PS3GRID.NET: Building a distributed supercomputer using the PlayStation 3

M. J. Harvey,[1,*] G. Giupponi,[2,†] J. Villà-Freixa,[2,‡] and G. De Fabritiis[2,§]

[1] *Information and Communications Technologies, Imperial College London, South Kensington, London, SW7 2AZ, UK*
[2] *Computational Biochemistry and Biophysics Lab (GRIB-IMIM), Universitat Pompeu Fabra, Barcelona Biomedical Research Park (PRBB), C/ Doctor Aiguader 88, 08003 Barcelona, Spain*

I. INTRODUCTION

The PlayStation 3 (PS3) games console, launched in 2006, is the latest in Sony's line of games consoles which have been distinguished by their technical capabilities and innovative design. In the case of the PS3, it is the first commodity device to contain the IBM Cell processor. The Cell is remarkable for its novel multi-core architecture that is designed to optimize the types of computation particularly common in graphical and games applications. As well as providing a boon for gamers this processor, and thus the PS3, is of particular interest for computational scientists eager to take advantage of the cheap high-performance computing power.

Although the Cell processor is over an order of magnitude faster than standard Intel or AMD processors for some scientific applications [1], it is still insufficient to satisfy the requirements of many modern computer simulations. In the field of molecular modeling in particular it is routine to use hundreds of processors in a dedicated parallel computer for a single simulation. Our goal in creating PS3GRID was to build an infrastructure that allows us to treat a collection of individual PS3 consoles as a distributed molecular simulation computational environment.

Inevitably, making efficient use of a collection of PS3 consoles is much more difficult than using dedicated high performance computing resources, especially as we wished to allow owners of PS3s to volunteer spare time on their consoles to our project. Two problems in particular required our attention:

- **reliability and trust**: because the PS3s we wish to use are outside of our control, the pool of machines available to us must be treated as transient: a volunteer may choose to add or remove their PS3 at any time. Our infrastructure must accommodate this and be able to correct for the loss of results arising from an incomplete simulation. Additionally, the results from completed simulations must be carefully checked to ensure that they are correct. Errors could arise from defective hardware or from malicious users deliberately altering the behaviour of our simulations.

- **loose coupling**: high performance computing (HPC) machines have dedicated low latency, high bandwidth communications between processors. This allows a parallel application to scale efficiently over many processing cores. The PS3, in contrast, has only a general-purpose ethernet network that is unsuited to supporting communications sensitive parallel applications. The issue is further exacerbated when the individual PS3s are distributed around the world and connected only via the general Internet.

To address the first issue, we chose to employ the Berkeley Open Infrastructure for Network Computing (BOINC) framework. It is designed explicitly for constructing distributed computing systems from volunteered resources and fit our requirements very well. It was originally developed for one of the first community distributed computing projects, Seti@HOME [2], and has been deployed by dozens of other projects since.

The second problem, that of efficiently using the resources, was solved by the combination of the development of a molecular dynamics (MD) code highly optimized for the Cell processor [1] and then by applying novel statistical techniques that allow us achieve our goals using an ensemble of short simulations [3]. This allows us to allocate work to each PS3 independently, rather than attempt to scale a single simulation across multiple PS3s in parallel.

The PS3 resources contributed to PS3GRID have allowed us to perform classical molecular dynamics simulations of biomolecular systems in far higher detail than would have been practical even with access to dedicated HPC resources, largely because of considerations of costs.

