Computers and Electronics in Agriculture 89 (2012) 94–99

Contents lists available at SciVerse ScienceDirect

Computers and Electronics in Agriculture

journal homepage: www.elsevier.com/locate/compag

Application note

Parallelization and optimization of spatial analysis for large scale environmental model data assembly

Gang Zhao [a,b,*], Brett A. Bryan [b], Darran King [b], Xiaodong Song [b,d], Qiang Yu [a,c]

[a] Institute of Geographic Sciences and Natural Resources Research, Chinese Academy of Sciences, 11A Datun Road, Anwai, Beijing 100101, China
[b] CSIRO Ecosystem Sciences and Sustainable Agriculture Flagship, Waite Campus, Urrbrae, SA 5064, Australia
[c] Plant Functional Biology and Climate Change Cluster, School of the Environment, University of Technology Sydney, Broadway 2007 NSW, PO Box 123, Australia
[d] Institute of Urban Environment, Chinese Academy of Sciences, Xiamen 361021, China

ARTICLE INFO

Article history:
Received 6 March 2012
Received in revised form 23 June 2012
Accepted 9 August 2012
Available online xxxx

Keywords:
Climate data
High-performance computing
Zonal statistics
Array-based algorithm
GIS

ABSTRACT

Spatial–temporal modelling of environmental systems such as agriculture, forestry, and water resources requires high resolution input data. Assembling and summarizing this data in the appropriate format for model input often requires a series of spatial analyses which can be extremely time-consuming, especially when many large data sets are involved. In this paper we investigated the ability of high-performance computing techniques to improve the efficiency of spatial analysis for model data assembly. We implemented an array-based algorithm to calculate summary statistics for long time-series daily grid climate data sets for 11,575 climate–soil zones across the Australian wheat-growing regions for input into a crop simulation model. We developed a zonal statistics algorithm using Python's Numpy module then parallelized it and processed it using a shared memory, multi-processor system. We assessed algorithm performance with a varying number of CPU cores, and assessed the influence of load balancing on the efficiency of parallel processing. Compared with traditional desktop GIS software, the serial and parallel (32 cores) implementation achieved about 180 and 1440 times speed-up, respectively. We also found that the most efficient computation occurred when not all of the available CPU cores were used, and the chunk size of jobs also had an important influence on computing efficiency. The algorithm and the parallel processing scheme provides a useful approach to address computing challenges posed by spatial analysis of numerous large data sets for large scale environmental modelling.

1. Introduction

Process-based environmental models such as those that estimate the growth of agricultural crops, forest stands, or water resources typically require high spatial and temporal resolution input data (Van Wesemael et al., 2010). One of the seemingly foundational laws of environmental modelling is that this data is rarely in exactly the right format and resolution required by the model. For example, the study units are irregular agro-ecological or climate/soil zones (Devendra and Thomas, 2002; Fischer et al., 2002), while the input data are usually site data from station observation or grid data from spatial interpolation. As a result, a significant amount of pre-processing is often required to summarize and assemble data prior to running the model. This can be computationally demanding and time-consuming. Traditional Geographic Information Systems (GIS) are yet to widely embrace the shift in computer technology to multi-core processors (Bryan, 2012). Hence, the efficiency of data processing within GIS software has largely ceased to increase over the past decade following the limitations on computer processor clock speed (Dongarra et al., 2007). Although some GIS vendors have proposed some batch methods, we found that the capacity still cannot meet the requirements of our application. So we developed a customized algorithm using high-performance computing (HPC) to increase the efficiency of data pre-processing.

