Roofline Analysis and Performance Optimization of the MGB Hydrological Model

Henrique R. A. Freitas[1], **Celso L. Mendes**[2]

[1]Image Processing Division
National Institute for Space Research
São José dos Campos – SP – Brazil

[2]Laboratory for Computing and Applied Mathematics
National Institute for Space Research
São José dos Campos – SP – Brazil

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

Abstract. *The Roofline model gives insights about the performance behavior of applications bounded by either memory or processor limits, providing useful guidelines for performance improvements. This work uses the Roofline model on the analysis of the MGB model that simulates hydrological processes in large-scale watersheds. Real-world input data are used to characterize the performance on two multicore architectures, one with only CPUs and one with CPUs/GPU. The MGB model performance is improved with optimizations for better memory use, and also with shared-memory (OpenMP) and GPU (OpenACC) parallelism. CPU performance achieves* $42.51\,\%$ *and* $50.17\,\%$ *of each system's peak, whereas GPU performance is low due to overheads caused by the MGB model structure.*

1. Introduction

Most hardware architectures of current computers comprise hierarchical memory sets, fused multiply-add (FMA) and vector instructions, separate functional units, and more diverse features that can be exploited for performance improvements in software applications [Hennessy and Patterson 2007, Dolbeau 2015]. Besides these characteristics, computer systems are now multicore commonly consisting of independent CPUs, each with several cores (some multithreaded) that perform concurrent computations [Hill and Marty 2008]. Thus, building efficient and portable software that benefit from the hardware potentials requires tools that reveal the behavior of applications under such systems.

The Roofline model [Williams et al. 2009] is a bound and bottleneck-based tool for a visual representation of the memory and processor capabilities of a computer system that identifies the factors limiting the performance of an application, and offers guidelines for choosing which memory/compute optimizations are necessary to improve the performance. The model relates the flops per byte transferred or arithmetic intensity (AI) to floating-point performance (flops/s). Generally, memory optimizations are first recommended to increase the AI before trying compute optimizations.

The contribution of this work is the performance analysis with the Roofline model of the MGB hydrological model [Collischonn 2001] that is widely employed for the understanding of hydrological processes in large-scale watersheds. Performance is first improved with optimizations on the implementation for better memory use, and then with

shared-memory and GPU parallelism using OpenMP and OpenACC, respectively, for productivity and portability purposes. The parallel executions on CPU were conducted with different numbers of threads for a selected thread binding since affinity affects performance.

1.1. Structure of the Work

The structure of this paper is as follows. The MGB model is described in Section 2. Section 3 gives the characteristics of the computer systems used as testbed. Section 4 presents the Roofline analysis of the MGB model, and the details of the memory/compute optimizations performed. Results of the performance evaluation, and optimization analysis are in Section 5. Related works that are based on Roofline model analysis are included in Section 6. Conclusions are summarized in Section 7.

2. MGB Hydrological Model

The "Modelo de Grandes Bacias" (MGB) [Collischonn 2001] is a hydrological model developed at the IPH-UFRGS in Brazil focusing on hydrological processes in large-scale watersheds, particularly in the South America region. The model simulates 1D propagation of water flows on rivers and watersheds [Fan et al. 2014] for the analysis of extreme events (floods and droughts), forecast of river discharge, estimation of the effects of climate change due to vegetation and soil cover etc [Paiva et al. 2011].

Numerical solutions are computed from the inertial simplification of the Saint-Venant equations, i.e., continuity (1) and momentum (2) 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.

$$\frac{\partial h}{\partial t}+\frac{\partial q}{\partial x}=0 \tag{1}$$

$$\frac{\partial q}{\partial t}+gh\frac{\partial y}{\partial x}+g\frac{|q|qn^2}{h^{7/3}}=0 \tag{2}$$

These equations are solved from initial and boundary conditions with forward in time and centered in space finite difference approximations that define an explicit numerical scheme (3)-(7), where N is the number of catchments of the watershed, and i and k are spatial and temporal indexes, respectively.