---

*e-mail: m.j.harvey@imperial.ac.uk
†e-mail: giovanni.giupponi@upf.edu
‡e-mail: jvilla@imim.es
§e-mail: gianni.defabritiis@upf.edu

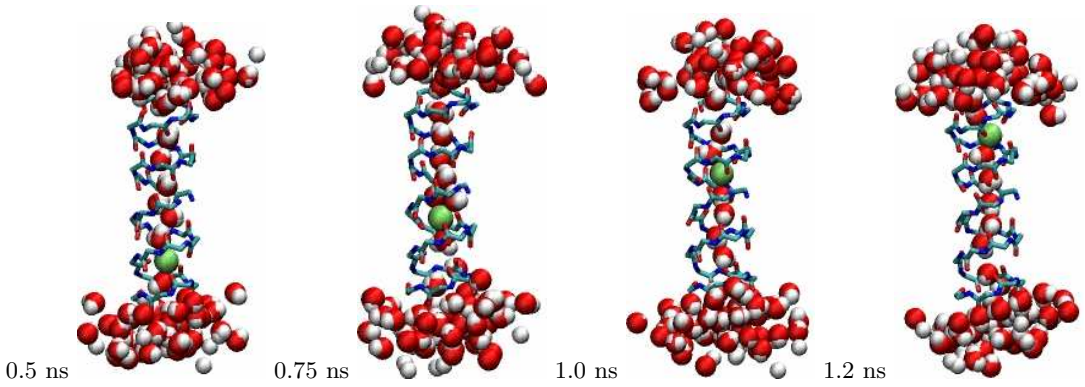


FIG. 1: Four snapshots at different simulated times of the molecular dynamics trajectory of a K+ ion pulled across the channel. The K+ ion is shown in green.

II. MOLECULAR DYNAMICS OF BIOMOLECULAR SYSTEMS

Molecular dynamics (MD) is a simulation methodology which enables the modeling of very large molecular systems at an atomic level. The atoms are treated classically, with their interactions approximated with an empirical force field. Each atom is represented using classical equations of motion (Newton's equations of motion) and evolves according to a force field which models the chemical nature of each atom (carbon, oxygen, hydrogen and so on) in its local environment. In principle, each atom interacts with all the others within a certain interaction radius, as, at long distances, the interaction between atoms is weaker. This cut-off distance is usually between 10 to 12 Å (1 Angstrom is 1E-10 meters). Each step of a molecular dynamics simulation usually corresponds to just 1 fs (10E-15 seconds). Both factors contribute to the large computational cost of molecular dynamics simulations: bridging molecular, atomistic scales with biological scales (micro- to milli-seconds) is therefore a big challenge in computational biology. These characteristics of MD algorithms limit simulations to modeling at most a few nanoseconds of the evolution of the system, even when run on HPC systems; Many orders of magnitude less than is required to model biological processes, which commonly occur over timescales of micro- or milliseconds. If we manage to overcome this limitation, then the biomedical applications of molecular dynamics would be much broader.

The process of ion traversal of membrane channels is particularly important for cell regulation. For the purpose of testing PS3GRID, we studied this process using a simple model of a single Gramicidin-A pore in a biological cell [4–6]. Gramicidin A is a polypeptide molecule with antibiotic properties. It acts on the cell wall of a bacterium, creating a trans-membrane pore that is selectively permeable to ions (Figure 1). This disturbs the concentration of ions within the cell, leading to cell death. Although extensively investigated, previous computational studies have failed to recover the energetic properties of the pore channel [7].

Recently developed statistical techniques [8–10] allow us to take a novel computational protocol which, although not requiring any fewer computing resources than previous methods, does allow us to replace a long simulation with an ensemble of much shorter ones. In each simulation within the ensemble, the ion is forced through the protein channel at a much greater rate, sufficient to drive the system away from its equilibrium system. By subjecting these non-equilibrium simulations to statistical analysis [11], it is possible for us to recover an estimate of the equilibrium free energy profile (a fundamental thermodynamic quantity of a physical system [3]) by computing averages over repeated independent runs.

The ability to recover the free energy profile from an ensemble of simulations has important practical implications. We are now able to use multiple compute resources simultaneously. Unfortunately, our system is still sufficiently large that the short simulations cannot be practically performed on even high-end commodity workstations, requiring still 24-32 processors on dedicated HPC resources for each run.

The PS3, with its Cell processor, and the PS3GRID infrastructure allow us to bridge this performance gap and run the ensemble simulations directly on undedicated, commodity systems.