In this application, we focused on the assembly of high spatial and temporal resolution climate data to simulate agricultural systems. Variation in climate across both time and space significantly influence crop growth in agricultural systems (Hansen and Jones, 2000; Luo et al., 2003; Reidsma et al., 2009). Understanding how spatial–temporal variation of climate impacts agricultural systems is of critical importance for agricultural decision-making (Overpeck et al., 2011). Process-based crop models are a common way to study agricultural systems at regional or national scales and usually need a complete and accurate source of climate data across the whole landscape (Bryan et al., 2010, 2011; Safir et al., 2008; Zhao et al., in press). Interpolating station-based climate records to raster layers is a common practice to overcome the deficiencies

* Corresponding author at: CSIRO Ecosystem Sciences, Waite Campus, Urrbrae, SA 5064, Australia. Tel.: +61 08 8303 8679; fax: +61 08 8303 8582.
E-mail address: Gang.Zhao@csiro.au (G. Zhao).

0168-1699/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.compag.2012.08.007

of observational data at the regional scale (Jeffrey et al., 2001; Thornton et al., 2009). However, coupling this kind of spatial data with agricultural systems models needs a series of intensive spatial analyses, which has impeded the application of agricultural systems models to large areas at high spatial resolution, despite the fact that the computing resources are more readily available today (Finley et al., 2012).

The purpose of this study was to prepare climate data for the Agricultural Production Systems sIMulator (APSIM), a process-based agricultural systems model (Keating et al., 2003; Wang et al., 2009), to simulate wheat productivity under various management practices across Australia's wheat-growing regions. To achieve this it was necessary to extract and summarize 122 years of daily gridded climate data for 11,575 climate–soil zones. A zonal statistics algorithm, commonly found with raster GIS, was developed in Python's Numpy module to calculate summary statistics on the raster climate data layers for each zone. We parallelized this algorithm and assessed its performance under a varying number of CPU cores. We also assessed the efficiency of parallel computing with different job scheduling and load balancing approaches to identify the most efficient high-performance computing strategy. The utility of these techniques for data assembly and summary for input into environmental models is discussed.

2. Climate–soil zones and climate data

The study area forms a crescent-shaped area from the northeast coast of Queensland around the south of the continent to the west coast of Western Australia (Fig. 1). The study area includes a 100 km buffer around the actual areas sown to wheat in 2006 (ABARE, 2006; Marinoni et al., 2012). The whole study area was divided into 11,575 climate–soil zones (CS zones) with relatively homogenous climate and soil properties. Using zones as basic modelling units instead of grid cells enables us to reduce the computing load of the agricultural systems modelling yet still capture the major biophysical variation in soils and climate.

APSIM requires five key climate characteristics including minimum temperature, maximum temperature, radiation, rainfall, and evaporation. A set of spatially extrapolated raster layers of daily historical climate for Australia have been produced by the Australian Bureau of Meteorology (BOM). The pixel values are an interpolated representation of roughly 4600 station-based daily records. The processes and methods for producing this data can be found in Jeffrey et al. (2001). The data sets containing 122 years of daily historical climate data, from 1889 to 2010, were stored in NetCDF format files. Thus, a total of 222,790 (44,558 * 5) data sets needed to be manipulated and converted into APSIM input data format.

Climate data can be attributed to CS zones through various means such as assigning values from the nearest meteorological station record. However, due to the irregular shape of the zones and of the spatial arrangement of stations, we judged that the best way to assemble representative climate data for each zone was to summarize the interpolated grid data. We calculated the mean of grid cell values within CS zones as input in APSIM.

3. Algorithm and parallel processing

3.1. Design and implementation

The zonal statistics algorithm developed in this study aggregates continuous values in one raster for each zone in another raster, and computes the statistics (Fig. 2). The algorithm takes as input two raster data sets, namely zones and values, which share the same resolution and extent. Similar algorithms can be found in many raster GIS. Initially, we undertook the spatial analyses using a combination of ArcGIS tools in batch processing: *Make Net-*

