the more evenly the tasks can be distributed. Nevertheless the tasks themselves cannot be too small, because that might lead to using more time on task management than on computations themselves. Therefore a proper task decomposition is the key for better parallelisation and hardware utilisation. Moreover, some kind of priority queue can be employed for better load balancing if for example, the tasks are unevenly sized and the task size can be easily estimated.

With a distributed memory architecture, such dynamic load balancing is much harder and might come at a significant computational price making it fruitless or even counter-productive. However, since today's hardware allows execution of several dozens of threads with shared memory, this can be easily exploited to balance the load within a single node and thus shorten execution times on individual nodes.

Hierarchical parallelisation architecture

MPI, OpenMP and OpenACC are complementary technologies. They should be used in tandem for the best hardware utilisation. Even though some use cases might overlap between those technologies, their design assumptions are different and therefore should be treated differently. Usage patterns that are efficient for MPI might be suboptimal for OpenMP and counter-productive in case of OpenACC. For optimal hardware utilisation a hierarchical parallelisation architecture can be employed.

Since high resolution oceanographic models might require not several dozen, but over a hundred CPU cores for operational forecasts to execute within a satisfactory time limit, MPI or other distributed memory based technologies are needed to facilitate communication between several computer nodes. However, within a single node with shared memory a thread based parallelisation solution might be better. It is important to note that a different task decomposition might be required for task distribution between CPU cores than that for tasks distributed between computer nodes. The third level of parallelisation might be the GPGPU usage. GPGPU parallelisation is different than the process or thread based parallelisation, therefore a proper task decomposition has to be implemented. Another thing to consider when employing GPGPU is the number of accelerators available and how that number corresponds to the CPU core count. Proper distribution of tasks between threads and accelerators is crucial.

SUMMARY

The performed code analysis and profiling of the 3D-CEMBS model showed that the model's bottleneck are not the floating point computations, but communication and memory management.

To improve memory management the code should be refactored to remove global variable usage. Since Fortran, the dominant language the model was written in, is an old programming language, it encourages old programming techniques that are no longer adequate for today's needs. Adhering to modern programming paradigms could improve maintainability of the code and even have a positive influence on model performance.

The main performance problem of the 3D-CEMBS model is the communication bottleneck. It is caused by the task decomposition that was designed for distributed memory systems. Changing the task decomposition and utilising OpenMP or other thread based technology could improve the model performance. Choosing the right task decomposition for oceanographic models is a topic open for research.

Since floating point computation performance is not the bottleneck in 3D-CEMBS, the usage of OpenACC cannot improve the model performance. However, if the communication overhead is reduced or eliminated and numerical computations become the dominant part of the model, then using OpenACC or other GPGPU technology could be once again considered.

References:

[1] Dijkstra, E. W. (1972). The humble programmer. Communications of the ACM, 15 (10), 859–866.

[2] Dzierzbicka-Głowacka, L., Jakacki, J., Janecki, M., Nowicki, A. (2013). Activation of the operational ecohydrodynamic model (3D CEMBS) – the hydrodynamic part. OCEANOLOGIA, 55 (3), 519–541.

[3] Dzierzbicka-Głowacka, L., Janecki, M., Nowicki, A., Jakacki, J. (2013). Activation of the operational ecohydrodynamic model (3D CEMBS) – the ecosystem module. OCEANOLOGIA, 55 (3), 543–572.

[4] Dzierzbicka-Głowacka, L., Nowicki, A., Janecki, M. (2014). The Automatic Monitoring System for 3D-CEMBSv2 in the Operational Version. Journal of Environmental Science and Engineering Technology, 2014 (1), 1–9.

[5] Funkquist, L., Kleine, E. (2007). An introduction to HIROMB, an operational baroclinic model for the Baltic Sea. REPORT OCEANOGRAPHY, 37.

[6] HSA Foundation (2015). HSA Platform System Architecture Specification.

[7] Jones, P. W., Worley, P. H., Yoshida, Y., White III, J. B., Levesque, J. (2005). Practical performance portability in the Parallel Ocean Program (POP). Concurrency: Practice and Experience, 17 (10), 1317–1327.

[8] Kerbyson, D. J., Jones, P. W. (2005). A performance model of the Parallel Ocean Program. International Journal of High Performance Computing Applications, 19 (3), 261–276.

[9] Message Passing Interface Forum (1997). MPI-2: Extensions to the Message-Passing Interface.

[10] Nowicki, A., Dzierzbicka-Głowacka, L., Janecki, M., Kałas, M. (2014). Assimilation of the satellite SST data in the 3D CEMBS model. OCEANOLOGIA, 57 (1).

[11] OpenACC-Standard.org (2013). The OpenACC Application Programming Interface.

[12] OpenMP Architecture Review Board (2013). OpenMP Application Program Interface.

[13] Peaceman, D. W., Rachford Jr., H. H. (1955). The numerical solution of parabolic and elliptic differential equations. Journal of the Society for Industrial and Applied Mathematics, 3 (1), 28–41.

[14] Piotrowski, P. (2014). Running WAM wave model on GPGPU. In 7th EuroGOOS conference. Lisbon.

[15] Singhal, S., Aneja, S., Liu, F., Real, L. V., George, T. (2014). IFM: A Scalable High Resolution Flood Modeling Framework. Lecture Notes in Computer Science, 8632, 692–703.

[16] Smith, R., Jones, P., Briegleb, B., Bryan, F., Danabasoglu, G., Dennis, J., Dukowicz, J., Eden, C., Fox-Kemper, B., Gent, P., Hecht, M., Jayne, S., Jochum, M., Large, W., Lindsay, K., et al. (2010). The Parallel Ocean Program (POP) Reference Manual. Los Alamos National Laboratory.

otrzymano/received: 20-04-2015
zaakceptowano/accepted: 02-06-2015
opublikowano/published: 26-06-2015

Adres do korespondencji:
e-mail: ppiotrowski@im.gda.pl

Tables: 1
Figures: 2
References: 16

DOI: 10.5604/12307424.1158133
www.bullmaritimeinstitute.com/fulltxt.php?ICID=1158133