CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2012; **24**:54–61
Published online 1 May 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1729

Accelerating climate and weather simulations through hybrid computing

Shujia Zhou[1,*,†,‡], Carlos Cruz[1], Daniel Duffy[1], Robert Tucker[2] and Mark Purcell[2]

[1]*NASA Goddard Space Flight Center Greenbelt, MD, U.S.A.*
[2]*IBM Technology Campus, DSL, Damastown, Co. Dublin, Ireland*

SUMMARY

Unconventional multi- and many-core processors (e.g. IBM® Cell B.E.™ and NVIDIA® GPU) have emerged as effective accelerators in trial climate and weather simulations. Yet these climate and weather models typically run on parallel computers with conventional processors (e.g. Intel®, AMD®, and IBM) using Message Passing Interface. To address challenges involved in efficiently and easily connecting accelerators to parallel computers, we investigated using IBM's Dynamic Application Virtualization™ (IBM DAV) software in a prototype hybrid computing system with representative climate and weather model components. The hybrid system comprises two Intel blades and two IBM QS22 Cell B.E. blades, connected with both InfiniBand® (IB) and 1-Gigabit Ethernet. The system significantly accelerates a solar radiation model component by offloading compute-intensive calculations to the Cell blades. Systematic tests show that IBM DAV can seamlessly offload compute-intensive calculations from Intel blades to Cell B.E. blades in a scalable, load-balanced manner. However, noticeable communication overhead was observed, mainly due to IP over the IB protocol. Full utilization of IB Sockets Direct Protocol and the lower latency production version of IBM DAV will reduce this overhead. Copyright © 2011 John Wiley & Sons, Ltd.

Received 2 September 2010; Accepted 30 January 2011

KEY WORDS: acceleration; climate; weather; hybrid computing

1. INTRODUCTION

Like most of high-performance computing (HPC) applications, typical climate and weather models are written in Fortran and Message Passing Interface (MPI) and run on a parallel computer system with conventional processors (e.g. Intel, AMD, IBM). With the advent of unconventional multi- and many-core processors (e.g. IBM Cell B.E. and NVIDIA GPU) enabling an order-of-magnitude performance speedup [1–4], there emerges a tremendous interest in utilizing these unconventional processors as accelerators in a parallel computer system. Since the programming paradigms in these accelerators are very closely related to their hardware and distinctly different from conventional parallel programming based on MPI, those hardware-specific programming paradigms must be encapsulated for the accelerators to be widely accepted. Therefore, a software tool that can connect the accelerators to the parallel computer system in a user-friendly, efficient, and scalable manner is highly desirable.

There are several options available to facilitate the integration of accelerators, though most have major shortcomings. Socket programming is flexible and efficient; however, it is low-level and error prone. MPI seems like a natural choice, but we found several issues with it. In particular,

*Correspondence to: Shujia Zhou, NASA Goddard Space Flight Center, Greenbelt, MD 20771, U.S.A.
†E-mail: shujia.zhou@nasa.gov
‡Employee of Northrop Grumman Corporation.

using MPI for a connection requires the existing applications to modify their MPI communicators to accommodate the accelerators [5]. For typical climate and weather models [6–8], code modifications could take significant time and effort, due to both high complexity and immense length, possibly encompassing hundreds of thousands of lines of code. Remote procedure call (RPC) is a good candidate, given its user-friendly interface. However, traditional RPC mechanisms allow applications to call functions from libraries running on remote machines, which requires the client application to use a specific API, thus forcing an application rewrite. The time and effort required is significant and, as a result, offloading functions using traditional RPC mechanisms are not often viable. Such RPC mechanisms can also create maintenance problems, since changes in the API require rewriting the client applications. In the industrial domain, these problems are a major barrier to offloading computation to systems optimized for particular types of processing, such as computer systems based on the IBM Cell B.E. processor, which is heavily optimized for numerical computing.