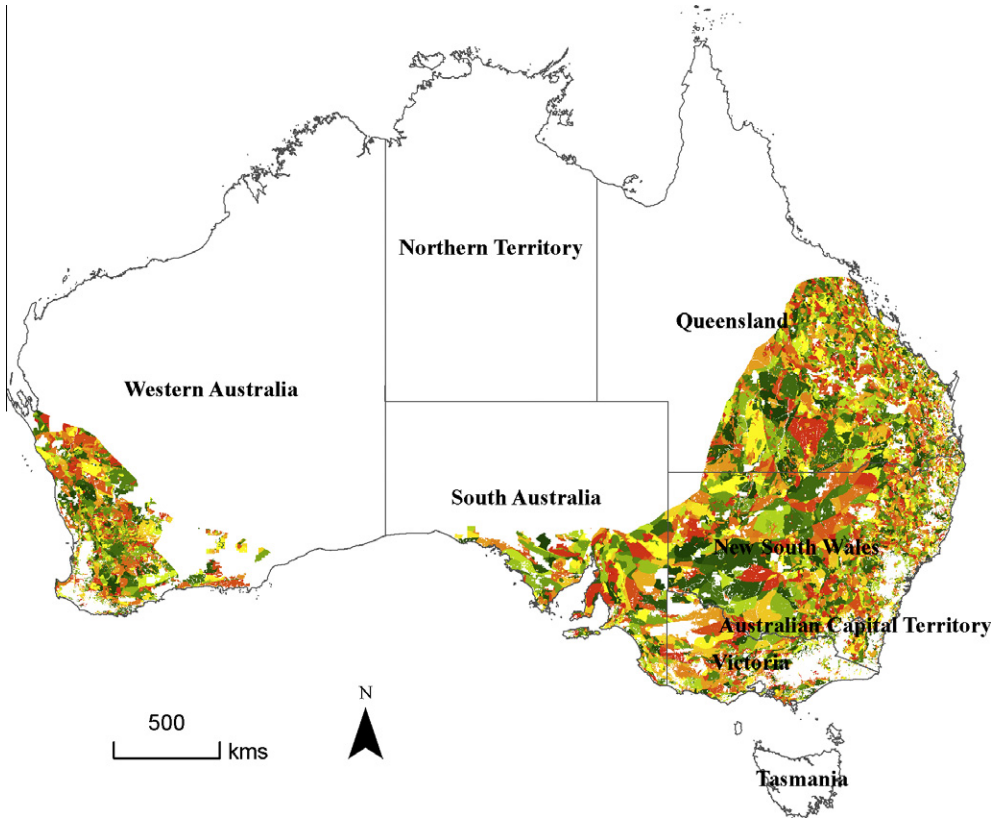

Fig. 1. Climate–soil zones. Each zone is demonstrated by one unique color.

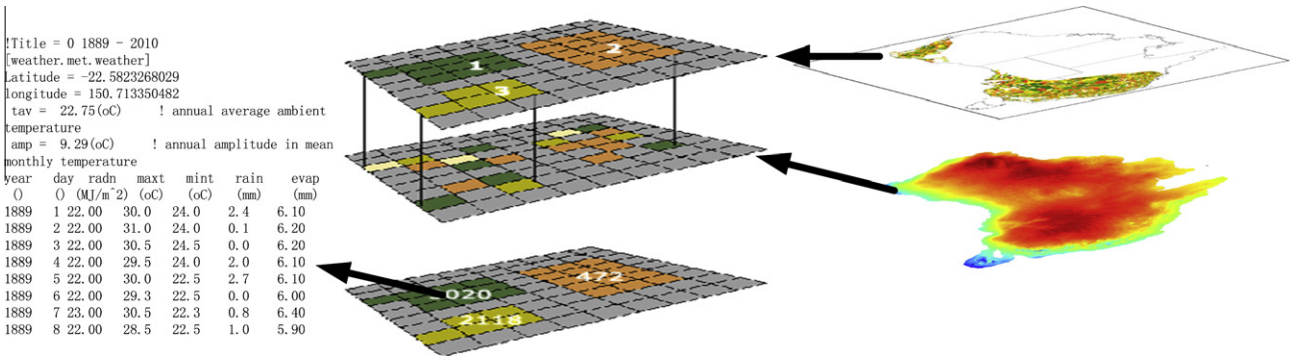

Fig. 2. Schematic of spatial statistics of the grid climate data for CS zones. Left is the format of APSIM climate data. In the middle part of the figure, the top map is the climate–soil zones, the middle map is the grid climate data, and the bottom one is the statistic results. Right part of the figure is the corresponding data of this study.