III. CELL PROCESSOR ARCHITECTURE

The present version of the Cell processor comprises one general purpose PowerPC processing element (PPE) which runs the operating system and acts as a standard processor and 8 independent, specialized, synergistic processing

elements (SPEs). Main memory can be accessed only by the PPE core: each SPE must use its limited in-chip local memory (local store) of 256 KB. This memory is accessed directly without any intermediate caching. Each core (PPE or SPE) features a single instruction multiple data (SIMD) vector unit. The SPEs can, in total, perform about 230 GFLOPS at 3.2 GHz for single precision floating-point operations.

Currently, the SPEs perform floating-point operations an order of magnitude slower in double precision than in single precision. It is expected that the next revision of the Cell processor will contain better support for double precision operations. The main elements of a SPE are a data processing core, also called the synergistic processing unit (SPU), and a memory flow controller (MFC) which handles communications between main memory and the local memory of the SPE (local store). The SPU can only access the local store using a high bandwidth, low latency link without intermediate caching which can load data into the registers of the SPU in just few clock cycles. A direct memory access (DMA) operation managed by the MFC allows to copy data from main memory to the local store. The DMA is initiated by the SPE asynchronously allowing for overlapping communication (MFC) and computation (SPU), therefore partially hiding the time of data transfer into the local store.

A. Why Cell?

Historically, gains in processor performance have been achieved primarily by rising clock speeds which has been accomplished by ever finer fabrication processes. In recent years, it has become increasingly difficult to continue to increase clock speeds because of limits in process technology and the increasing power demand of faster processing cores. Despite this, Moore's Law, the empirical observation that the density of transistors on an integrated circuit doubles every 18 months has continued to hold true.

In order to continue to improve processor performance, manufacturers have been forced reconsider their 'single fast core' design and take advantage of the greater transistor counts to build CPUs containing multiple independent processing cores.

Although the aggregate performance of multi-core CPUs has continued to increase, because the cores are independent it is no longer possible for serial, single-threaded programs to take advantage of the increased processing capability. Instead, it is necessary for codes to be parallelised: adapted to allow the computation to be performed concurrently on multiple cores.

As well as the increased difficulty in making efficient use of the cores within the processor, multi-core CPUs are further limited by memory bandwidth. Minimising the cost of accessing main memory has been a long-term challenge for processor designers and has two problems that must be considered and that are exacerbated by multi-core architectures:

- **latency**: For modern processors, it will typically take hundreds of cycles for main memory to respond to a memory access request. It is now routine for processors to contain on-die cache memories into which main memory contents are speculatively read and written data is are stored before being flushed back to main memory asynchronously.
- **bandwidth**: The efficiency with which a processor can operate on large blocks of data is often limited by the rate at which data can be transferred across the link to main memory. If this connection has insufficient capacity, the processor can become starved and the full cost of accessing memory will be incurred. As the core count increases, the fraction of memory bandwidth available per-core diminishes, increasing the likelihood of starvation.

The Cell processor is the first general-purpose processor to implement a multi-core architecture that has features specifically designed to mitigate the effects of this 'memory wall'. This design allows the Cell to overlap computation with memory access and enables carefully designed applications to hide the cost of the latter.

B. Optimising for Cell

All this computational power comes at the cost of ease of use: to make best use of the processor codes must be carefully multi-threaded and vectorized. The Cell processor can be programmed as a multi-core chip with nine heterogeneous cores using standard ANSI C and relying on the libraries from the IBM system development kit (SDK) [12] to handle communication, synchronization and SIMD computation. The programming paradigm is an important aspect which distinguishes the Cell processor from other specialized processors, for example graphical processing units (GPUs). In fact, recent products like NVidia's Complete Unified Device Architecture (CUDA) SDK [13] reduce the difficulty of programming GPU devices by a non-standard C-like programming language. On the contrary, the Cell

processor adopts a common C approach using a set of advanced but standard programming techniques and languages like C/C++, already in use on standard multi-processor machines, supported by a Cell processor specific system library.