The Virtualizer component of IBM Dynamic Application Virtualization (IBM DAV) was developed to address this offloading problem [9, 10]. However, it has only been used for accelerating financial business applications, such as offloading Microsoft Excel calculations to IBM Cell B.E. blades.

There is a great interest in utilizing the accelerators to speedup the compute-intensive simulations such as climate and weather simulations. A typical climate and weather model consists of dynamics component solving fluid dynamics equations on a numerical grid (e.g. latitude–longitude grid) and physics components resolving the physical processes (e.g. cloud) on the subgrid level. In particular, physics components typically only have a vertical dependence (i.e. column-based) and have been shown to be easily ported to GPU [3] and Cell B.E. [4] with nearly an order-of-magnitude speedup. In this paper, we will investigate utilizing IBM DAV with a representative, well-studied climate and weather model component, solar radiation [4, 11], running on a prototype hybrid computing system composed of Intel blades and IBM Cell B.E. blades.

2. DESCRIPTION OF IBM DAV

The detailed IBM DAV architecture is shown in Figure 1. A user simply inserts IBM DAV-specific tags into a descriptor file (i.e. a header file in C-language code) and the Virtualizer can generate

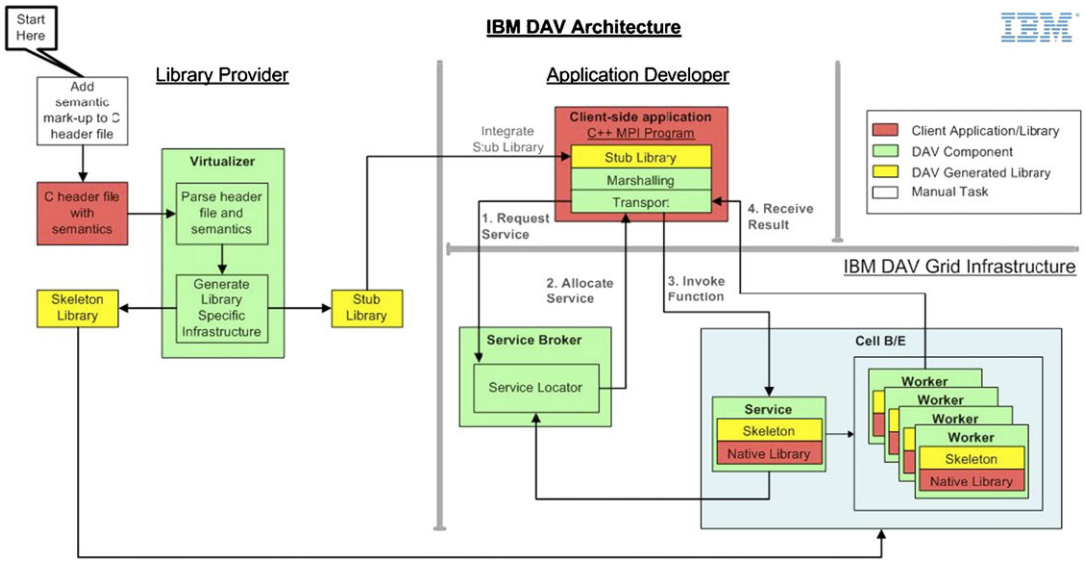

Figure 1. IBM DAV software enables C/C++, Java, and Excel applications to easily offload compute-intensive routines to remote accelerators via an IBM DAV Service Broker. IBM DAV Services run on other IBM HPC architectures, including Cell B.E., Power, BlueGene, and x86.

DOI: 10.1002/cpe

libraries that exactly mimic the interface of the local computer libraries. As a result, using IBM DAV eliminates application code changes normally required to offload functions to remote computers. The client application needs only relink to the Virtualizer-generated libraries, instead of the native code libraries. The Virtualizer currently generates C, C++, and Java stub libraries. Furthermore, IBM DAV can be extended to support other languages and applications, if required.

