Parallelization of a Large-Scale Watersheds Hydrological Model using CPU and GPU

Henrique R. A. Freitas, Celso L. Mendes

National Institute for Space Research (INPE)
PO BOX 515 – 12.227-010 – São José dos Campos – SP – Brazil

{henrique.renno,celso.mendes}@inpe.br

Abstract. *Hydrological models are commonly employed to calculate water flows on rivers and watersheds for the analysis of extreme events in nature. Computations in these models can grow depending on the numerical method, and also on the spatial and temporal resolutions, thus affecting the model efficiency and utility. This work parallelizes the MGB hydrological model on either CPU with OpenMP or GPU with OpenACC, respectively, aiming at the improvement in performance by employing computing resources of an HPC system. An analysis of the sequential and parallel executions is presented together with the runtime, speedup, efficiency, and load balance achieved.*

1. Introduction

Hydrological modeling is commonly employed in both research and industry, and is primarily concerned with water flows on rivers and watersheds. Applications include the analysis of extreme events (floods and droughts), forecast of river discharge etc.

The main equations associated to water flow processes are PDEs known as the Saint-Venant equations, mainly used in rainfall-runoff models [Paiva et al. 2011]. Depending on the numerical method used to solve these equations, the accuracy and quality of the model solution are improved, although more computational effort may be required. Moreover, the model performance and utility are also affected by the spatial and temporal resolutions because runtimes predominantly depend on domain size.

Parallel computing is a standard way to reduce runtimes with widely available multiprocessor computers (CPUs), and graphics processing units (GPUs) used for parallel computations. Many works investigated parallelization of hydrological models: CalTWiMS (MPI) [Pau and Sanders 2006], JFLOW (GPU) [Lamb et al. 2009], LISFLOOD-FP (OpenMP) [Neal et al. 2009], and LISMIN (GPU) [Sarates 2015].

The objective of this work is to parallelize the MGB hydrological model on either CPU with OpenMP or GPU with OpenACC, respectively, by employing computing resources of an HPC system, showing the improvement in performance, and presenting the runtime, speedup, efficiency, and load balance achieved.

2. MGB hydrological model

The MGB (Modelo de Grandes Bacias) hydrological model is developed at the IPH-UFRGS in Brazil focusing on improving the knowledge of hydrological processes in large-scale watersheds, particularly in the South America region.

The model has 53 Fortran90 source files with modules of main, calibration and inertial model variables, and three hydrological units represent the spatial discretization: catchments, subbasins, and hydrological response units (HRUs). Catchments and subbasins are regions that contribute water to drainage network segments and outlet points, respectively, whereas HRUs are regions of similar hydrological behavior.

The MGB model simulates 1D propagation of water flows from the inertial simplification of the Saint-Venant equations (figure 1a) [Fan et al. 2014] formed by the continuity (homogeneous hyperbolic PDE of convection) and the momentum (pressure, bed gradients, and friction) equations, where h is water height, q is discharge, y = h+z is water level relative to elevation z, g is the acceleration of gravity, n is the Manning coefficient, t is time, and x is longitudinal distance.

These equations are solved with forward in time and centered in space finite difference approximations (figure 1b) for an explicit numerical scheme (figure 1c) using initial and boundary conditions, where z is the bottom of the river elevation, NC is number of catchments, and i and k are spatial and temporal indexes, respectively. The scheme loops through each catchment with the inertial model that is comprised of 3 routines (figure 1d) that calculate stable time steps *tflood* for $\alpha = 0.7$ (1), water height and discharge at position i+1/2 (2,3), and water height and water level at position i (4,5).

