The Canadian Internetworked Scientific Supercomputer

Christopher Pinchak, Paul Lu, Jonathan Schaeffer, and Mark Goldenberg [a]
`http://www.cs.ualberta.ca/˜ciss`

[a]Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2E8
{pinchak,paullu,jonathan,goldenbe}@cs.ualberta.ca

On November 4, 2002, a Canada-wide virtual supercomputer gave a chemistry research team the opportunity to do several years worth of computing in a single day. This experiment, called CISS-1 (Canadian Internetworked Scientific Supercomputer), had three research impacts: (1) in partnership with C3.ca, created a new precedent for cooperation among Canadian high-performance computing sites, (2) demonstrated the scalability and capabilities of the Trellis system for wide-area high-performance metacomputing, and (3) produced a new computational chemistry result.

Using roughly 1,376 dedicated processors, at 20 facilities, and in 18 administrative domains across the country, approximately 3.5 CPU years of computing were completed. CISS-1 is a prototype for future research projects requiring large-scale computing in Canada. This is the first step towards making CISS a regular event on the Canadian research landscape.

Le 4 novembre 2002, un super-ordinateur virtuel pan-canadien a fourni à une équipe de chimistes l'équivalent de plusieurs années de calcul en une seule journée. Cette expérience, appelée CISS-1 (Canadian Internetworked Scientific Supercomputer)*, a eu trois impacts sur la recherche : (1) En collaboration avec le C3.ca, elle a créé un précédent de coopération entre les centre de calcul canadiens; (2) elle a démontré la mise à l'échelle et la capacité du système* Trellis *dans le calcul de haute performance sur une vaste étendue géographique; (3) elle a produit un nouveau résultat en chimie numérique.*

En utilisant environ 1 376 processeurs dédiés, sur 20 centres et 18 domaines administratifs à travers le Canada, environ 3,5 CPU-année de calculs ont été complétées. CISS-1 est un prototype de projets futurs demandant de vastes ressources de calcul au Canada. Il s'agit de la première étape visant à faire de CISS un événement régulier sur la scène de la recherche canadienne.

1 Introduction

Over the past four years, the Canada Foundation for Innovation (CFI) has partnered in acquiring $160 million in high-performance computing (HPC) equipment at over 20 sites in Canada. These resources have had a huge, positive research impact. However, this funding strategy is not amenable to creating large computing centres. For example, there is still only one Canadian academic site on the November 2002 listing of the world's Top 500 Supercomputer Sites [1]. Unfortunately, some scientific problems require computational resources greater than that of any single site (or regional consortia) in Canada.

CISS (pronounced "kiss"), the Canadian Internetworked Scientific Supercomputer, is an attempt to harness the research computing capacity in Canada to create a virtual supercomputer. Creating a nation-wide virtual supercomputer has long been a goal of C3.ca [2], Canada's national high-performance computing consortium, with support from a Natural Sciences and Engineering Research Council of Canada (NSERC) Major Facilities Access (MFA) Grant. The idea for CISS was spawned at the HPCS 2002 conference. With a CFI policy that requires each facility to make at least 20% of their resources available to external users, the timing was right to realize this vision.

Grid software packages, such as Globus [3], have the potential to meet our needs, but they require all participating sites to install and configure several software packages. Instead, we decided to use the Trellis system, developed by Paul Lu and colleagues at the University of Alberta [4], for the CISS experiment. Trellis emphasizes the use of widely-deployed software installed at the user-level, thus reducing the dependence on system administrators for extensive installation and configuration. For example, the Trellis *placeholder scheduling* model [5,6] is a simple, user-level approach to global scheduling over a wide-area network of heterogeneous resources.

The first CISS experiment, dubbed CISS-1, gave chemist Dr. Wolfgang Jäger and his research team the chance to get a new research result in a single day [7,8]. The application required up to six years of computing time. Normally, he would not tackle a problem of this size since other researchers with access to more computing power could beat him to the result.