The overall performance is strongly dependent on the the effective use of Cell hardware which is largely left to the code and compiler. However, each step in the optimization can be taken incrementally. An existing application would run on the Cell processor by a simple re-compilation of the code using only the PPE core, with no effort, but also without advantages from a performance viewpoint. In order to obtain the highest performance, it is necessary to use all the SPEs, vector hardware and to adapt to the memory access architecture.

Vectorization of the code is very important because the SPEs are not optimized to run scalar code and handling unaligned data. A SIMD add instruction (spu_add) allows the computation of four simultaneous floating-point add operations by operating on a 128 bit data type (a vector float). These intrinsic primitives are for the most part derived from the more standard AltiVec instruction calls in the PowerPC element (eg vec_add). The compiler automatically aligns vector types to 16 byte memory boundaries which can then be loaded directly into the SPE registers. Manual data alignment and padding are also necessary for data communications between local stores and main memory.

After vectorization of the compute-intensive parts of the code, the work must be distributed on multiple SPEs using multi-threaded programming techniques that entail handling synchronization between processing threads running on the 9 processing cores of the Cell processor. The libraries of the SDK provide several ways to control SPE threads which in most cases are similar to other libraries providing threading primitives. It is also best to avoid conditional branching in the computational intensive parts of the code because SPEs lack appropriate hardware for branch prediction.

Optimizations discussed so far would be beneficial to standard processors as well (for instance using the streaming SIMD extensions (SSE) of Intel processors). Unique to the Cell processor is the SPE core design which makes all these optimization steps crucial for performance and the local store which provides very fast access to local data. The SPE core design provides reduced power consumption and higher clock frequencies, while the memory architecture is designed to mitigate the high latency incurred in access main memory. This new memory architecture requires the programmer to consider the limited size of the local store of each SPE and carefully plan the communication between local store and main memory to maximize the overlap of computation and communication. Overall, good knowledge of standard parallel and vector programming techniques represents the largest learning obstacle to program the Cell processor, as well as standard multi-core chips.

C. CellMD

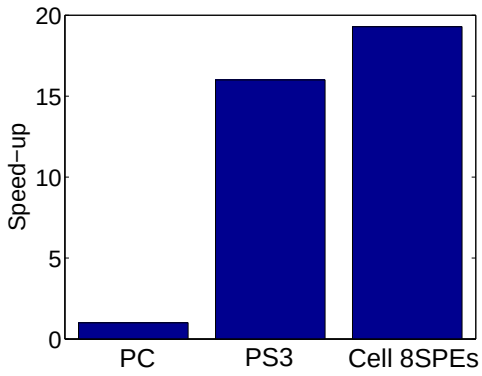
The molecular dynamics engine of this project is based on CellMD, an MD application optimized to run on the Cell processor [1, 14]. In summary, comparing the standard processor version of the code compiled on a 2GHz Opteron based PC with the Cell-tailored version of the software running on an IBM Cell blade, a speedup relative to the Opteron reference system of approximatevely 19 times is obtained reliably for many different molecular system sizes, even with just 2,500 atoms. A description and benchmarks of this code on the Cell processor are reported in [1]. For PS3GRID, we use a reduced version adapted to be used within the BOINC infrastructure. Figure 2 shows benchmark results obtained running CellMD a simulation of a 30,000 atom Gramicidin A model on a PS3. The top graph shows the average execution time for 50 time-steps whilst the lower gives the speed-up factor obtained running CellMD on a 2GHz Opteron PC and on a PS3 using $1, 2, 4$ and 6 SPEs.

We note that the fastest execution time (the maximum speed-up) is obtained when running the simulation on our development system, an IBM Cell blade server, which allows for the use of all the 8 SPE present in a Cell processor [15]. However, the PS3 seems to be faster on the same number of SPEs. Unlike the blade server, the Cell processor in the PS3 has only 7 active SPEs. Furthermore, only 6 are available when running Linux on the PS3.