CDF Raster Layer, *Resample*, and *zonal Statistics as Table*. However, as it took about 180 s to process one data set (which, by extrapolation, would take 464 days to complete all the data sets), this method was impractical.

We first converted the polygon format CS zone data to raster format at 0.01° resolution. This resolution was chosen to maintain the smallest CS zone (1.83 km^2). We then developed a new algorithm which imports the CS zones and climate data into Numpy arrays (Jones et al., 2001), and then applies a sequence of operations including resampling the 0.05° resolution climate data to match the 0.01° resolution CS zone data, aggregating climate values belonging to each zone, and computing the statistics (Fig. 3). We implemented all the operations with the Python programming language (Van Rossum and Python Community, 2012) and the Numpy, netCDF4 (netCDF4, 2011), GDAL (GDAL, 2012), and Python Imaging Library (PIL, Lundh and Ellis, 2012) packages. NetCDF4 and GDAL were used to import climate and zones data into Numpy arrays. We resampled the climate data using the nearest neighbour method with the resize function from PIL. The aggregation operation was implemented with Numpy's *bincount* function using the zones array as input data and climate data array as weights.

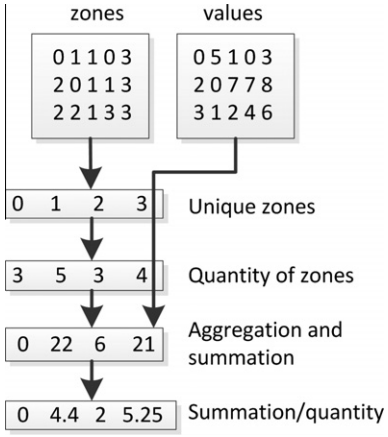

Fig. 3. Algorithm for zonal statistics.

To quantify the performance of our application we used a shared memory multi-processor server with four Intel(R) Xeon(R) X7560 @ CPUs and shared 512G Random Access Memory (RAM) and running Microsoft Windows Server 2008 R2. Each CPU had 8 cores with 2.27 GHz clock speed and 24 Mb L3 cache. All the operations were conducted in main memory.

3.2. Parallel processing

To summarise the climate grid layers more efficiently, we accelerated the processing through parallelization. Parallel processing carries out multiple operations or tasks simultaneously using

```
import pp                                          Load library
Zones_Defining_File='CSZones.img'                  Mask out no data values
ZonesArray,no_dataValue=ReadDataOfZones(Zones_Defining_File)   Prepare zones defining data
ZoneArray_Masked,Unique_Zones=MaskOutNoDataValueForZones(ZonesArray,no_dataValue)
Climate_Data_Names=ConstructClimateDataNamesList(1889)   Construct a list of climate data names
job_server = pp.Server()                           Start pp execution server with the number of workers
                                                   set to the number of processors in the system
for aw in range(1,33):
    for chunksize in [1,10,20,30]:                 Job chunk sizes
        Chunks=DividejobstoEvenchunks(Climate_Data_Names, chunksize )
        job_server.set_ncpus(aw)                   Set active workers
        Results=[]
        for chunk in Chunks:                       Submit function to the execution queue
            result=job_server.submit(ZonalStatistic,   Function to be executed in worker process
            (ZoneArray_Masked,Unique_Zones,chunk)      Arguments of the ZonalStatistic
            (ResampleArray,ReadNetCDF,MaskOutNoData,   Functions which will be called
             ReadDataofZones,AggregationAndStatistic)  from ZonalStatistic
            ("netCDF4","osgeo","numpy","scipy.ndimage","datetime","PIL",
            "itertools","scipy.misc"))             Module names to import in worker processes
            Results.append(result)                 Collect results
        StatisticsComputingPerformance(Results)    Calculate statistics of worker performance
job_server.destroy()                               Kill ppworkers and closes open files

Vary number of active workers
Divide jobs to even chunks corresponding to chunk size
```

Fig. 4. Code for embedding zonal statistic algorithm into Parallel Python.

multiple processors or cores. Processing long time-series grid data can be achieved through single-instruction, multiple-data parallelism (SIMD).

