20th International Congress on Modelling and Simulation, Adelaide, Australia, 1–6 December 2013
www.mssanz.org.au/modsim2013

A balancing act in heterogeneous computing – Developing the AWRA-Landscape data assimilation system

J.-M. Perraud[a], D. Collins[a], J. C. Bowden[a], T. Raupach [a,b], P. A. Manser[c], M.P. Stenson[a] and L.J. Renzullo[a]

[a] *Commonwealth Scientific and Industrial Research Organisation*
[b] *now: Environmental Remote Sensing Laboratory (LTE), École Polytechnique Fédérale de Lausanne (EPFL), GR C2 564, 1015 Lausanne, Switzerland).*
[c] *Manser Services Pty Ltd*
Email: jean-michel.perraud@csiro.au

Abstract: The Australian Water Resource Assessment Landscape (AWRA-L) model is a landscape surface water model running at a daily time step on 277,770 grid cells. To make the best use of observations to guide the model results, an Ensemble Kalman filter (EnKF) technique is used to assimilate remotely sensed satellite data. It executes one hundred instances of the model in parallel, leading to substantial computational challenges by hydrologic modelling standards. The AWRA-L data assimilation system was originally implemented in the R language. While it is a concise implementation, runtime performance is not adequate to perform experiments at the scale of Australia, even using computational clusters.

The AWRA-L data assimilation (DA) system must perform in exploratory research as well as in a broader operational model orchestrated by the software product Delft-FEWS (Flood Early Warning System) at the Bureau of Meteorology. The infrastructure available in each case is significantly different, as is the prioritisation of features. Software use in a research context requires execution speed as well as the possibility for the principal researcher to perform new experiments. These can be done on compute clusters, optionally with state of the art devices with Graphical Processing Units (GPU). The operational version of AWRA-L DA is designed to fit in a time stepping FEWS workflow. This workflow is such that AWRA-L DA must be run over the whole nation for a given time step before stepping temporally, a constraint not present in the research context. To reuse existing model code, the C# language is chosen for the core of AWRA-L DA, with a path to test and, if suitable, migrate to OpenCL for some of the computations. The operational system must scale well within a single process with multiple threads to maximize the use of the operational infrastructure. For the research needs it must also scale well across compute nodes, where network throughput is more likely to become a significant performance bottleneck.

In order to assess the performance of computing on GPU devices for the AWRA-L DA, we use OpenCL.NET to run OpenCL code from the C# system. AWRA-L DA is added to an AWRA system where an implementation of the model already exists for calibration purposes. Calibration and DA contexts differ, notably in terms of primacy of the temporal versus spatial dimensions of the problem for computations. A custom implementation of AWRA-L for DA is necessary. We trial a code template technology to manage these parallel implementations, where the use of object-oriented and generic programming is technically not sufficient to limit duplications. The final assessment of the system in terms of runtime shows that the C# implementation scales well in a multi-threaded mode up to six to eight cores, enabling the efficient use of the resources of the operational system. A multi-process version dividing the problem spatially is used for research purposes, complemented with a final collation of partial results. Combining multi-threading and multiple processes in a cluster environment, we can balance between multi-threading overheads and the input/output (I/O) bottlenecks of a large number of processes. A usage with thirty two processes, each with two threads, is chosen for research experiments. We project that the use of a native language such as C++ would have brought an additional raw runtime performance of ~40%. The gain would however be less in a real use case due to I/O throughput considerations. Our trial of OpenCL, even limited to portions of the code already shows an improvement of 50% over a single threaded pure C# version. These figures suggest that were more runtime performance gains necessary for a problem of gridded nature such as AWRA-L DA, there are compelling reasons to consider GPU enabled computations.