$$\Delta t{=}min(\alpha\frac{\Delta x}{\sqrt{gh_i^k}}), i{=}1,2,\ldots,N \tag{3}$$

$$h_{i+\frac{1}{2}}^k{=}max(y_i^k,y_{i+1}^k){-}max(z_i,z_{i+1}) \tag{4}$$

$$q_{i+\frac{1}{2}}^{k+1}{=}\frac{q_{i+\frac{1}{2}}^k-g\Delta t\left(h_{i+\frac{1}{2}}^k\right)\frac{\left(y_{i+1}^k-y_i^k\right)}{\Delta x}}{1+\frac{g\Delta t\left(\left|q_{i+\frac{1}{2}}^k\right|\right)n^2}{\left(h_{i+\frac{1}{2}}^k\right)^{7/3}}} \tag{5}$$

$$h_i^{k+1}{=}h_i^k-\frac{\Delta t}{\Delta x}\left(q_{i+\frac{1}{2}}^{k+1}-q_{i-\frac{1}{2}}^{k+1}\right) \tag{6}$$

$$y_i^{k+1}{=}z_i{+}h_i^{k+1} \tag{7}$$

The scheme is divided into three routines called successively for each MGB model iteration while the inertial model time step Δt satisfies stability, looping through catchments to compute stable time steps for α=0.70 (3), water height and discharge at $i+1/2$ (4)-(5), and water height and level at i (6)-(7).

The spatial discretization of the MGB model consists of three hydrological units: catchments, subbasins, and hydrological response units (HRUs). Catchments and subbasins are regions that contribute water to drainage network segments and to outlet points, respectively, whereas HRUs are regions of similar hydrological behavior. Figure 1 illustrates the discretization of the river length according to (3)-(7).

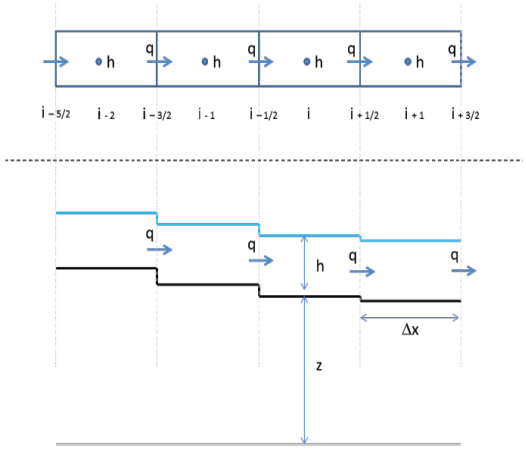


Figure 1. River length discretization for the inertial model [Fan et al. 2014].

Besides the Saint-Venant equations, the MGB model also computes evapotranspiration from the Penman-Monteith equations [Shuttleworth 1993], and vertical water balance in soil [Collischonn 2001] to further compute surface, subsurface and groundwater flows. The overall performance of the model is affected not only by the numerical method, but also by the spatial and temporal resolutions of the input dataset because execution times depend predominantly on domain size, which constitutes a key factor that significantly impacts the utility of the model.

3. Computer Systems and Datasets for Performance Evaluation

3.1. Computer Systems

The computational testbed consists of two computer systems, namely **urano** and **z800**, maintained by the Image Processing Division (DPI) at the National Institute for Space Research (INPE) in Brazil. Both systems include CPUs based on the Intel Westmere-EP microarchitecture without FMA instructions, but with 128-bit SSE4 vector instructions, and hyper-threading. The **z800** has an additional GPU. The **urano** system is a high performance server with Linux Fedora Core 23, whereas the **z800** system is a desktop workstation with Linux Ubuntu 16.04.1.

Details of the hardware specifications of each system are in Table 1 that includes the peak double-precision processor performance for all cores, and the peak bandwidth of caches and DRAM memory. The peak bandwidths were measured with the Intel

Table 1. Computer systems used for performance evaluation

Name		urano	z800	
Processor		2 Intel Xeon X5670	2 Intel Xeon E5620	1 NVIDIA GeForce GTX 760
Cores / Threads[a]		6 / 24	4 / 16	1152
Frequency (GHz)		2.93	2.40	1.06
Peak performance (Gflops/s)		70.32	38.40	1221.12
Memory size (GB)[b]		192	11	2
Peak bandwidth (GB/s)[b]		12.80	10.66	96.13
Stream bandwidth (GB/s)		11.33	10.52	-
L1	Size (bytes)	32 K	32 K	
	Peak bandwidth (GB/s)	748.61	555.51	
L2	Size (bytes)	256 K	256 K	
	Peak bandwidth (GB/s)	337.70	326.78	
L3	Size (bytes)	12 M	12 M	
	Peak bandwidth (GB/s)	163.99	135.33	

[a]Only cores are included for GPU. [b]CPU values are associated with DRAM memory.