3. HYBRID COMPUTING PROTOTYPE SYSTEM

To evaluate the IBM DAV in a production-quality computer system such as NASA NCCS Discover [12], we have constructed a prototype hybrid computing system: There are two Intel blades: d01 has two dual-core Intel Xeon Dempsey 3.2 GHz processors and d02 has two quad-core Intel Xeon® Harpertown 2.5 GHz processors. Both of these blade types are in use on Discover. In addition, there are two IBM QS22 blades, p01 and p02. Each blade has two IBM PowerXCell™ 8i processors, containing one PPE and eight SPEs. The PPE is a dual-thread PowerPC™ Architecture processor with 3.2 GHz and 512 KB L2 cache and the SPE has 256 kB local storage for instructions and data. Both 1-Gigabit Ethernet and 4X DDR InfiniBand (16 Gbit s^{-1} bandwidth) are used to connect the Intel blades with the Cell B.E. blades.

The IBM DAV client is installed on the Intel blades, whereas the IBM DAV server is on the Cell B.E blades. Since most HPC applications use MPI for parallel communication and execution, we developed a test driver with MPI. This driver can launch a user-specified number of MPI processes in d01 and d02. Each MPI process is able to make a request to the IBM DAV broker for the service provided by the Cell B.E. After the broker receives the requests, it arranges them in a queue and dispatches them to the service provider located on the PPEs. For maximum scalability, the broker should not run on a node that also performs the compute-intensive tasks. In our prototype system, the broker is located on a PPE of p01. Since the compute-intensive tasks are carried out at SPEs, the broker's overhead is believed to be insignificant. Figure 2 illustrates the offloading with IBM DAV for one configuration involved with d02 and p01.

In our prototype system, IBM's GPFS™ file system is used. Although IBM DAV does not require a file system, it is typically used in a production computer system such as NCCS Discover. Hence, files are accessible in the same directory in both Intel and IBM Cell B.E. blades. Since there are two different computer architectures among those blades, caution must be used in compiling files and running a job. For example, an IBM DAV-designated 'client' directory should be used in the Intel blades, while an IBM DAV-designated 'server' directory should be used in IBM Cell

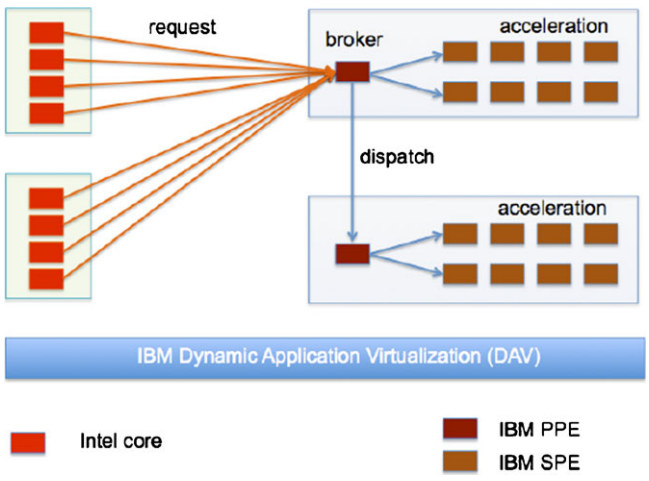

Figure 2. In the hybrid computing prototype system, compute-intensive functions can be offloaded from Intel processor cores to IBM Cell B.E. through the Service Broker of IBM Dynamics Application Virtualization software.

DOI: 10.1002/cpe

B.E. blades. In some cases, a directory specific for a particular architecture is used to prevent such a problem.