Keywords: *High performance computing, NET, OpenCL, R language, data assimilation*

1. INTRODUCTION

The Australian Water Resources Assessment system (AWRA) is a national-scale hydrologic workflow, designed to support two new nationally significant water information products published by the Australian Bureau of Meteorology (BoM), the annual National Water Accounts and Water Resources Assessments, and has been developed as part of the Water Information Research and Development Alliance (WIRADA) between the Commonwealth Scientific, Industrial and Research Organisation (CSIRO) and the BoM. The AWRA system described in Stenson et al. (2012) is composed of several loosely coupled modelling components, representing different aspects of the terrestrial water balance, including AWRA-R for river modelling, AWRA-G for groundwater, and AWRA-L for landscape processes. AWRA-L is a gridded land surface model, with each model instance representing approximately 25 km^2. AWRA-L was designed to make best use of all available observation to guide the internal model states, including many remotely sensed data products not normally associated with hydrologic modelling, such as evapotranspiration and soil moisture. One of the most novel uses of remotely sensed data is data assimilation, which has been applied to the AWRA-L model. The simultaneous development of research and operational software systems to support AWRA-L DA forms the subject of this paper.

Over the past decade or so, several remotely sensed satellite data products have been available that can help to constrain a water balance model to better represent the state of the real system. Data assimilation is a process by which new observations and their uncertainty are incorporated in a computer model. The DA technique applied to AWRA-L so far is an EnKF technique. Renzullo et al. (2013) gives a detailed conceptual description. The size of the resulting computational problem is substantial. The spatial extent of the model is 277,770 grid cells over Australia. The extent spatially and across model ensembles is illustrated in Figure 1. The model is run at a daily time step over 12 years. The size of the ensemble of models is not intrinsically fixed but in practice currently set to one hundred. Adding up computations across grid cells and model ensembles, the resulting problem size is equivalent to overall 1.21e+11 accumulated time steps, leading to substantial computational requirements. With some care taken the memory footprint can be limited to ten Gigabytes. A typical size for the outputs on disk of one simulation is ~0.4 Terabytes. The computational requirements of this problem are commensurate with those for some of the calibration of AWRA-L described in Vleeshouwer et al. (2013) and reuses some of the prior know-how from multi-threaded calibrations in Perraud et al. (2009), but the data sizes on-disk and in-memory are substantially higher for the AWRA-L DA.

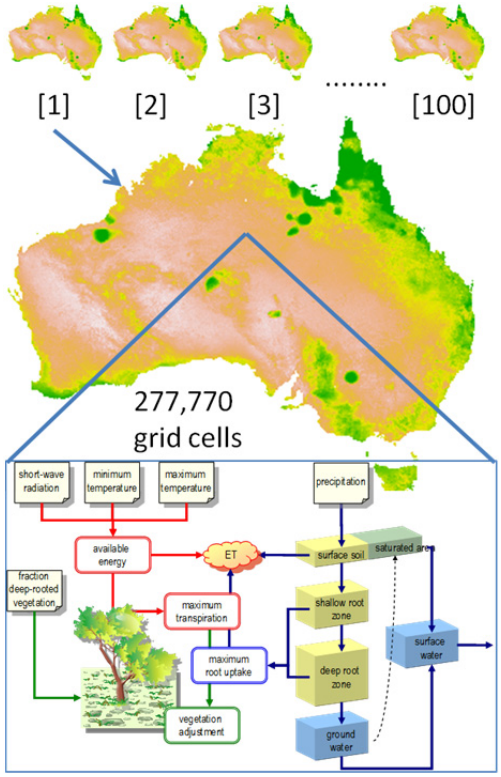

Figure 1. Problem size of AWRA-L data assimilation with EnKF using one hundred replicates.

2. USE CASES AND REQUIREMENTS

The DA system delivered has two main applications: supporting investigative research, and an operational version that is run from the Flood Early Warning System (FEWS) software described in Gijsbers et al. (2008). An overarching goal is to have most software components used in both systems and avoid where possible the drift towards parallel implementations that incur additional software maintenance and transitioning costs. As shown in Figure 2, the data assimilation engine and the AWRA-L modelling components are common to both systems, and shielded from the particular software and hardware infrastructure on which they are executed.