Memory Latency Checker (Intel MLC) [Viswanathan et al. 2013], a tool designed to measure memory latency and bandwidth [Ruggiero 2008].

The **urano** and **z800** systems have DDR3 DRAM memory of bandwidths $12.80\,\mathrm{GB/s}$ ($800\,\mathrm{MHz}$, 2 channels) and $10.66\,\mathrm{GB/s}$ ($1333\,\mathrm{MHz}$, 1 channel). The **z800** has a GDDR5 GPU memory of $3004\,\mathrm{MHz}$ and $256\,\mathrm{bit}$. Potential memory bandwidths were found through the STREAM benchmark [McCalpin 1995].

3.2. Datasets

Different datasets were used as input to the MGB model with hydrometeorogical data from two regions of the world where the Purus and Niger rivers, respectively, are located. The Purus river is one of the main tributaries of the Amazon river in Brazil, with drainage area of $370\,000\,\mathrm{km}^2$, and average discharge of $11\,\mathrm{km}^3/\mathrm{s}$ [Paiva et al. 2011]. The Niger river is the largest river in West Africa, with drainage area of $657\,000\,\mathrm{km}^2$ [Fleischmann et al. 2017], and average discharge of $15\,\mathrm{km}^3/\mathrm{s}$ [Zwarts et al. 2005].

Each dataset includes discretization data of catchments, subbasins, and HRUs (spatial) for each time step (temporal). The Purus dataset has 1984 catchments, 16 subbasins, and 9 HRUs for 4747 time steps, whereas the Niger dataset has 4307 catchments, 9 subbasins, and 11 HRUs for 5800 time steps. The MGB model was run in simulation mode with preset calibration parameters.

4. Roofline Analysis and Optimizations

In this section, we present our analysis of the MGB model, based on traditional code profiling techniques and on the Roofline approach. Possible optimizations for the MGB model are discussed based on the observations derived from the Roofline analysis.

4.1. Profiling of the MGB Model

Profiling of the MGB model identified three routines of the inertial model as the most time-consuming routines, namely `timestep` (STE), `discharge` (DIS), and `continuity` (CON). Table 2 shows the CPU runtime of each routine, and the percentage relative to the CPU runtime of the MGB model for executions on each system with the Purus and Niger input datasets. From the two systems, the routines represent on average $80.64\,\%$ (Purus) and $85.23\,\%$ (Niger) of the MGB model runtime. The MGB model was compiled with the `ifort` 17.0.7 compiler using the `-O2` flag for automatic ILP and vectorization.

Table 2. Profiling of the MGB hydrological model

System		urano			z800		
Datasets		Routines					
		STE	DIS	CON	STE	DIS	CON
Purus	Runtime[a]	19.65	140.85	76.05	24.23	173.27	89.83
	% of MGB	6.83	48.93	26.42	6.67	47.69	24.73
	MGB runtime[a]	287.88			363.30		
Niger	Runtime[a]	22.69	163.99	82.76	28.24	201.28	98.10
	% of MGB	7.26	52.44	26.46	7.27	51.78	25.24
	MGB runtime[a]	312.73			388.69		

[a]Runtimes are in seconds.

4.2. Roofline Analysis

The Roofline model [Williams et al. 2009] is a valuable tool that provides insights about the behavior of applications on computer systems with diverse features and capabilities of memory and processor. The development and optimization of applications can be guided by the model for better utilization of the hardware potentials, thus resulting in performance improvements.