We implemented the parallelism with the Parallel Python (PP) package, which provides a mechanism for parallel execution of Python code on SMP (systems with multiple processors or cores) and clusters (computers connected via network) through process level parallelism (Vitalii, 2011). To implement this algorithm, we firstly read the zones data into a Numpy array and masked out the no-data values (Fig. 4). As all the climate data share the same zones, we assigned the zone-array in the master process. We built the parallelism in program loops and dynamically assigned the tasks to workers (processes). A job server was created and the number of workers specified. The master process distributed the zones array and climate data path to the workers. The workers executed the algorithm and computed the statistics concurrently. The master process collated summary climate information and processing statistics from the workers.

3.3. Job scheduling and load balancing

In this study, the parallel compute time was mainly consumed by data exchange and synchronization between master and worker processes and the computation of zonal statistics by workers. Parallel Python implements a dynamic job allocation mechanism that automatically allocates new jobs to idle workers or workers who have completed previous jobs. If the job allocation time is longer than job execution time, the power of active workers cannot be fully exploited. Hence, increasing the *chunk size* of parallel processing by combining many smaller jobs into fewer larger jobs may im-

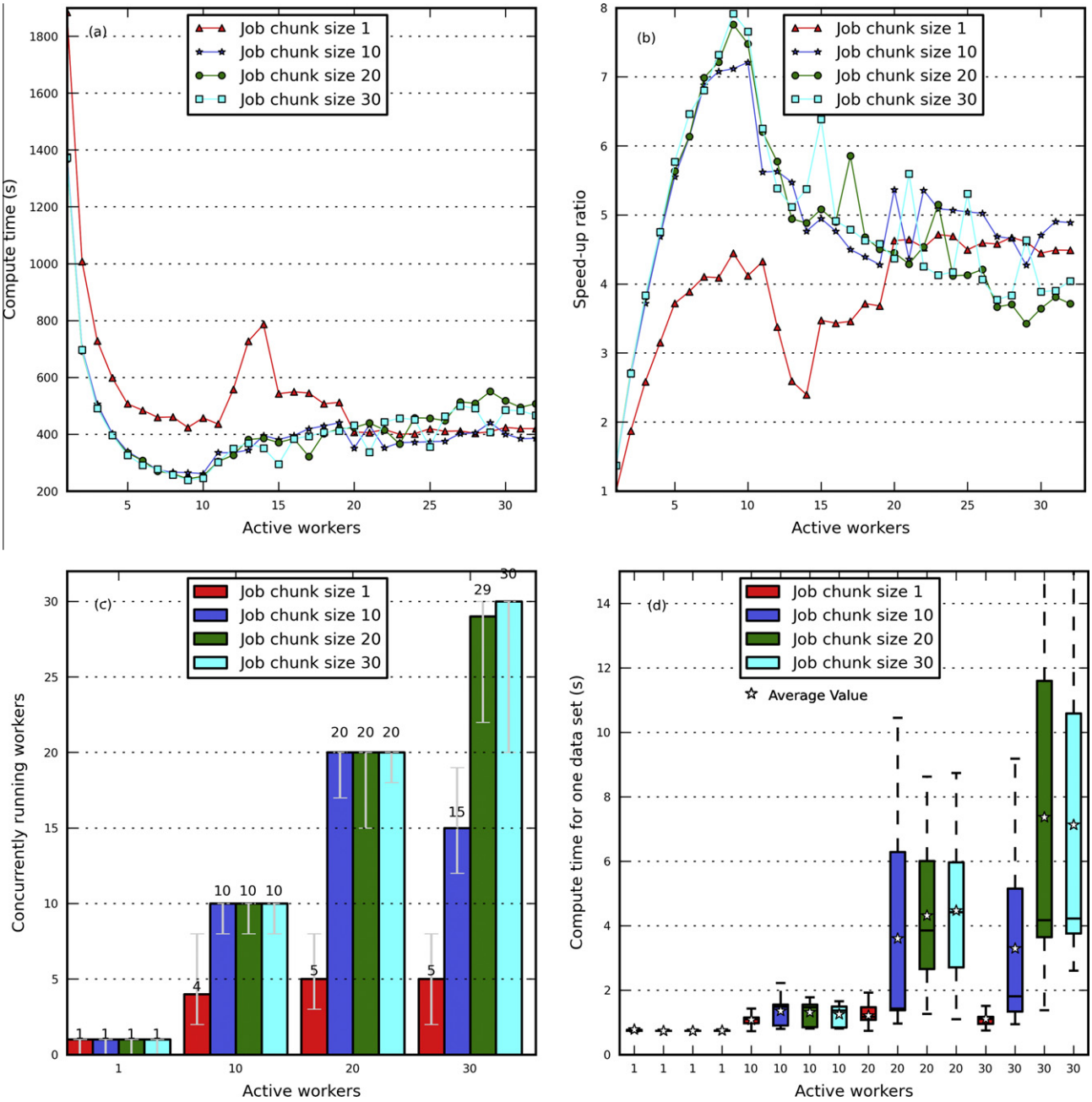

Fig. 5. Total compute time (a), speed-up ratio (b), median (with minimum and maximum) number of concurrently running workers (c), and variation in compute time per data set and (d) with different number of active workers and job chunk sizes. All graphs reflect indices from processing a single year of data (1825 climate data layers).

prove computing efficiency. To investigate the effects of job chunk size on processing performance, we set four different job chunk sizes and assessed their efficiencies with respect to compute time, speed-up ratio, and concurrently running workers (see Section 4, Results).

3.4. Evaluation of compute time and concurrency