The research application can be done in a context where computational clusters are available for shared use. The principal researcher's original implementation is in pure R, his language of choice. While most adapted to his needs in terms of language, the runtime even on a compute cluster using about 1200 processing cores is

in the order of ten days and has file-related logistical difficulties. Given the iterative nature of exploratory research, a minimum of two weeks for a given experiment is a serious hindrance. The runtime issue cannot reasonably be alleviated with further computing power, given the shared nature of the computing infrastructure. The target runtime infrastructure for DA is the CSIRO Bragg cluster, a Linux/Microsoft Windows hybrid system supporting central processing unit (CPU) and general purpose graphical processing unit (GPU) applications. The spatially gridded nature of the AWRA-L model structure is a good candidate for execution on a GPU.

The execution of various software applications in the operational system is scheduled from Delft-FEWS. DA is one possible part of the workflow execution. Note that the modelling purpose is not flood forecasting but water accounting in this case; FEWS here is used as a general purpose workflow orchestration and data management system. A key constraint emanating from using FEWS, or at least from the way it is used in this case, is that the data assimilation is performed across the spatial dimension for the whole nation, for a given simulation time step. The target operational infrastructure is such that the software can be run on a multi-core virtual machine, possibly several multi-core machines. The infrastructure is however not a compute cluster, in particular it is deemed not possible to set up support for the Message Passing Interface (MPI), limiting options for scaling across computers while executing in accordance with the FEWS workflow.

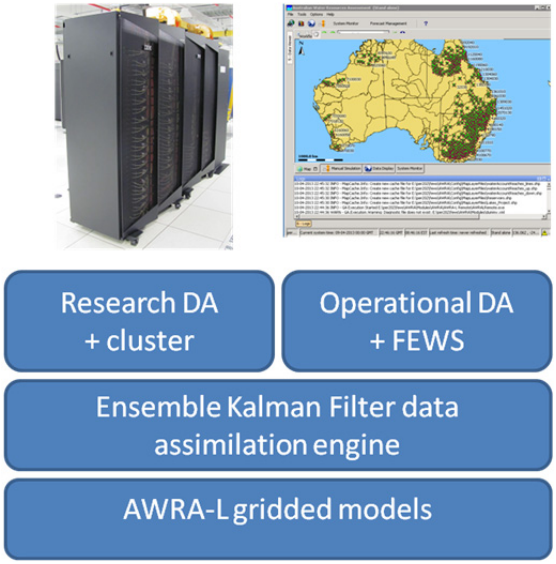

Figure 2. High-level target architecture.

Conceptually the current modelling problem formulation is such that there is no spatial dependency between cells, and it can be carved and parallelized spatially from start to finish, requiring only a final collation of partial results. This is thus highly amenable to running on a cluster, and indeed was used already with the original R implementation. AWRA-L DA itself need not include the concept of catchments, which would only make the load balancing across cluster compute nodes more difficult. The aggregation of runoff is eventually performed in subsequent steps of the overall modelling workflow.

The AWRA-L DA component is part of a larger software system. Related components are the pre-existing calibration sub-system (Vleeshouwer et al. 2013) and the single-instance AWRA-L simulation executor. Both are mostly written in the C# language. Both components pre-date the work on the data assimilation component and have substantial parts that warrant reuse. Two notable ones are the access arrangements to be run from FEWS and the input/output handling of XML and netCDF files.

Early in the AWRA-L DA feasibility investigations we intended to be compatible with and use the OpenDA toolbox (http://www.openda.org) described by Weerts et al. (2011). Due to competing resourcing across projects we could not include this toolbox in the work done so far. We nevertheless tried to design the AWRA-L DA engine so that interoperability with OpenDA is not precluded in the future.