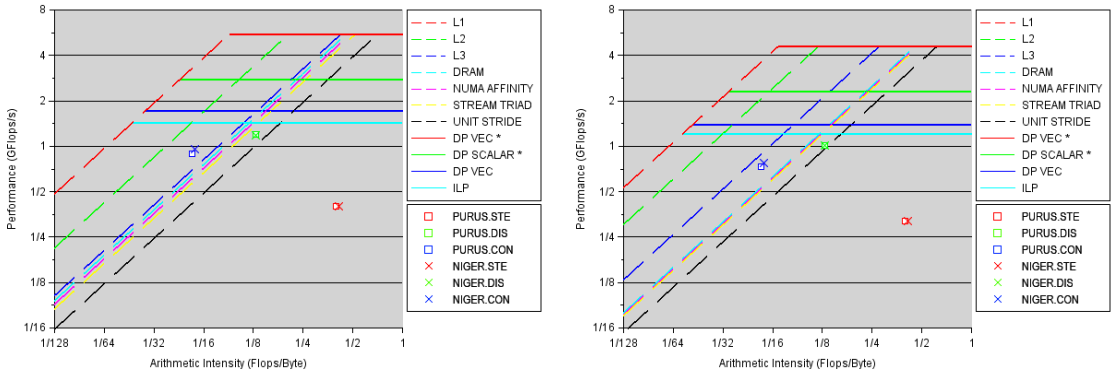
As mentioned in Section 1, the Roofline model uses the arithmetic intensity I (flops/byte) of an algorithm to identify whether its performance on a given system is bounded by either memory or processor limits. On a computer system with peak memory bandwidth B (bytes/s) and peak floating-point processor performance F (flops/s), the maximum attainable performance P for I is $P = \min\{B \times I, F\}$.

The original Roofline model considers the AI computed from off-chip memory transfers between the caches and DRAM memory, although it is not sufficient to fully describe the performance of applications on modern hardware architectures. Our work uses a more precise Roofline model referred to as Cache-Aware Roofline model (CARM) [Ilic et al. 2013], which also accounts for the on-chip memory traffic with data transfers between different cache levels for more accurate performance information.

In the present work, performance data were computed from hardware counters collected with the Performance API (PAPI) 5.6.0 [Terpstra et al. 2010]. The hardware counters used for the number of bytes are `PAPI_LD_INS` and `PAPI_SR_INS` that provide the number of memory instructions of loads and stores for bytes transferred from and to the L1 cache (write-back mode), respectively. Thus, the total number of bytes is

`8*(PAPI_LD_INS+PAPI_SR_INS)` because computations are in double precision. The hardware counter used for the number of floating-point operations is `PAPI_FP_OPS`.

Figure 2 exhibits the rooflines of the **urano** and **z800** systems, including ceilings for memory/processor capabilities of a single processor core (derived from Table 1), and marks for the routines evaluated. The routines `discharge` and `continuity` are located under memory ceilings, whereas the routine `timestep` is under a compute ceiling.



(a) (b)

Figure 2. Rooflines of (a) urano and (b) z800 systems with MGB routines marks.

Apart from the memory and processor ceilings already explained, the rooflines in Figure 2 include the memory ceilings for non-uniform memory access (NUMA) affinity and unit-stride access, and the processor ceilings for ILP and vectorization. These ceilings were determined with a microbenchmark that performs only load instructions and floating-point operations on one array, so that instruction/operation latencies affect the values of the ceilings. The theoretical ceilings (marked with star) do not account for such latencies.

The NUMA affinity ceiling characterizes the bandwidth of using a processor core that accesses data allocated to its local memory, whereas the unit-stride ceiling considers data that are accessed contiguously in memory. The ILP and vectorization ceilings are the performance with loop unrolling and with vector instructions, respectively. From the unit-stride and vectorization ceilings, the ridge points of the systems (minimum AI required for maximum performance) are I=0.23 (**urano**) and I=0.20 (**z800**).

4.3. MGB Model Optimizations

The routine `timestep` requires only compute optimizations because the AI is about 0.42 (Table 3), which is located under the compute ceilings, i.e., to the right of the ridge point. This routine executes few memory instructions (only two array accesses) to compute the stable time steps of the inertial model, but a minimum reduction operation limits the performance.

Even though the AI of the routine `discharge` is 0.13, the performance is very close to the top compute ceilings because this routine presents a large number of floating-point operations. Such performance can be improved not only with compute optimizations but also with memory optimizations, since it remains below the ceilings related to DRAM memory accesses, so a better cache memory use is beneficial for performance.

The routine `continuity` presents the lowest AI of approximately 0.06, but both rooflines indicate that the data used are kept in cache, and it can also be improved with compute optimizations (**urano**). However, gains in performance are hindered by a linear search performed in the routine that selects values from a table required for the interpolation of water levels and cross-sectional areas.