The IBM DAV Virtualizer reads the 'semantic markup' or 'tags' applied to the function prototype(s) to be virtualizer within the supplied header file. These tags provide the IBM DAV Virtualizer with information about the sizes of data to be transported to and back from the accelerator. Based on the function prototype(s) and tags, IBM DAV performs automatic code generation for client stub and server skeleton codes each time the IBM DAV Virtualizer is launched. When a compute kernel interface is stabilized, we use a directory such as 'CommonFiles' to hold the compute kernel with the interface of IBM DAV-generated client stub and its driver with the interface of IBM DAV-generated server skeleton. Establishing the underlining IBM DAV system software environment could be non-trivial. Therefore, we developed a bash script to facilitate the setup process: (1) Set up environmental variables such as the paths of IBM_DAV_PATH, IBM_DAV_SERVER_PATH, and IBM_DAV_VIRTUALIZER. (2) Launch the IBM DAV code generator to create client stub and server skeleton codes. (3) Copy the offloaded compute kernel along with its input data from a user directory such as 'CommonFiles' to the IBM DAV-designated directory, 'client'. (4) Build the .so library of the offloaded compute kernel at IBM DAV servers. Since our IBM DAV server is PowerPC architecture (Cell B.E./PPE), the offloaded computer kernel has to be built on the PPE, so ssh is used to build the .so library there. We will look into the Cell SDK cross-compiler in the Intel processor in the second stage of investigation, so that we can build the IBM DAV server on the Intel processor. (5) Launch IBM DAV deployer on IBM DAV servers. (6) Compile the codes in the 'client' directory. After running this script, a user can launch a driver in the 'client' directory to invoke the service running on the Cell B.E. blades.

To evaluate the user interface of IBM DAV, we first offloaded the solar radiation function to the PPE. After that, we maintained the same IBM DAV client stub interface and extracted the compute kernel and further dispatched them to eight SPEs through pthread, just like the standalone solar radiation function dispatches its compute kernel from PPE to SPEs. Now the PPE mainly serves for transferring the data between the client on the Intel processor and the compute kernel on the SPE accelerators. It is important to note that IBM DAV uses its automatic code generators for the client stub code. Hence, as long as the interface definition defined in the .h header file does not change (typically true, since it just defines the input and output variables), a user can select the offloaded functions from a variety of accelerators without changing its calling function. This is a very efficient and convenient way of encapsulating the special programming paradigms that are closely associated with specific accelerators. For example, a user can choose to replace the solar radiation function normally running at one x86 processor with the one running on the PPE or SPEs.

We believe that this argument is likewise valid for NVIDIA GPU, which IBM DAV also supports. In the case of NVIDIA GPU, the IBM DAV client for PPE is replaced with the one for an x86 processor which hosts the GPU. IBM DAV clients have already been demonstrated offloading to an IBM DAV service that makes CUDA calls to an NVIDIA Quadro® Plex S4 system. The IBM DAV Service calls into the CUDA library which runs the kernel on the attached GPU's. A BlackScholes option-pricing calculation was virtualized using the IBM DAV Virtualizer. The generated IBM DAV Service was running on an IBM x3455 server. This was a Quad-Core 2.3 GHz AMD Opteron™ single processor system running SLES 10.1 64-bit. The attached NVidia Quadro Plex S4 has GPU Type Quadro FX 5600 with 128 CUDA Parallel Processor Cores. The IBM DAV Service was making CUDA calls to the attached GPU's thus allowing the IBM DAV Client to utilize the remote GPU capabilities.

To deploy IBM DAV in production-quality climate and weather models, the IBM DAV-offloaded functions have to be easily incorporated into model build environments. There are two issues to be resolved. First, the top-level model drivers are written in Fortran. Second, almost all such model build environments are based on Makefiles. For example, climate models such as NASA GEOS5 and weather models such as WRF use Makefiles where the code is compiled into object files (.o) and libraries (.a), then linked together to create an executable. Hence, it is clear that the IBM DAV-offloaded functions have to be callable by a Fortran driver, thus compiled into .o or .a files, and then substituted for the corresponding .o or .a library of the original functions when the acceleration option is chosen.