The DA system needs to address the following quality attributes:

- Testability: While the conceptual essence of the problem can be summarized in a single linear algebra equation, the problem is multi-dimensional and the proper operation of the software depends a lot on the statistical properties of the states of the models in an ensemble. This type of problem is more difficult to capture in unit tests or debug than typical single-instance model runs. Runtime performance and resilience to infrastructure issues are very desirable attributes.
- Performance: Given the computationally intensive nature of the problem in terms of calculation as well as data volumes, the system needs to be lean. Some more readable code constructs may need to be traded off for more efficient ones.
- Scalability: The operational version of the system may need to run on a single, multi-core machine, while the research version may run on a compute cluster.
- Interoperability: Given the strong potential of a gridded model execution on GPUs, the C# code needs to be coupled to GPU computing to test the technology and have a path to increase the role of GPU computing.

- Maintainability: Further research will entail new DA experiments. It should be designed such that these experiments can be made as much as possible by modification of input and text-based configuration files, and if code changes are necessary, the most likely features in need of change should be customizable via a plug-in approach.
- Flexibility: The DA system must be able to assimilate an arbitrary number of input data streams.

3. SYSTEM DESCRIPTION

3.1. Technology mix

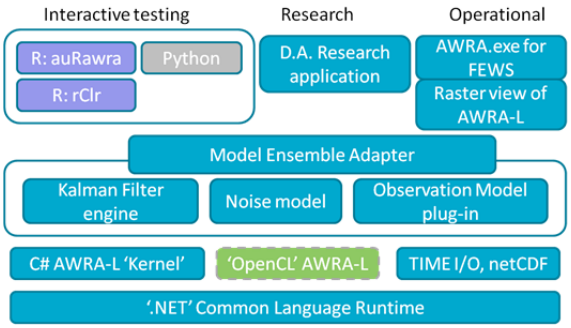

Figure 3. Overview of the architecture and main languages used for components.

Table 1 shows the main technologies and languages in use in the DA system, including a brief description of their purpose. Figure 3 is a diagram of the architecture of the solution colour-coded to convey which language is used for each component. In software projects such as the one presented here there is almost always a tension between uniformity of programming language and reusing the most appropriate functionality that may be in another language. Anecdotal evidence and surveys show a shift away from focus on a single language for programmers. While the bulk of the functionalities of the AWRA-L DA system are coded in C#, several modules are written in other languages, and interoperability is an important aspect of the system. The most basic approach of file or text exchange between processes is usually not viable given the performance requirements, not to mention the tedium. The DA system uses notably IKVM for interoperability with Java, OpenCL.NET, and a novel approach for in-process interoperability with R.

Table 1. Technology mix for AWRA-L Data Assimilation.

Language	Interoperability layer	Purpose
C#	N/A (core system)	• Compatibility with the existing program called from FEWS • know-how and development tools; consistency with the AWRA-L calibration programs • Reuse of pre-existing libraries, managed memory
Java (for FEWS)	Inter-process	Integration platform and workflow execution of DA within the whole system.
Java (for netCDF)	IKVM	Reuse the high-level facilities for spatial/temporal data sets, including NcML for convenient virtual data sets. Prior, proven experience. Managed memory.
OpenCL	OpenCL.NET	Assess the feasibility of a GPU-based gridded data assimilation.
"T4" code templates	N/A	Manage consistently AWRA-L implementation for various geometries of the input/output data
R packages	rClr, R.NET	• Comparison to the reference R implementation • Interactive testing • Retain an access arrangement from R for the principal scientist
C/C++	.NET P/Invoke	(Optional) AWRA-L coupling with a C++ groundwater model

3.2. Managing varying structures for model states

The presence of the "T4" text template technology (http://msdn.microsoft.com/en-us/library/vstudio/bb126445.aspx) deserves an explanation, as this seems a little known feature, and it plays a role in interoperability between heterogeneous components that is different from the other technologies.