Table 3. Performance data of routines for complete runs of MGB model

System		urano			z800		
Datasets		Routines					
		STE	DIS	CON	STE	DIS	CON
Purus	Flops[a]	8.22	177.02	69.47	8.23	176.81	69.37
	Bytes[a]	19.71	1352.29	1261.22	19.76	1352.35	1261.28
	AI	0.42	0.13	0.06	0.42	0.13	0.05
Niger	Flops[a]	9.57	204.94	80.27	9.57	204.75	80.09
	Bytes[a]	22.37	1573.28	1409.27	22.40	1573.31	1409.30
	AI	0.43	0.13	0.06	0.43	0.13	0.06

[a]Flops and bytes are in billions.

The original scheme of the inertial model is exhibited in Figure 3 that includes the successive calls made to the routines while stable time steps of the inertial model accumulate to one time step of the MGB model. Each routine has a loop that iterates through the catchments to perform the required computations.

```
1  subroutine inertial_model()
2    inestep = 0.0 ! initialize total time steps
3    do while(inestep < mgbstep)
4      call timestep() ! inertial model time step: inedt
5      call discharge() ! water height and discharge at i+1/2
6      call continuity() ! water height and water level at i
7      inestep = inestep+inedt ! update total time steps
8    end do
9  end subroutine
```

Figure 3. Scheme of the inertial model.

Before applying optimizations with CPU/GPU parallelism, as the routines are called successively for each inertial model iteration, the algorithms of the routines were joined together as one routine. This modification ensures better memory use by maintaining most variables in cache, particularly for the routine `discharge`, and also reduces overheads associated with routine calls, thus improving the overall performance of the MGB model.

Figure 4(a) shows a simplified view of the algorithm of routine `continuity` with an OpenMP directive for parallelization on multiple CPUs [Dagum and Menon 1998], and the `static` schedule for an even workload distribution (loop iterations are load balanced). Similar OpenACC directives were used for parallelization on GPU [Wienke et al. 2012] (`!$ACC parallel loop private(...)`) with explicit transfer of data between host (CPU) and device (GPU) memories (Figure 4(b)), but the MGB model was compiled with the `pgf90` 18.10-1 compiler that supports OpenACC.

```
1  subroutine continuity()
2    !$OMP parallel do private(...) schedule(static)
3    do IC = 1,N ! loop through catchments
4      ! computations before linear search
5      do ! linear search
6        ! computations inside linear search
7        if (search is finished) then
8          exit ! exit linear search
9        end if
10     end do
11     ! computations after linear search
12   end do
13   !$OMP end parallel do
14 end subroutine
```

(a)

```
1  subroutine mgb()
2    mgbiter = 0 ! initialize total iterations
3    !$ACC data copyin(22 array variables)
4    do while(mgbiter < mgbiters)
5      ! CPU computations before inertial model
6      ! read new rainfall from input file
7      ! update rainfall,evapotranspiration,discharge on CPU
8      !$ACC update device(evapotranspiration,rainfall,discharge)
9      call inertial() ! call inertial model
10     !$ACC update host(discharge,area,height,level)
11     ! CPU computations after inertial model
12     mgbiter = mgbiter+1 ! update mgb model iteration
13   end do
14 end subroutine
```

(b)

Figure 4. (a) OpenMP in routine `continuity`, (b) MGB model scheme with OpenACC parallelization and CPU/GPU data transfers.

5. Experimental Results

This section presents the experimental results obtained from applying the optimizations previously mentioned based on the Roofline analysis. The modification of joining the three routines of the inertial model into one routine for better cache use reduced the runtimes of the MGB model, thus characterizing the first performance improvement.

Figure 5 shows sequential runtimes of the separate routines stacked on top of each other (STE/DIS/CON) compared to the joined routine (JOIN), and runtimes of the original (MGB.ORIG) and joined (MGB.JOIN) MGB model versions. On average, JOIN runtimes were $3.39\,\%$ (Purus) and $4.45\,\%$ (Niger) lower than STE/DIS/CON runtimes.