The run time of a program is the time between the start and end of programme execution on all participating processors. The run time of a parallel program depends on many factors, such as the architecture of the execution platform, the compiler and operating system, the parallel programming environment, and the dependencies between the computations (Rauber and Rünger, 2010). A standard measure of parallel program performance is its reduction of the execution time on the specific platform (Rauber and Rünger, 2010).

To quantify the performance benefits of parallelization we processed climate data for a single year (1825 data sets) on a varying number of CPU cores (1,2,...,32). Using data on the start and end time for each individual worker process and the overall master process we calculated the variation in computing efficiency using four main metrics. We calculated the overall compute time and speed-up ratio. The speed-up ratio indicates the reduction in execution time and is calculated as $S(n) = T_S/T_n$, where T_S is the run time using one core and T_n is the run time using n cores. We also calculated the number of truly concurrent running processes under different chunk sizes polled at 0.1 s intervals over each run period and report the median, minimum, and maximum number. Lastly, we examined the variation in time taken to process a single data set with different numbers of workers and chunk sizes.

4. Results

Processing a single year of data (1825 data sets) using our zonal statistics algorithm took 1884.15 s for a single worker, or roughly one second per climate data set (Fig. 5a). The improvement was significant compared with 180 s per data set for desktop GIS processing. Run time decreased further with an increasing number of active workers up to around nine cores (228 s for 1825 jobs), beyond which performance degraded. A larger chunk size achieved greater processing efficiency, especially when less than 20 active workers were used. When more than 20 workers were active, there was little difference in processing efficiency of chunk sizes larger than a single job. The compute time increased when more than 10 active workers were engaged, no matter how big the chunk size (Fig. 5b).

The maximum concurrently running workers was eight for jobs with a chunk size of 1, no matter how many active workers were set. With chunk sizes of 20 and 30, the number of concurrently running workers almost keeps pace with the number of active workers (Fig. 5c). The compute time of processing one data set with a job chunk size of 1 did not increase when the number of active workers increased. Both the average and variance in processing time per data set increased significantly when the job chunk size was increased to 20 and 30 (Fig. 5d).