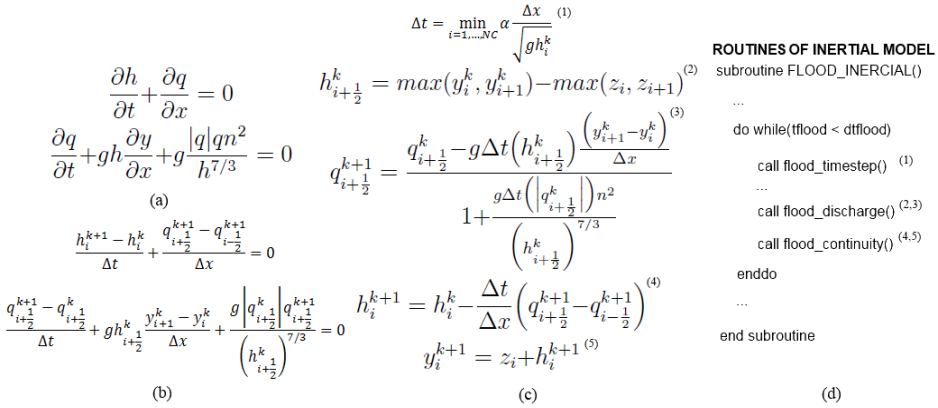

Figure 1 - MGB model: (a) continuity and momentum equations, (b) finite difference approximations, (c) explicit numerical scheme, and (d) Fortran routine of inertial model

3. Parallelization of the MGB model

This work aims at parallelizing loops present in the most time-consuming routines of the MGB model on either CPU with OpenMP or GPU with OpenACC, respectively, for the improvement in performance by employing computing resources of the Laquibrido cluster [Mendes 2016], maintained by LAC/INPE and acquired with funds from the government of Brazil. This HPC system has nodes with 2 Intel Xeon E5-2660v2 x86 processors of 2.2 GHz (10 cores each, 2 threads per core for 40 virtual threads as 40 CPUs), 128 GB of RAM memory, and 2 NVIDIA Tesla K20m GPUs of 705MHz.

OpenMP is an API that parallelizes applications with multiprocessor computers in shared memory environments. The API is designed as a set of compiler directives and a library of functions that dynamically create threads for parallel execution, usually of loops with independent instructions. OpenACC is a standard of directives that supports GPU devices. Transfer of data between **host** (CPU) and **device** (GPU) is necessary for intermediate computations to be performed on the device, and results to be stored back into main memory.

The MGB model is run for two test cases: Purus, one of the main tributaries of the Amazon river, and Niger, the largest river in West Africa. The Purus and Niger input data have 1984 catchments, 16 subbasins, 9 HRUs for 4747 time steps, and 4307 catchments, 9 subbasins, 11 HRUs for 5800 time steps, respectively.

By instrumenting the MGB model for profiling with the pgf90 compiler, three routines of the inertial model were identified as the most time-consuming routines. Table 1 shows information from the profiling with CPU times of the sequential execution of the MGB model for the two test cases.

Table 1 – Profiling of sequential execution of the MGB model

	Purus			Niger		
Routine	**Calls**	**Runtime (s)**	**Percentage of model runtime (%)**	**Calls**	**Runtime (s)**	**Percentage of model runtime (%)**
flood_continuity	1181973	92.55	46.16	633649	104.23	45.29
flood_discharge	1181973	75.22	37.52	633649	85.95	37.35
flood_timestep	1181973	24.89	12.42	633649	25.56	11.11
Other routines	-	7.82	3.90	-	14.37	6.25
MGB model	-	200.48	100	-	230.11	100

The routines flood_timestep, flood_discharge, and flood_continuity represent 96.1% and 93.75% of model runtime for the Purus and Niger test cases, respectively. Therefore OpenMP and OpenACC directives were used in loops that iterate over catchments with independent instructions, setting private variables and reduction operations where necessary.

For instance, the OpenMP and OpenACC directives used in the routine flood_timestep are in table 2, and the directives used in the other routines are similar with minor differences. The *static* schedule is used for an even workload distribution because the computations assigned to each thread are practically the same, i.e., the number of instructions executed by each thread are approximately equal, so that other data decomposition scheme such as *dynamic*, or *guided* result in a worse performance.