Despite this, the speed up factor using only 6 SPEs is greater than 16 times, still a very good performance/cost ratio. This speedup easily balances the effort needed to put together a distributed computing environment which can enable us to perform molecular simulations in a distributed environment. Also, we consider that this speed-up is critical to the successful deployment of a grid of PS3 committed to biomolecular simulations due to the intrinsic cost of molecular simulations work units and the unreliability of the resources in network distributed computing settings.

Although the PS3's 256MB of main memory is small by modern workstation or server standards, it is perfectly adequate for our application: a molecular dynamics simulation of a 30,000 atom system requires less than 10MB of dynamic storage. Whilst the PS3's memory is sufficient to allow us to model systems with approximately half million atoms, in practice such a large simulation would be impractically slow.

Double-precision arithmetic is often considered a pre-requisite for serious scientific and numerical computing as it minimises the accumulation of round-off error and allows a much greater range in comparison to single-precision math. However, for molecular simulations this is not the case and single precision furnishes a valid alternative to double



	PC	1 SPE	2 SPEs	4 SPEs	6 SPEs
Time (seconds)	73.3	20.4	10.9	6.07	4.47
Speed-up relative to Opteron 2Ghz	1	3.6	6.7	12.1	16.4
Speed-up relative to 1 SPE		1	1.9	3.4	4.6

FIG. 2: Performance of CellMD software run on PS3. Execution time (top) and speed up factor (bottom) running Gramicidin-A for 50 time-steps on a 2GHz Opteron PC and $1, 2, 4$ and 6 SPEs (estimates are computed over longer runs and rescaled to 50 iterations).

precision [1]. In the general case, it is often possible to conduct the majority of a computation in single-precision, reserving double precision for ill-conditioned and critical sections. The techniques used to achieve this are described in further detail in [16].

IV. PS3GRID

We choose the Berkeley Open Infrastructure for Network Computing [17] (BOINC) as the middleware to enable willing PS3 users to easily share the computational burden of our scientific application. The BOINC software platform provides end-to-end distributed computing infrastructure that provides generic user authentication, file transfer, client-side and work-flow management functions. Its modular structure permits it to be easily customized for the requirements of any given project.

A sophisticated client-side tool provides a wrapper for the project application (the scientific payload code). This approach minimizes the work required to adapt a code to operate as a BOINC-aware application. However, the collaboration of up to tens of thousands of not-trusted contributors using a public network poses serious challenges to the availability, stability and reliability of such infrastructure. In order to mitigate these problems, BOINC middleware provides solutions such as digital-signing of binaries, redundant calculation and advanced work scheduling. In addition, the BOINC code is open and can be customized for specific projects needs. The wide BOINC user base and the increasing number of computational projects (and related scientific results) that use it demonstrate the maturity and efficiency of such middleware.

As mentioned above, the PS3 game console has attracted considerable interest in the high performance computing community, sparking a variety of projects that rely on its innovative architecture and peak performance. In particular, the implementation of our research project benefits from the fact that Terrasoft has recently adapted its Yellow Dog Linux (YDL) [18] distribution to work on a PS3. We therefore can install Linux on a PS3 game console, which can consequently be considered as a fully functional and equipped personal computer. More importantly, Linux acts as glue between BOINC (client) middleware and Cell hardware, and this allows us to follow the standard procedure to set up a desktop distributed computing project using BOINC software, which is the configuration of data and scheduling servers in combination with the arrangement of a suitable set of executable and data files to be downloaded using the BOINC client by volunteering users (see [17] for details).