DOI: 10.1002/cpe

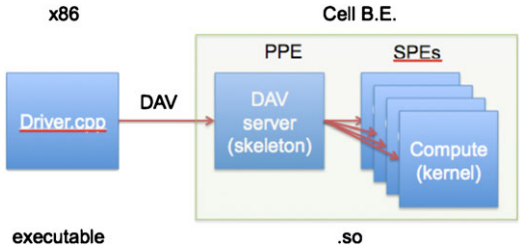

Figure 3. Schematic of IBM DAV-offloading computing from an x86 processor to SPEs through PPE.

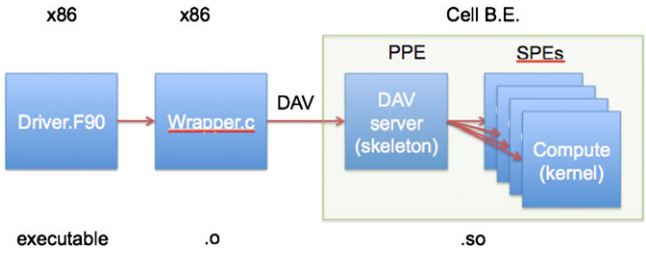

Figure 4. Schematic of IBM DAV-offloading computing to SPEs through a C wrapper function and Fortran 2005.

Currently, IBM DAV requests a C++ program to call the IBM DAV-offloaded function, which is compiled as a .so library as shown in Figure 3. To find out whether we can make this .so library callable by a Fortran driver, we developed a Fortran test driver and C wrapper code, which is derived from the IBM DAV-generated C++ test driver of this .so library. We have successfully used this C wrapper to create a .o library, and used Intel's Fortran 2005 compiler to enable a standard way for the Fortran test driver to call the IBM DAV-offloaded function in the .o library. Thus, we have shown that IBM DAV-offloaded functions can be incorporated into the model build system based on Makefiles. Figure 4 illustrates the whole code structure in this test.

It is worth noting that the IBM DAV client can be installed under a system directory or under a user's home directory. This option makes deployment of an IBM DAV client in a production computer system more convenient and less intrusive.

4. RESULTS

To measure the performance of IBM DAV, we used the C version of the solar radiation code, which has been ported to Cell B.E. previously [4]. With IBM DAV, 256 columns in the solar radiation model can be processed in a Cell B.E processor. There are 20 input variables in one- or two-dimensional arrays with the types of integer, character, and float. The total input size is 7 717 008 bits. There are eight output float variables in one- or two-dimensional arrays. The total output size is 1 245 184 bits. (IBM DAV tags the output variables as 'inout'.) In both Intel and Cell blades, gcc with –O2 is used.

We used MPI to communicate between Intel processor cores in two Intel blades. In each MPI process, a request call for the service provided by Cell B.E. is launched. The IBM DAV broker located in Cell blade p01 receives all the requests and evenly arranges them in a queue in Cell blade p01 and p02, respectively. Since each Cell blade has two Cell B.E. processors, two Cell blades have four Cell B.E. processors in total. Systematic testing has been carried out with various numbers of requests. We have observed that in cases having four requests or less, the speedup increases nearly linearly with the number of requests. For the cases having six or eight requests, we have seen that some requests take up to $\sim$66% more time to complete than that for a single request. Hence, IBM DAV speeds up the performance through a broker in a scalable and load-balance manner. This feature could be used to help optimize the overall computing power in a hybrid computer system. For example, what is the optimal ratio of conventional processors to accelerators?

DOI: 10.1002/cpe

Table I. IBM DAV Performance compared with the standalone cases with Intel Xeon core and Cell B.E. processor (microseconds).

	Total time	Communication over IPoIB network	Data rearrangement before transferring via DMA	Compute kernel invocation, and execution
One Intel Xeon core	2 044 162			
One Cell B.E. processor	65 676			
One Cell B.E. processor with IBM DAV	132 555	45 775	13 451	73 329

To quantify the overhead of IBM DAV, we took the use case with one request. The performance data is listed in Table I. We found that the total time is 132 555 ms. The total time includes (1) Transferring the input data from the Intel core to PPE with IP over IB protocol (IPoIB). (2) PPE converts two-dimensional arrays into one-dimensional arrays and arranges the data to be transferred to SPEs via Cell B.E.'s Direct Access Memory (DMA). (3) PPE creates threads and invokes the compute kernels at eight SPEs. (4) Eight SPEs fetch the input data from PPE via DMA, perform the calculations, and return the output data to PPE via DMA. (5) Transfer the output data from PPE to the Intel core with IPoIB. It took 73 329 ms in Steps 3 and 4, 13 451 ms in Step 2, and 45 775 ms in Steps 1 and 5.