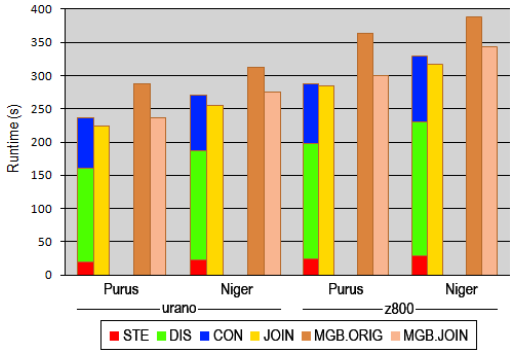


Figure 5. MGB model sequential CPU runtimes for separate/joined routines.

Meanwhile, MGB.JOIN runtimes were on average $17.68\,\%$ (Purus) and $11.79\,\%$ (Niger) lower than MGB.ORIG. This runtime reduction results from a decrease by a factor of three on the overheads of the number of routine calls, which approximately changed from $3.54\,\mathrm{M}$ to $1.18\,\mathrm{M}$ for Purus, and from $1.90\,\mathrm{M}$ to $0.63\,\mathrm{M}$ for Niger.

Cache use is another factor that contributes to the runtimes reduction of JOIN. On both systems, the L3 cache is shared between all the cores of a NUMA node, and the latency of a core to access the L3 cache is higher than the latency to access the L1/L2 private caches. Table 4 includes the L3 cache misses collected with `PAPI` for the separate and joined routines, where the latter presents fewer misses.

Table 4. L3 cache misses of the separate and joined routines

System	urano		z800	
Datasets	Routines			
	SEP[a]	JOIN	SEP[a]	JOIN
Purus	567267	463625	302278	248703
Niger	3356726	2949643	3241516	1204063

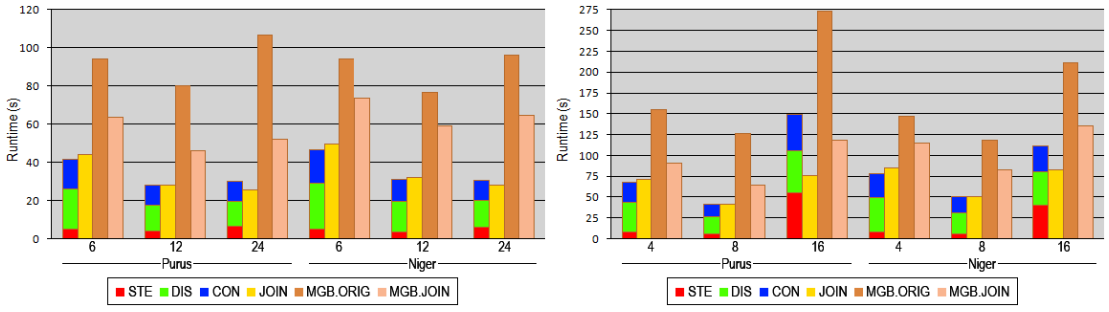
[a]SEP refers to the separate routines STE, DIS, and CON.

5.1. CPU Parallelism

Besides the performance improvements by joining the routines, the use of parallelism also reduced the runtimes of the MGB model. Since the computer systems have two processors, each processor on a separate NUMA node, and as the processors support hyper-threading, the numbers of threads chosen for conducting the parallel executions were the number of cores on a single processor, and the number of cores/threads on two processors.

Resource contention between threads is avoided by binding each thread to a single physical core for the cases where the number of OpenMP threads is not equal to the number of physical threads. This configuration is possible by setting two OpenMP environment variables, namely `OMP_PLACES` = **cores**, and `OMP_PROC_BIND` = **close**. Results of the shared-memory parallelization with OpenMP are exhibited in Figure 6.

The MGB.JOIN parallel runtimes are also lower than the MGB.ORIG runtimes because less overhead is introduced by routine calls in the inertial model. In both systems, the lowest runtimes are achieved with the number of threads equal to the number of all available cores because this configuration uses the largest number of concurrent threads with the least contention for resources. Overheads with 24 (**urano**) and 16 (**z800**) threads are not amortized by the low number of loop iterations.



(a) (b)

Figure 6. MGB model parallel CPU runtimes on (a) urano and (b) z800 systems, for a varying number of threads employed.