The CISS-1 application, based on the MOLPRO software package [9], was "embarrassingly parallel" – filling in data points in a three-dimensional spatial grid. A parameter space study is the natural way to formulate the problem. This application was deliberately chosen for its simplicity, since it was felt that the logistics of getting more than 1,000 processors working together the first time would require extensive effort. It did, and even then we underestimated how time-consuming it would be!

CISS is an attempt to build the software *and* social infrastructure for a Canada-wide metacomputer (i.e., a useful aggregate of individual computer systems). Although there are still many unsolved technical problems in the metacomputing infrastructure, it may be an even greater challenge to establish a community of researchers to effectively share HPC resources. CISS-1 is just the first step towards our long-term vision. We envision CISS being utilized on a regular basis by researchers from across the country. This would be unique in the world and a tremendous opportunity for Canadian scientists to tackle problems that they previously thought were too large.

This paper is a semi-technical discussion of the motivation, implementation, and outcomes (both technical and social) of CISS. The contribution is in documenting and quantifying CISS as well as the lessons learned from the experience. Finally, we wish to acknowledge the support and hard work of a lot of people and agencies that made CISS possible, as detailed in the Appendix.

2 Trellis and Related Work

The concept of grid computing is popular these days. The goals of the Trellis project are more modest and simpler than that of grid computing. In particular, Trellis is focussed on supporting scientific applications on high-performance computing systems; supporting business applications and Web services are not explicit goals of the Trellis project. Therefore, we prefer to use older terminology—metacomputing—to reflect the more limited scope of our work. Currently, Trellis does not use any of the new software that might be considered part of grid computing, but the design of Trellis supports the incorporation of and/or co-existence with grid technology in the future.

In computing science, the dream of metacomputing has been around for decades. In various forms, and with important distinctions, it has also been known as distributed computing, batch scheduling, cycle stealing, peer-to-peer systems, and (most recently) grid computing. Some well-known, contemporary examples in this area include SETI@home [10], Project RC5/distributed.net [11], Condor [12], and the projects associated with Globus/Open Grid Service Architecture (OGSA) [3]. Of course, there are many, many other related projects around the world, including some in Canada (for example, Grid Canada [13] and the University of Victoria Grid Testbed [14]).

The Trellis philosophy has been to write the minimal amount of new software and to require the minimum of superuser support. Simplicity and software re-use have been important design principles. Trellis uses mostly widely-deployed, existing software systems so that system administrators who want to participate in CISS do not have to install new software. In fact, all that is needed to integrate a new site into CISS is for the system administrator to create a regular user account.

2.1 Overlay Metacomputers

Users often want to aggregate the computing power of a number of individual computers to get maximum job throughput and minimize makespans (i.e., time from start to finish to complete a workload). Using the Trellis software, users create an *overlay metacomputer* from a list of computers they have access to. An overlay metacomputer is like a grid or virtual supercomputer, except that the metacomputer exists entirely at the user-level and is considered an overlay because different metacomputers can be created on a per-user and per-application basis. The overlay metacomputer can be as simple as one computer, or as complicated as hundreds of computers in many administrative domains. Therefore, CISS is an overlay metacomputer of Canadian HPC systems.

2.2 Metacomputing Applications

In HPC, one often makes a distinction between *capacity* and *capability* computing. Capacity computing is centered on maximizing the throughput for a workload with multiple jobs. A large set of independent jobs is typical of capacity computing. Capability computing is centered on minimizing the turnaround time of individual jobs. Often, capability computing deals with applications that require advanced parallel algorithms, low-latency interconnection networks, high-performance input/output, or large amounts of physical memory to complete in a reasonable amount of time.

At the level of the workload, CISS is focussed on capacity computing; the goal is to complete as many independent jobs as possible. At the level of the individual jobs, CISS supports applications that are shared-memory or message-passing codes (i.e., capability computing) as long as all of the processes execute within the same machine or cluster [5,6]. But, so far, the CISS applications have all been embarrassingly parallel sequential jobs; each job is an independent data point in a parameter space.