AWRA-L is intrinsically a lumped time series model, and used as such in lumped catchment calibration. In the context of national scale assimilation, in theory, the spatial dimension need not necessarily take precedence over the temporal one, as a collection of lumped model structures (one per cell). In practice however, given the structure of the input data, an explicitly spatial model structure is required in most contexts. Besides, using GPU computing is another compelling reason to have a vectorized model structure, possibly but not necessarily a spatial one. Compounding this, the model may need to run on a single-precision floating-point data to reduce the memory footprint in data assimilation. Object-oriented techniques and the use of generic classes can help to prevent the need for duplicated model algorithms in situations.

However, for reasons we cannot detail practically in this paper, their usefulness proved limited for the needs we just described, and already at the cost of significant complexity.

Code generation is a possible solution in such situations. "T4" is not the only technology around for this but is readily available in Visual Studio. The all-important model algorithms are captured in a single place, verified against a reference implementation, without further duplications.

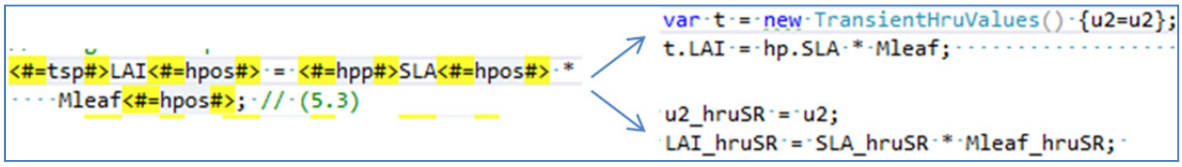

Figure 4. Code generation with T4 template.

Figure 4 gives a sample of the code generation via a T4 template. It is currently feasible to generate model structures in C# and C++, vectorized or not and with varying degrees of object orientation. One obvious application is to quickly benchmark the performance of languages and, more importantly, the effect of the data arrangement in memory. A code template approach is also a safer and more efficient way to target multiple languages or platforms for a model. Preliminary assessment of the runtime performance that given the current technological state of play GPU computing is the most likely candidate for substantial improvements on that front.

3.3. GPU computing

Parallel computation using OpenCL (http://www.khronos.org/opencl) on GPUs was very successful for the repeated computation of the model equations, and compiling OpenCL to target multicore CPUs also tested positively. This was primarily due to data independence of the model and the use of arrays to store variables, allowing efficient vectorisation of computations. The availability of *min*, *max*, and other transcendental functions such as *exp* from within kernels also eased the transition from the original C# code that made a lot of use of these types of functions. The initial C# model code was split into smaller kernels for the OpenCL.NET implementation. This increased memory requirements as temporary variables were converted to arrays for sharing between kernels, and these temporary variable arrays also increased memory transfer overheads. Despite the increased memory use and transfer time, this approach provided improved performance compared to using a single large kernel. We believe that the small kernels reduced 'register pressure', resulting in improved memory bandwidth utilization and thus increased throughput overall. OpenCL offline code analysis tools from GPU and CPU vendors were used extensively to check whether the kernels could be vectorized.

One major issue in the use of GPUs in computation of the model kernels was that of having sufficient available GPU memory for the full Australian dataset. Two complementary approaches were taken to address this. First, on each compute node of the Bragg cluster all three GPU devices were acquired, with the work shared split across those devices. Second, a further data parallel approach was used again exploiting the spatial independence of each grid cell. In this approach the spatial grid was split into slices, with one slice for each compute process. Independent compute processes could then operate on the smaller data subset, writing their results out to flattened binary arrays. A post processing step would then reassemble the binary arrays into spatial rasters. This second approach was used for both GPU and CPU computations, and was the primary method used to scale up to 8 to 10 nodes on the Bragg cluster.

Although deemed possible, the use of GPUs for the EnKF algorithms was not implemented as GPU based kernels using OpenCL. A large amount of code was involved and substantial algorithmic changes would be required to make the code vectorisable and suitable for GPU use. Development in this area is considered worthwhile for two reasons. First, it would remove the need to transfer data between the CPU and GPU code sections. Second, it would provide a speedup to the essentially independent EnKF computations.