From the lowest runtimes of the parallel executions on CPU, the average performance of JOIN computed for the Purus and Niger datasets is $9.08\,\mathrm{Gflops/s}$ on **urano** and $5.94\,\mathrm{Gflops/s}$ on **z800**. The highest performance achievable (vectorization ceiling) with all the cores on **urano** and **z800** are $21.36\,\mathrm{Gflops/s}$ and $11.84\,\mathrm{Gflops/s}$, respectively. Therefore, the performance of JOIN is about $42.51\,\%$ (**urano**) and $50.17\,\%$ (**z800**) of the maximum possible performance on each system.

5.2. GPU Parallelism

The runtimes achieved with parallelization on GPU were higher than the optimal parallel CPU runtimes. Figure 7 exhibits the GPU runtimes including above each bar how many times these runtimes were higher than the average optimal CPU runtimes.

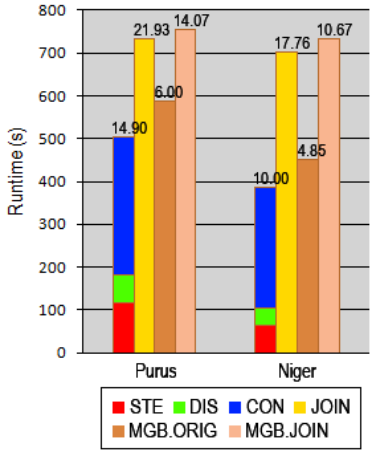


Figure 7. GPU runtimes of routines and MGB model.

GPU runtimes are high because too much OpenACC overhead is introduced by the millions of routine calls, which is not amortized by the thousands of loop iterations (number of catchments), i.e., computations are not GPU intensive. The routines `timestep` and `continuity` do not show performance gains with GPU because the former has a low number of computations and performs a reduction operation, whereas the latter includes a linear search that uses the parallel GPU resources poorly. GPU performance of routine `discharge` is slightly better for the larger number of computations (Flops in Table 3).

6. Related Work

Recent works analyzed the performance of computational applications based on Roofline model. In [Wittmann et al. 2018], more accurate performance limits are computed from an analytical performance model that captures the sequential and parallel execution phases of a sparse direct solver with different memory bandwidths for each phase. The phases are defined from a particular data structure used for efficient data storage in memory.

The methodology presented in [Yang et al. 2019] constructs a hierarchical Roofline model for GPUs that incorporates L1, L2, device memory and system memory bandwidths. A NVIDIA utility is used to collect the information necessary to compute the performance and arithmetic intensity of the applications processed on GPU. Three applications of different computational characteristics were evaluated.

In our work, the Roofline model consists of more general ceilings described in [Williams et al. 2009], which helped to identify a relationship between the performance of different routines of the MGB model. Moreover, we do not investigate performance analysis with the Roofline model on GPU because the GPU parallelization of the MGB model indicated that the model must be fully restructured for adequate GPU use, in order to avoid excessive GPU initialization overhead that limits the MGB model performance. To the authors' knowledge, no other work presents the Roofline analysis of the performance of a hydrological model.

7. Conclusion

This work presented the performance evaluation of the MGB hydrological model with analysis based on the Roofline model that provided the memory and processor capabilities of two computer systems presenting different hardware and performance characteristics, both with multiple CPUs and one with an additional GPU.

The Roofline model worked as a guideline for understanding the behavior of the MGB model executed with two distinct input datasets, and the performance data for the Roofline analysis were collected from hardware counters accessible through PAPI.

The most time-consuming routines of the MGB model were identified, where the modification of joining the routines for better memory use resulted in performance improvements, characterizing a first optimization. Furthermore, parallelization on multiple CPUs and on GPU was performed with OpenMP and OpenACC standards, respectively.

Parallelization on CPUs showed better performance than on GPU because one routine has few computations, and other performs a linear search. Moreover, overhead is introduced into GPU runtimes due to the large number of routine calls, and not so intensive computations. The best performance achieved on CPU uses the number of threads equal to the number of all available cores on each system for a particular thread binding.

As future work, NUMA effects will be analyzed with the parallel initialization of the MGB model variables by placing threads and data on the same NUMA node to minimize data access latency times for potential runtimes reduction. The use of different thread bindings will be investigated to identify whether thread placement plays a role on the MGB model performance. MGB model runs in calibration mode, which is used to define watersheds parameters, will also be considered for performance analysis because the calibration method is a computationally intensive task.