Table 2 – OpenMP and OpenACC directives for the routine flood_timestep

API	Compiler directive
OpenMP	!$OMP parallel do private (hmaxfl,dtflood) reduction(min:dttest) schedule(static)
OpenACC	!$ACC data copyin(SRIO,Hfl) !$ACC parallel loop private(hmaxfl,dtflood) reduction(min:dttest)

4. Preliminary results

Results in table 3 for the Purus and Niger test cases with 1900 and 4300 catchments, respectively, include the sequential (as 1 thread) and parallel runtimes (RT; wall times), speedup (SP; sequential over parallel runtime), efficiency (EF; speedup over number of threads), and load balance (LB; difference between maximum and minimum thread times) using either threads with OpenMP or GPU with OpenACC.

Table 3 – Performance of parallelized routines of the MGB model (in seconds)

Purus	**Routine**												**MGB model**
	flood_timestep				flood_discharge				flood_continuity				
CPU Threads	RT	SP	EF	LB	RT	SP	EF	LB	RT	SP	EF	LB	SP
1	21.7	1.0	-	-	218.2	1.0	-	-	91.4	1.0	-	-	1.0
2	12.0	1.8	0.9	12.0-11.7 = 0.3	115.6	1.8	0.9	115.6-109.0 = 6.6	50.0	1.8	0.9	50.1-44.1 = 6.0	1.5
5	5.9	3.6	0.7	5.9-5.7 = 0.2	49.8	4.3	0.9	49.8-45.9 = 3.9	19.3	4.7	0.9	19.3-17.3 = 2.0	2.5
10	6.1	3.5	0.3	6.1-3.7 = 2.4	27.3	7.9	0.8	27.3-25.3 = 2.0	15.3	5.9	0.6	15.4-10.2 = 5.2	2.9
20	9.7	2.2	0.1	9.7-4.7 = 5.0	16.1	13.5	0.7	16.0-13.4 = 2.6	15.7	5.8	0.3	15.8-7.8 = 8.0	2.8
25	10.8	2.0	0.1	10.8-8.5 = 2.3	14.4	15.0	0.6	14.5-11.6 = 2.9	16.3	5.5	0.2	16.3-10.4 = 5.9	2.6
1 GPU	44.2	0.5	-	-	25.1	8.7	-	-	322.9	0.3	-	-	0.9
Niger	**Routine**												**MGB model**
	flood_timestep				flood_discharge				flood_continuity				
CPU Threads	RT	SP	EF	LB	RT	SP	EF	LB	RT	SP	EF	LB	SP
1	22.7	1.0	-	-	226.0	1.0	-	-	88.8	1.0	-	-	1.0
2	12.6	1.8	0.9	12.6-11.9 = 0.7	127.7	1.8	0.9	127.7-113.6 = 14.1	51.4	1.7	0.9	51.4-43.4 = 8.0	1.5
5	5.6	4.0	0.8	5.6-5.4 = 0.2	53.5	4.2	0.8	53.5-48.8 = 4.7	22.8	3.9	0.8	22.9-16.7 = 6.2	2.8
10	4.0	5.6	0.6	4.0-3.5 = 0.5	28.1	8.0	0.8	28.1-26.3 = 1.8	12.1	7.3	0.7	12.1-8.7 = 3.4	3.7
20	5.5	4.0	0.2	5.5-4.2 = 1.3	17.9	12.6	0.6	18.0-13.4 = 4.6	9.1	9.7	0.5	9.1-5.1 = 4.0	3.3
25	6.0	3.8	0.2	6.0-4.1 = 1.9	15.2	14.8	0.6	15.2-11.5 = 3.7	10.5	8.4	0.3	10.5-5.0 = 5.5	3.3
1 GPU	22.8	1.0	-	-	12.6	17.9	-	-	350.6	0.3	-	-	0.9