On the client side, each PS3 performing processing for a BOINC project runs a BOINC client which provides a harness in which the application code is run. The client conducts all communications with the project servers and performs work-unit file staging and controls when the application code is running according to a policy set

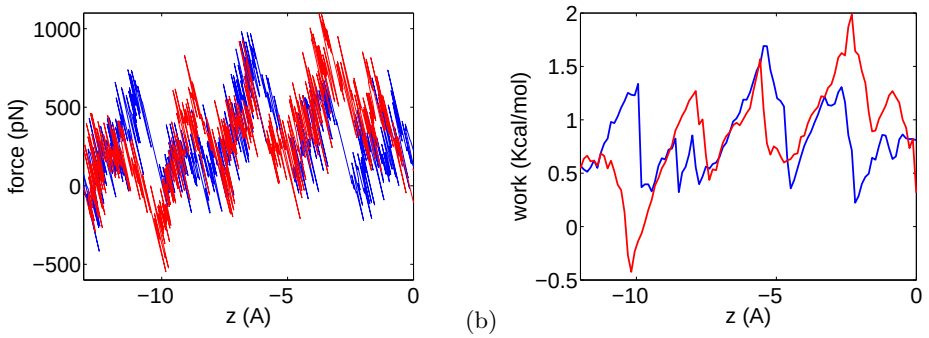


FIG. 3: (a) An example of two realizations of the forces experienced by K+ pulled trough Gramicidin A. (b) Average local work W reconstructed from the force over 100 binned intervals.

by the system's operator. A C API and library provides a set of routines that the application code can use for communicating with the BOINC client [17]. Each computational task farmed out to client Cell processors via the BOINC infrastructure is an independent entity known as a 'work unit'. These are specified by the set of input files and by the names of the resultant output files that the application code is expected to produce. The only difference between our project and other well known projects that use BOINC (such as SETI@Home [2]) is that the underlying computing resources are PS3 game consoles instead of personal computers. As a consequence, what we report here can be replicated with minimal effort once other efficient Cell-tailored software will become available.

A. Computational protocol

To recover the ion-channel permeability the Gramicidin-A pore, we first construct an atomistic model of the protein embedded within a lipid bilayer representative of a cell membrane. The system is hydrated to simulate realistic conditions in the cell. A potassium ion $K+$ is placed within the trans-membrane pore formed by the Gramicidin-A protein.

We then apply an artificial pulling force to the ion, forcing it back and forth along the channel. This forward-reverse steered molecular dynamics protocol is based on the Crooks' formula [11]. Crooks formula allows to reconstruct free energy differences $\Delta A_{0\to 1}$ between two states from a set of non-equilibrium molecular dynamics simulations which connect the two equilibrium end-states. This technique is known as steered molecular dynamics (SMD).

The total free energy profile of crossing is computed using the protocol described by [3]. For each pulling experiment, the realization of the work produced by the pulling force over the reaction coordinate is returned. This work is computed from the instantaneous force acting on the pulled ion at each ion position (Figure 3a). The ion transfers the molecular forces to the pulling spring (Figure 3b) acting as a probe for the local molecular potential energy surface which shows several binding sites with steep forces along z. Binning the data over 100 intervals we compute the average work at each bin position. Different runs produce different non-equilibrium realization of the work (Figure 3b) which are then averaged to obtain the free energy.

The PS3GRID distributed computing environment particularly suits this method as many pulling simulations can be broken into smaller work-units and off-loaded and run simultaneously on PS3s of subscribed users. We stress here the key fact of using CellMD on a PS3, as one single pulling experiment simulation lasts more than 14 days on a 2GHz Opteron PC, but just 22 hours on the PS3. This time frame would make our application impossible on a PC grid considering the extremely volatile on-grid-persistence of the average user.

B. Science&Society

In a volunteer based project the participation of people is crucial because the capabilities of the system are directly proportional to the number of users it attracts. All BOINC projects therefore require an investment in terms of public dissemination and science&society interactions.