5. Discussion and conclusion

With the development of open source software and programming languages that are operating system independent and very flexible, it becomes possible to develop tailored spatial analysis algorithms for applications involving expensive computation. In this paper, we developed and implemented a Python-based serial and parallel version of a zonal statistics algorithm commonly found in many off-the-shelf GIS packages. This method is useful for model data processing and assembly especially when the study units are irregular zones. Although the algorithm was developed for zonal statistics, it can easily adapt to other applications like statistical or regression-based downscaling of GCM data, calculation of statistics for stochastic climate generators, interpolation of climate data surfaces from point-based records. In the parallel version of the algorithm, most of the data exchanges and operations were conducted in memory which decreased the compute time substantially. With the optimal configuration of nine workers and chunk size of 30 jobs, the entire climate data base could be processed within 8 h whilst desktop GIS software needs about 464 days – 1440 times faster.

Several factors affected the performance of parallel processing. Increasing chunk size improved load balancing and processing efficiency. This is consistent with the theoretical prediction that too much time consumed in data exchange and synchronization will decrease the performance. If the execution time of a single job is shorter than the time needed in scheduling and coordinating jobs, some workers will sit idle. Aggregating individual climate grid data processing jobs into large chunk sizes can improve the utilization ratio of active workers and cut down job allocation time. However, increasing the chunk size too much degrades performance because the large chunks reduce the ability to balance computational loads and the total compute time will depend on the slowest worker and time taken for the last job to complete. Overall, parallel programming in shared memory systems requires a wise choice of job chunk size and active workers to optimize the computing efficiency.

The concerns over the impact of climate variability and climate change on crop productivity have stimulated comprehensive research on the sustainability of current agricultural systems. So far, many large scale applications of agricultural models have been restricted by the availability and quality of the spatial input data. The serial and parallel version of the zonal statistics algorithm implemented in this paper exemplifies how large scale spatial computation can be dramatically accelerated by optimally utilizing open source software tools and readily available computing resources. The improved computation efficiency supports large scale, high resolution spatio-temporal modelling of environmental systems.

Acknowledgements

The authors are grateful for the support of CSIRO's Sustainable Agriculture Flagship, the National Basic Research Program of China (Grant No. 2012CB955304), WRON Data Centre, and Chinese Scholarship Council. Efforts and comments from Andrew Higgins, Mike Grundy, and two anonymous reviewers greatly improved the manuscript.

References

ABARE, 2006. Australian Crop Report. Australian Bureau of Agricultural and Resource Economics and Sciences.

Bryan, B.A., 2012. High-performance computing tools for the integrated assessment and modelling of social–ecological systems. Environmental Modelling & Software. http://dx.doi.org/10.1016/j.envsoft.2012.02.006.

Bryan, B.A., King, D., Wang, E., 2010. Biofuels agriculture: landscape-scale trade-offs between fuel, economics, carbon, energy, food, and fiber. GCB Bioenergy 2, 330–345.

Bryan, B.A., King, D., Ward, J.R., 2011. Modelling and mapping agricultural opportunity costs to guide landscape planning for natural resource management. Ecological Indicators 11, 199–208.

Devendra, C., Thomas, D., 2002. Crop–animal systems in Asia: importance of livestock and characterisation of agro-ecological zones. Agriculture System 71, 5–15.

Dongarra, J., Gannon, D., Fox, G., Kennedy, K., 2007. The impact of multicore on computational science software. CTWatch Quarterly 3, 1–10.