The runtimes of the routine flood_timestep are lower with either 5 (Purus) or 10 (Niger) threads on CPU because this routine has a small number of instructions (low granularity), and in both test cases the load balance time differences are lower with 5 threads. Performance on GPU is low due to the reduction operation, i.e., besides computations, logical instructions demand

more GPU effort. Figure 2 shows for several catchments that the routine flood_discharge (high granularity) presents GPU runtimes similar to the lowest CPU runtimes achieved with 25 threads. The routine flood_continuity reaches the lowest runtimes with either 10 (Purus) or 20 (Niger) threads on CPU, and a poor performance on GPU because this routine has a sequential inner loop for a table search of an interpolation process that hinders GPU parallel use.

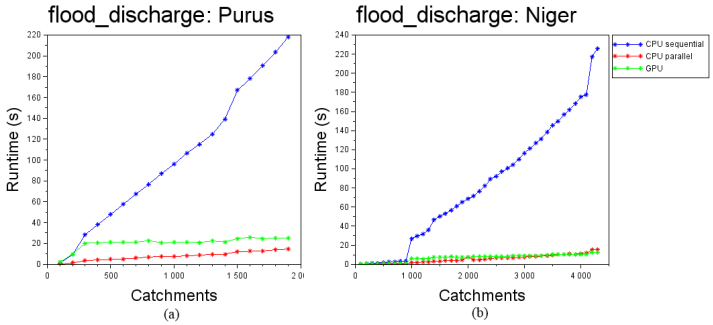

Figure 2 – CPU sequential, and CPU and GPU parallel runtimes of the routine flood_discharge for the (a) Purus, and (b) Niger test cases

5. Conclusions

This work aimed at parallelizing the MGB hydrological model in an HPC system for the improvement of performance. The parallelization was implemented on either CPU with OpenMP or GPU with OpenACC for two different test cases.

From the sequential execution of the MGB model, three routines were identified as the most time-consuming, and subsequently parallelized in loops. The routine flood_timestep achieved better performance on CPU using few threads in both test cases with the workload evenly distributed. The routines flood_discharge and flood_continuity presented low runtimes on CPU with many threads, the former with similar runtimes on GPU for the Niger test case.

As future work, the MGB model compilation will include flags for AVX vector instructions to further optimize the CPU performance. NUMA effects will be analyzed to detect if the HPC system hardware architecture impacts the behavior of the parallel implementation. A potential contribution will be the use of hardware counters to define measures for an automatic optimal CPU/GPU parallelization. A hybrid CPU/GPU scheme will be investigated.

References

Fan, F. M. and Pontes, P. R. M. and Paiva, R. C. D. (2014) "Avaliação de um método de propagação de cheias em rios com aproximação inercial das equações de Saint-Venant", RBRH – Revista Brasileira de Recursos Hídricos, v. 19, n. 4, pages 137-147.

Lamb, R. and Crossley, A. and Waller, S. (2009) "Fast 2D floodplain modeling using computer game technology", Flood Risk Management: Research and Practice.

Mendes, C. L. (2016) "Cluster LAQUIBRIDO", http://www.lac.inpe.br/~celso/LAQUIBRIDO.html.

Neal, J. C. and Fewtrell, T. J. and Trigg, M. (2009) "Parallelisation of storage cell flood model using OpenMP", Environmental Modelling & Software, v. 24, pages 872-877.

Paiva, R. C. D. and Collischonn, W. and Tucci, C. E. M. (2011) "Large scale hydrologic and hydrodynamic modeling using limited data and a GIS based approach", Journal of Hydrology, v. 406, pages 170-181.

Pau, J. C. and Sanders, B. F. (2006) "Performance of parallel implementations of an explicit finite-volume shallow-water model", Journal of Computing in Civil Engineering, v. 20, n. 2, pages 99-110.

Sarates, A. S. (2015), Optimizing Two-dimensional Shallow Water Based Flood Hydrological Model with Stream Architectures, Master Thesis, Universidade Federal do Rio Grande do Sul.