The public interface of the PS3GRID project is the ps3grid portal `http://www.ps3grid.net/PS3GRID` (Fig. 4) based on the server framework provided by BOINC. It contains information on how to join the project, the goal of the project and provide public forums for communication between users and scientists. We have been using this website

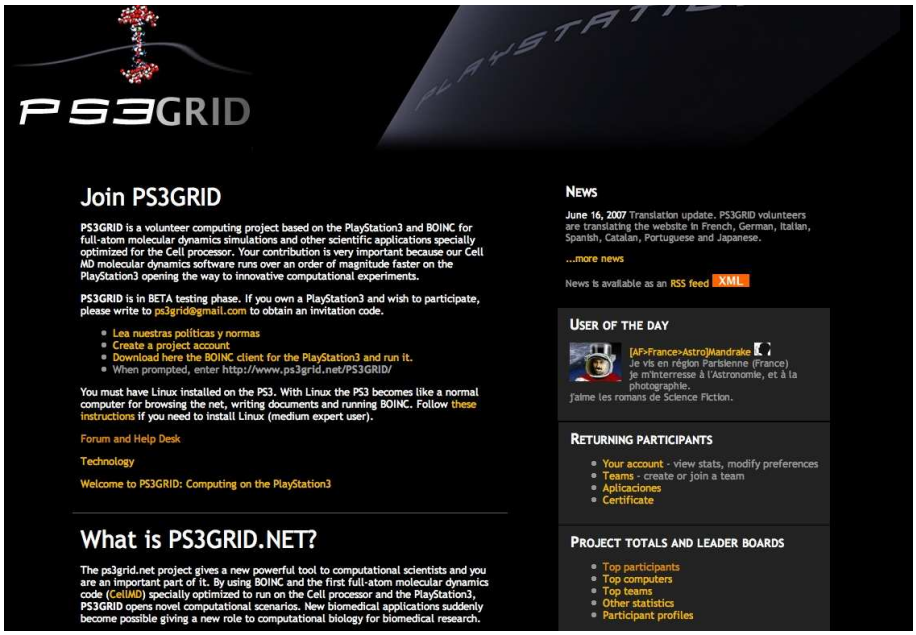


FIG. 4: A view of the public PS3GRID.net website. The PS3 BOINC client can be download ed directly from PS3GRID.net

as the only form of advertisement for PS3GRID to date, but previous interest was generated by the CellMD code alone which was featured in the media [19] due to the novelty of using the PlayStation 3 for molecular simulations demonstrated in November 2006. As a result, the website is, at the time of writing, visited by over 300 new visitors per day and obtain a number of hits of the order of millions in a google search due to the novelty of the use of game consoles (largely due to 'boincstats' websites which aggregate user statistics over BOINC projects).

During the beta phase of the project, we have restricted account creation using an invitation code in order to limit the participants to people really interested in the project. Invitation codes are obtained by a simple request to a email address posted on the website. After only a few months since the beginning of the project we have over 450 registered users of whom over 35 are actively donating cycles from their PlayStation3. These numbers are short lived and growing but, of course, are not going to be comparable to other BOINC projects because we run exclusively on PS3s and require the user to install Linux on the machine. In the near future, PS3GRID should also be available for the native PS3 operating system if Sony will release the BOINC client for the PS3 native operating system currently under development by Sony engineers. This will give all owners of PS3 console the ability to participate in PS3GRID without the need to install a new operating system.

In the beta testing phase, this user pool was able to generate a computational power of 300 personal computers, a sustained floating-point performance of 400 GFLOPS (for comparison, a single workstation can manage only approximately 1 GFLOPS when running a comparable code), 5 GB of data, 100 ns of molecular dynamics trajectories and over 6 years of computation by a single PC. All this in a time window of approximately a month!