Without IBM DAV, the solar radiation code takes 2.044162 s in one core of Intel Xeon Dempsey dual processors. So offloading a solar radiation function from one core of Intel Xeon Dempsey processor to one Cell B.E. with eight SPEs yields a $\sim$15.4x speedup. However, the overhead in transferring the data over IB network is $\sim$102%. In this prototype system, IBM DAV has used IPoIB for communication between the Intel and Cell B.E. blades, which is believed to contribute to most of this overhead.

To further understand the communication overhead from IPoIB, we installed OpenMPI with IPoIB on both Intel blades and Cell B.E. blades, which is the only MPI implementation currently supporting big-endian and little-endian conversion. It took $\sim$16 010 ms to send 20 one-dimensional float arrays with 12 057 elements from d01 to p01, whose total size is equivalent to 7 717 008 bits, the input data size when using IBM DAV. So the corresponding bandwidth is $\sim 482\,\mathrm{Mb\,s^{-1}}$, which is much less than the IB peak bandwidth. When we used the bandwidth of OpenMPI, it took $\sim$18 595 ms for sending the data, 8 962 192 bits, which combines both input and output data transferred by IBM DAV. So communications using OpenMPI is $\sim$2.46 times faster than using IBM DAV. One factor contributing the IBM DAV's large communication overhead compared to OpenMPI is that IBM DAV marshals parameter data to enable disparate application programming languages (i.e. Excel/Java) to interact directly with C-based routines. If this marshalling layer were removed, on the assumption of a C-to-C distributed system, the communication overhead would be reduced significantly.

IBM DAV can be extended to use the IB Socket Direct Protocol (SDP), which will further reduce the latency. Additionally, the array conversion step could also be removed, decreasing the overhead considerably.

IBM DAV has a problem using –O3 compilation flag, which will be investigated later.

5. DISCUSSION

As discussed above, IBM DAV appears to be a good candidate for connecting an accelerator to processors on a computer cluster in terms of scalability and being user-friendly. Furthermore, this approach lends itself toward an architecture that would allow computing centers to augment existing clusters with GPUs in a very cost-effective manner. However, to adopt this technology in

DOI: 10.1002/cpe

a production-quality computer system, we have to further examine other aspects such as system cost, code release, programming model, etc.

The acceleration model used in our prototype requires high-speed connections for communication between CPU processors (e.g. Intel) and accelerators (e.g. Cell B.E. and GPU). In order for this architecture to be cost effective, the connection communication time must be smaller than the time saved by offloading the calculations to the accelerators in order for this architecture to be cost effective.

In addition, the offload acceleration model duplicates the memory storage requirement for the data used in the offloaded calculations, which adds to the system cost. Hence for the offload acceleration model to be economical, two important features need to be in place. First, the offload engine needs to have low-latency and high-bandwidth access to the CPU, preferably through RDMA. Second, the offloaded job has to have a rather high ratio of computing to communication. In this scenario, it is quite possible that a formerly compute-bound job may become a communication-bound job in the offload acceleration mode.

Climate and weather models have numerous model components. Typically, a code release depends on a number of third-party system libraries (e.g. MPI) and application libraries (e.g. NetCDF and HDF5), and tunes for a few popular compilers (e.g. Intel, IBM, or PGI). It can be quite challenging to adapt the accelerated code to frequently evolving versions of climate and weather models. We believe that making the accelerated code rely less on third-party libraries, and compiling the accelerated code as a library, are two effective ways of ensuring that the accelerated code will be included in each new model release. IBM DAV appears to achieve that goal.