Finley, A., Banerjee, S., Gelfand, A., 2012. Bayesian dynamic modeling for large space-time datasets using Gaussian predictive processes. Journal of Geographical Systems 14, 29–47.
Fischer, G., Van Velthuizen, H., Shah, M., Nachtergaele, F., 2002. Global Agro-Ecological Assessment for Agriculture in the 21st Century: Methodology and Results. International Institute for Applied Systems Analysis.
GDAL, 2012. GDAL – Geospatial Data Abstraction Library. Open Source Geospatial Foundation <http://gdal.osgeo.org> (accessed 23.07.12).
Hansen, J.W., Jones, J.W., 2000. Scaling-up crop models for climate variability applications. Agriculture System 65, 43–72.
Jeffrey, S.J., Carter, J.O., Moodie, K.B., Beswick, A.R., 2001. Using spatial interpolation to construct a comprehensive archive of Australian climate data. Environmental Modelling and Software 16, 309–330.
Jones, E., Oliphant, T., Peterson, P., 2001. SciPy: Open Source Scientific Tools for Python. <http://www.scipy.org>.
Keating, B.A., Carberry, P.S., Hammer, G.L., Probert, M.E., Robertson, M.J., Holzworth, D., Huth, N.I., Hargreaves, J.N.G., Meinke, H., Hochman, Z., McLean, G., Verburg, K., Snow, V., Dimes, J.P., Silburn, M., Wang, E., Brown, S., Bristow, K.L., Asseng, S., Chapman, S., McCown, R.L., Freebairn, D.M., Smith, C.J., 2003. An overview of APSIM, a model designed for farming systems simulation. European Journal of Agronomy 18, 267–288.
Lundh, F., Ellis, M., 2012. Python Imaging Library (PIL). <http://www.pythonware.com/products/pil/> (accessed 23.07.12).
Luo, Q.Y., Williams, M.A., Bellotti, W., Bryan, B., 2003. Quantitative and visual assessments of climate change impacts on South Australian wheat production. Agriculture System 77, 173–186.
Marinoni, O., Navarro Garcia, J., Marvanek, S., Prestwidge, D., Clifford, D., Laredo, L.A., 2012. Development of a system to produce maps of agricultural profit on a continental scale: an example for Australia. Agriculture System 105, 33–45.
netCDF4, 2011. netCDF4: Version 1.0. <http://code.google.com/p/netcdf4-python/> (accessed 23.07.12).
Overpeck, J.T., Meehl, G.A., Bony, S., Easterling, D.R., 2011. Climate data challenges in the 21st century. Science 331, 700.
Rauber, T., Rünger, G., 2010. Parallel Programming: for Multicore and Cluster Systems, second ed. Springer-Verlag New York Inc..
Reidsma, P., Ewert, F., Boogaard, H., Diepen, K.v., 2009. Regional crop modelling in Europe: the impact of climatic conditions and farm characteristics on maize yields. Agriculture System 100, 51–60.
Safir, G.R., Gage, S.H., Colunga-Garcia, M., Grace, P., Rowshan, S., 2008. Simulation of corn yields in the upper great lakes region of the US using a modeling framework. Computers and Electronics in Agriculture 60, 301–305.
Thornton, P.K., Jones, P.G., Alagarswamy, G., Andresen, J., 2009. Spatial variation of crop yield response to climate change in East Africa. Global Environment Change 19, 54–65.
Van Rossum, G., Python Community, 2012. Python Programming Language: Version 2.7.3.1 <http://www.python.org/> (accessed 23.07.12).
Van Wesemael, B., Paustian, K., Meersmans, J., Goidts, E., Barancikova, G., Easter, M., 2010. Agricultural management explains historic changes in regional soil carbon stocks. Proceedings of the National Academy of Sciences. 107, 14926–14930.
Vitalii, V., 2011. Parallel Python Software. <http://www.parallelpython.com> (accessed 23.07.12).
Wang, E., Cresswell, H., Bryan, B., Glover, M., King, D., 2009. Modelling farming systems performance at catchment and regional scales to support natural resource management. NJAS – Wageningen Journal of Life Sciences 57, 101–108.
Zhao, G., Bryan, B.A., King, D., Luo, Z., Wang, E., Bende-Michl, U., Song, X., Yu, Q., in press. Large-scale, high-resolution agricultural systems modeling using a hybrid approach combining grid computing and parallel processing. Environmental Modelling & Software. http://dx.doi.org/10.1016/j.envsoft.2012.08.007.