Conclusions

We have described a computational infrastructure called PS3GRID based on a BOINC distributed computing server for the PlayStation3 and the CellMD molecular dynamics optimized for full-atom molecular dynamics simulations on the Cell processor. The motivation behind this projects lay in the facts that CellMD performs over one order of magnitude faster on the Cell processor compared to an Opteron processor at 2GHz [1] and that the Sony Playstation3 game console has a very large user base (3 million consoles sold up to date) all of whom are potentially able to contribute to this project. We use this computational environment to compute K^+ ion permeability for Gramicidin A, a first test application. Although simpler than more important ion channels, it remains interesting because computational methods have so far failed to reproduce the experimental free energy barrier. Our benchmarks show that even with a user base of just several hundred volunteers (a conservative estimate given the number of users of other mature BOINC projects) the computational throughput would allow us to complete numerical molecular experiments on a daily basis.

PS3GRID.NET represents the first attempt to use BOINC and a full-atom simulation code (CellMD) for the Cell

processor [1] to build a distributed molecular simulation computational environment based on game consoles, while the use of special processors like graphical processing units (GPUs) and the Cell processor for folding smaller proteins in implicit solvent has also been recently announced by the folding@home project [20]. Indeed, the potential of the Cell processors for scientific applications [21] and the scope of PS3GRID are much wider than just this first application as it could serve as a computational engine for free energy calculations in ion channels and other proteins. In fact, CellMD and BOINC can compete with expensive multiprocessor high performance computers (HPC) in this application case opening the possibility of High Performance Network Computing (HPNC). However, other molecular applications of large molecular structures or different computational protocols would require supercomputers made of many Cell processors like the new IBM Roadrunner [22] featuring 16,000 Cell processors.

[1] G. De Fabritiis, Comp. Phys. Comm. **176**, 660 (2007).
[2] Search for ExtraTerrestrial Intelligence at Home `http://www.setiathome.berkeley.edu`.
[3] G. De Fabritiis, P. V. Coveney, and J. Villá-Freixa, submitted to Proteins.
[4] J. Aqvist and A. Warshel, Biophys J **56**, 171 (1989).
[5] B. Roux and M. Karplus, Annu Rev Biophys Biomol Struct **23**, 731 (1994).
[6] T. W. Allen, O. S. Andersen, and B. Roux, Biophys. J. **90**, 3447 (2006).
[7] B. L. de Groot, D. P. Tieleman, P. Pohl, and H. Grubmller, Biophys J **82**, 2934 (2002).
[8] M. Jensen, S. Park, E. Tajkhorshid, and K. Schulten, PNAS **99**, 6731 (2002).
[9] D. Collin, F. Ritort, C. Jarzynski, S. B. Smith, I. Tinoco, and C. Bustamante, Nature **437**, 231 (2005).
[10] I. Kosztin, B. Barz, and L. Janosi, J. Phys. Chem. **124**, 064106 (2006).
[11] G. Crooks, Phys. Rev. E **60**, 2721 (1999).
[12] IBM Cell Broadband Engine Software Deveopment Kit (SDK), `http://www.ibm.com/developerworks/power/cell/`.
[13] Complete Unified Device Architecture, NVidia, `http://developer.nvidia.com/object/cuda.html`.
[14] Acellera Limited `http://www.acellera.com`.
[15] IBM Cell Broadband Architecture tutorial `http://www.research.ibm.com/cell/`.
[16] J. Dongarra, International Conference on Parallel Computing (ICPP06) (2006), `doi://10.1109/ICPP.2006.68`.
[17] Berkeley Open Infrastructure for Network Computing `http://www.boinc.berkely.edu`.
[18] Yellow Dog Linux for PS3, `http://www.terrasoftsolutions.com/`.
[19] Press impact of CellMD includes: El Mundo, Avui, Innovation, TV3, La Sexta, RAI, ComRadio, Radio Cadena Ser, El Punt, Diario Medico, etc. A list of links is available at `http://www.ps3grid.net/PS3GRID/old_news.php`.
[20] Folding@home `http://folding.stanford.edu`.
[21] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, *The potential of Cell processor for scientific computing* (CF06 May 3-5, 2006, Ischia, Italy, 2006).
[22] IBM Roadrunner supercomputer, `http://www.lanl.gov/orgs/hpc/roadrunner/index.shtml`.