Acknowledgments

The authors are grateful to the research group of the “Instituto de Pesquisas Hidráulicas” (IPH) at the “Universidade Federal do Rio Grande do Sul” (UFRGS) for the availability of the MGB hydrological model, and also to the Image Processing Division (DPI) at the National Institute for Space Research (INPE) for the computational resources. This work was supported by the “Conselho Nacional de Desenvolvimento Científico e Tecnológico” (CNPq) in Brazil (process no. 140687/2016-5).

References

Collischonn, W. (2001). *Simulação hidrológica de grandes bacias*. PhD thesis, Universidade Federal do Rio Grande do Sul (UFRGS), Instituto de Pesquisas Hidráulicas (IPH).

Dagum, L. and Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Comp. Sci. Eng.*, 5(1):46–55.

Dolbeau, R. (2015). Theoretical peak FLOPS per instruction set on modern Intel CPUs. unpublished.

Fan, F. M., Pontes, P. R. M., Paiva, R. C. D., and Collischonn, W. (2014). Avaliação de um método de propagação de cheias em rios com aproximação inercial das equações de Saint-Venant. *R. Bras. Rec. Hídr.*, 19(4):137–147.

Fleischmann, A., Siqueira, V., Paris, A., Collischonn, W., Paiva, R. C. D., Pontes, P. R. M., Biancamara, S., Gosset, M., and Calmant, S. (2017). Representando interações entre hidrologia e hidrodinâmica em modelos de grande escala: estudo de caso no rio Níger, África. In *XXII SBRH - Simpósio Brasileiro de Recursos Hídricos*. ABRH - Associação Brasileira de Recursos Hídricos.

Hennessy, J. L. and Patterson, D. A. (2007). *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers, 4th edition.

Hill, M. D. and Marty, M. R. (2008). Amdahl's law in the multicore era. *IEEE Comp. Soc.*, 41(7):33–38.

Ilic, A., Pratas, F., and Sousa, L. (2013). Cache-aware roofline model: upgrading the loft. *IEEE Comp. Arch. Letr.*, 13(1):21–24.

McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. In *IEEE Comp. Soc. Tech. Cmte. Comp. Arch. Newsletter*, pages 19–25.

Paiva, R. C. D., Collischonn, W., and Tucci, C. E. M. (2011). Large scale hydrologic and hydrodynamic modeling using limited data and a GIS based approach. *J. Hydrol.*, 406(3-4):170–181.

Ruggiero, J. (2008). Measuring Cache and Memory Latency and CPU to Memory Bandwidth. Intel Corporation.

Shuttleworth, W. J. (1993). *Evaporation*, chapter 4, pages 1–53. Handbook of Hydrology. McGraw-Hill Education.

Terpstra, D., Jagode, H., You, H., and Dongarra, J. (2010). Collecting performance data with PAPI-C. In Heidelberg, S. B. ., editor, *Tools High Perf. Comp.*, pages 157–173. 3rd Parallel Tools Workshop.

Viswanathan, V., Kumar, K., Willhalm, T., Lu, P., Filipiak, B., and Sakthivelu, S. (2013). Intel Memory Latency Checker. Intel Corporation.

Wienke, S., Springer, P., Terboven, C., and an Mey, D. (2012). OpenACC - first experiences with real-world applications. In Berlin/Heidelberg, S., editor, *Lect. Notes Comp. Sci.*, volume 7484, pages 859–870. Euro-Par 2012 Parallel Proc.

Williams, S., Waterman, A., and Patterson, D. (2009). Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76.

Wittmann, M., Hager, G., Janalik, R., Lanser, M., Klawonn, A., Rheinbach, O., Schenk, O., and Wellein, G. (2018). Multicore performance engineering of sparse triangular solves using a modified roofline model. In *30th Int. Sym. Comp. Arch. High Perf. Comp.*, pages 233–241.

Yang, C., Kurth, T., and Williams, S. (2019). Hierarchical roofline analysis for GPUs: accelerating performance optimization for the NERSC-9 perlmutter system. In *Annual Cray Users Group Meeting (CUG'2019)*. Cray User Group.

Zwarts, L., van Beukering, P., Bakary, K., and Wymenga, E. (2005). The Niger, a lifeline - effective water management in the upper Niger basin. Altenburg & Wymenga ecological consultants.