4. SYSTEM ASSESSMENT

4.1. Performance assessment

Table 2 shows the effective, "wall clock" runtime for an experiment assimilating one data stream over 2006 to June 2012. Note that moving to assimilate five data streams increases the raw computation time by about 20%, which would be less in real conditions, so Table 2 is also indicative of runtime for assimilating up to five data streams. The comparison in Table 2 is done at feature parity: this is not a controlled benchmark

comparing exactly "like for like" in a raw computing performance sense. For instance the final implementation need not run an initial warm-up over 2000 to 2005. The table lists the summary of the computing resources to give elements for a fair comparison.

Table 2. Effective turnaround time for DA experiments.

Solution	Wall clock, national-scale	Computing resource
Pure R scripts	10-15 days	1200 computing cores
C# implementation, pilot for Murrumbidgee area (July 2012)	5-6 days	Extrapolated wall clock for 64 cores from pilot experiments on 8400 grid cells on 2 processing cores.
Final C# implementation (June 2012)	18 hours to 1 day	64 cores for simulation then 16 cores for data collation

Running multi-threaded data assimilation was identified as important in the production version, to keep a simple command line interface access for FEWS while taking advantage of multiple cores on the execution node. Figure 5 shows how the DA engine scales with multiple threads, with GPU code enabled or not. Note that the system performance is measured outside of its total execution from FEWS, where additional I/O and application overhead adds to it. Without GPU computation enabled, the system scales almost as theoretically predicted up to four cores, with a noticeable penalty starting to show for five to six cores. The single-core case with a GPU computation enabled shows a startling reduction of the runtime, most notably the step of model state inflation in the assimilation process. While the GPU enabled version scales less well and shows the start of an increased runtime beyond five cores, it still is faster than the non-GPU experiment throughout. As no GPU capabilities exist in the operational environment to which the DA system will be deployed, performance of the non-GPU implementation remains important.

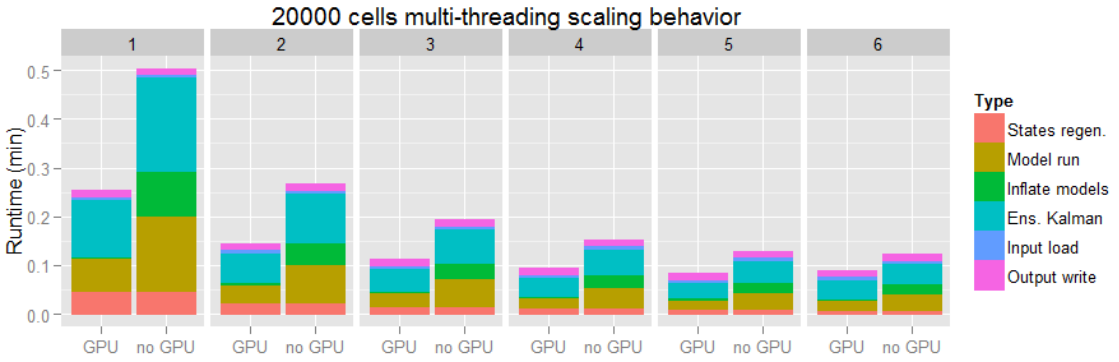

Figure 5. Multi-threading with/without GPU, 6-core Xeon 3.2GHz and NVIDIA Quadro 600.

A question we ask ourselves is how far the system performance is from a system written in a natively compiled language (C# vs. C++). This is comparing to the case of one core, 'no GPU' in Figure 5. While this is not possible to measure a full hypothetical system, there is a way to assess this with some level of confidence. The performance of the core model structure can reliably be assessed side by side using the aforementioned "T4" template generated code in C# and C++. The difference for AWRA-L stands at ~40% faster for C++ over C#, comparing cases with compiler optimizations enabled, most of the difference being provably the array bounds checks in C#. Given there is no suspected easily avoidable overheads in the full DA workflow outside of the model run, there is no reason *a priori* to suspect a much more drastic performance gain overall. Other anecdotal evidence corroborates this projection. While an appreciable speedup, this appears to be much less than what the partial use of GPU in tests suggests could be achieved with more intensive use of OpenCL code.