In our current acceleration configuration, we have two memory systems, causing an inhomogeneous programming model. An optimal configuration for acceleration is to have the accelerator sharing the same memory with the host processors. In this way, there is no need for duplicated memory storage, which simplifies the programming model and reduces the communication cost. Particularly in climate and weather models, a frequent calculation is to add the tendency from dynamics and physics components to the values at the current time step to obtain values for the next time step. In that operation, involved variables must be retrieved to perform the operation. If that operation is carried out in the host CPU processors, the involved data in the accelerator memory must be fetched to complete the calculations. That clearly degrades overall performance.

It is very interesting to see how computer processor vendors address this kind of inhomogeneous acceleration programming problem. For instance, the AMD Fusion combines general processor execution as well as 3D geometry processing and other functions of modern GPUs into a single package. This seems to be a step toward resolving the issue.

6. SUMMARY

We have investigated the software for connecting the accelerators to a parallel computer system (hybrid computing system) for climate and weather applications. In particular, we have studied IBM DAV with a solar radiation model component running in a prototype hybrid computing system consisting of both IBM Cell B.E. and Intel blades. We found that the IBM DAV is very competitive in the areas of user-friendliness and scalability. Its broker service could be used for optimizing the ratio of conventional processors and accelerators in a hybrid system. However, its current release needs to be improved in communication overhead, in particular through supporting low latency protocols such as the SDP. To adopt the accelerator in a computer system, other factors such as system costs, code release, and programming model have to be also considered.

ACKNOWLEDGEMENTS

We thank NASA High End Computing Program for support and NASA NCCS for providing prototype equipment. We also thank Scott Sinno for IBM DAV set up and MPI installation.

DOI: 10.1002/cpe

REFERENCES

1. Williams S, Shalf J, Oliker L, Kamil S, Husbands P, Yelick K. The potential of the cell processor for scientific computing. *Proceedings of the 3rd Conference on Computing Frontiers*, *CF'06*, Ischia, Italy, 3–5 May 2006.
2. Stone JE, Phillips JC, Freddolino PL, Hardy DJ, Trabuco LG, Schulten K. Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry* 2007; **28**:2618–2640.
3. Michalakes J, Vachharajani M. GPU acceleration of numerical weather prediction. *Proceedings of the Workshop on Large Scale Parallel Processing*, IPDPS, Miami, 14–18 April 2008.
4. Zhou S, Duffy D, Clune T, Suarez M, Williams S, Halem M. The impact of IBM cell technology on the programming paradigm in the context of computer systems for climate and weather models. *Concurrency and Computation*: *Practice and Exercise* 2009; **21**:2176.
5. Zhou S. Software support for hybrid computing. *eScience Workshop 2009*, Pittsburgh, PA, 15–17 October 2009.
6. The Weather Research & Forecasting Mode (WRF). Available at: http://www.wrf-model.org/index.php [31 March 2011].
7. The Community Earth System Model (CESM). Available at: http://www.cesm.ucar.edu/ [31 March 2011].
8. The Goddard Earth Observing System Model, Version 5 (GEOS-5). Available at: http://gmao.gsfc.nasa.gov/systems/geos5/ [31 March 2011].
9. IBM Dynamic Application Virtualization User Guide alphaWorks update, March 2009.
10. Purcell M, Callanan O, Gregg D. Streamlining offload computing to high performance architectures. *International Conference on Computational Science*, Baton Rouge, LA, U.S.A., 2009.
11. Chou M-D, Suarez MJ. A solar radiation parameterization (CLIR-AD-SW) for atmospheric studies. *NASA Technical Memorandum 104606*, vol. 15, 1999; 48.
12. NASA NCCS Discover computer system. Available at: http://www.nccs.nasa.gov/systems.html [31 March 2011].

FURTHER READING

1. Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
2. Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

DOI: 10.1002/cpe