5. CONCLUSION

The main DA experiments testing the effect of the assimilation of up to four concurrent gridded data streams were completed in May-June 2013, using the final implementation without GPU code enabled. The final components of the system are written in several programming languages, mostly C#, R, OpenCL and Java. The mix results from the sometimes competing needs from the use cases, the prior know-how of the authors, and the proven features in the heterogeneous libraries reused.

The spatially gridded nature of this data assimilation problem formulation makes it an excellent candidate for execution on GPU, and the results from the runtime performance assessment strongly suggest to increase the role of OpenCL in the DA engine. While other substantial parts of the system dealing with testing and data handling in other programming languages will still be needed in other languages, OpenCL clearly offers the most likely substantial gains when running a model over spatially gridded data, compared to any other language, native or managed. GPU model code may not work on all platforms nor be useful in non-gridded model uses and currently is not supported in the virtualised operational environment. To avoid model code duplication and its inherent risks, solutions such as code generation exist to maintain a consistent codebase across multiple model implementations, including in different languages.

While the focus of this work was on AWRA-L, the authors tried to limit the amount of coupling to this particular model and the software can be adapted to other water balance models if needed. The design of the AWRA-L DA components is also kept conceptually similar to those in OpenDA and a joint use should be envisaged in further work.

ACKNOWLEDGEMENTS

This study is carried out in the CSIRO Water for Healthy Country National Research Flagship and is supported by the Water Information Research and Development Alliance between CSIRO and the Australian Bureau of Meteorology.

We thank the anonymous reviewers for their constructive advice.

REFERENCES

Gijsbers, P.J.A., Werner, M.G.F., and Schellekens, J. (2008). Delft FEWS: A proven infrastructure to bring data, sensors and models together: In Proceedings of the iEMSs Fourth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2008). International Environmental Modelling and Software Society, Barcelona, Catalonia, July 2008, pp. 28-36.

Stenson, M.P., Fitch, P., Vleeshouwer, J., Frost, A., Bai, Q., Lerat, J., Leighton, B., Knapp, S., Warren, G., Van Dijk, A., Bacon, D., Pena Arancibia, J., Manser, P. and Shoesmith, J (2012). Operationalising the Australian Water Resources Assessment (AWRA) system. In: Water Information Research and Development Alliance: Science Symposium Proceedings; 1-5 August 2011; Melbourne. CSIRO; 2012. 36-45.

Perraud, J.-M., Vleeshouwer, J., Stenson, M., and Bridgart, R.J. (2009). Multi-threading and performance tuning a hydrologic model: a case study. 18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009.

Renzullo, L.J., Collins D., Perraud, J.-M., Henderson, B., Jin, W., and A. Smith (2013). Improving soil water representation in the Australian Water Resources Assessment landscape model through the assimilation of remotely-sensed soil moisture products, 20th MODSIM Congress, Adelaide, Australia 1-6 December 2013.

Vleeshouwer, J., Perraud, J.-M., Collins, D., Warren, G., Gallant, S. and R.J. Bridgart (2013). Using scientific workflows to calibrate an Australian land surface model (AWRA-L), 20th MODSIM Congress, Adelaide, Australia 1-6 December 2013.

Weerts, A., van Velzen, N., Verlaan, M., Sumihar, J., Hummel, S., El Serafy, G., Dhondia, J., Gerritsen, H., Vermeer-Ooms, S., Loots, E., Markus, A. and A. Kockx (2011). OpenDA: Open Source Generic Data Assimilation Environment and its Application in Geophysical Process Models, American Geophysical Union, Fall Meeting 2011.