Fortunately, embarrassingly parallel workloads are a natural and common occurrence in computational science, including chemistry, physics, discrete simulations, and bioinformatics. But, since not all workloads have independent jobs, future CISS experiments will address more sophisticated workloads, including workflow dependencies between jobs. For example, pipelines and job dependency graphs are supported by the Trellis system [15].

2.3 Issues in Global Scheduling

Metacomputers (and grids) need a system for global resource scheduling and allocation to deal with:

1. **Multiple Administrative Domains**: A global scheduler must be able to make use of resources from a wide variety of administrative domains and should do so while adhering to the local policies set out by the administration.

2. **Local Batch Scheduler Interaction**: Many HPC sites employ one or more batch schedulers to govern access to local computational resources. A global scheduler should take advantage of the underlying local batch scheduler, if it exists, by layering on top of it.

3. **Load Balancing**: A global scheduler is expected to improve performance by load balancing. Ideally, a global scheduler should provide lower makespans than any alternative methods the user has available.

2.4 Placeholder Scheduling

A placeholder is a mechanism for global scheduling in which each placeholder represents a potential unit of work. For further details, the interested reader is referred to other papers [5,6]. The current implementation of placeholder scheduling uses normal batch scheduler job scripts to implement a placeholder. Placeholders are submitted to the local batch scheduler with a normal, non-privileged user identity. Thus, local scheduling policies and job accounting are maintained.

When the job (a.k.a. placeholder) begins executing, it contacts a central server and requests the job's actual runtime parameters (i.e., late binding). For placeholders, the communication across administrative domains is handled using Secure Shell (SSH). In this way, a job's parameters are *pulled* by the placeholder rather than *pushed* by a central authority. In contrast, normal jobs hard-code all the parameters at the time of local job submission (i.e., early binding). Through the late binding of job parameters to placeholders, the global scheduler has maximum flexibility in implementing load balancing policies.

Placeholders do not require any special software, other than SSH, to be installed on individual computers. Moreover, they utilize existing normal user accounts at various sites, thereby allowing the sites to enforce per-user policies such as CPU accounting, disk quotas, and access restrictions.

2.5 TrellisWeb

TrellisWeb is an evolution of the PBSWeb system that was originally designed to simplify job submission to the Portable Batch System (PBS). TrellisWeb provides a convenient interface to placeholder scheduling to address:

1. **Resource Scheduling and Allocation**: With placeholder scheduling, global jobs can co-exist with local jobs from local users and all jobs are subject to local policies, such as priority schemes and queue structures. Additionally, zero-infrastructure placeholders can be used at sites without an underlying batch scheduler. Zero-infrastructure placeholders are, essentially, a script with an infinite loop of "get next job, compute job, and return result".

2. **Single Log On**: TrellisWeb provides a Web-based graphical user interface (GUI), or portal, with a single username/password access to overlay metacomputers.

3. **Access Control**: Placeholder scheduling and TrellisWeb use normal, non-privileged user accounts to provide a metacomputing environment. Therefore, the local site policies and access control are reflected implicitly within the system.

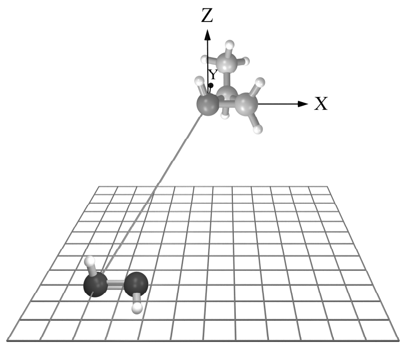

Figure 1. The "shifting" of the H_2O_2 molecule with respect to the space-fixed propyleneimine molecule.

3 The CISS-1 Application

The computational chemistry experiment was designed by Wolfgang Jäger, Aiko Huckauf, and Yunjie Xu of the Department of Chemistry, University of Alberta [8]. The chemistry application is the computational determination of chiral molecule interaction energies. The following description comes from Dr. Jäger.

"An object that cannot be superimposed with its mirror image is called chiral. Our hands are the most familiar example of such chiral objects. Image and mirror image are called enantiomers of the chiral object. Chiral molecules play important roles in nature. For example, life itself is based on single enantiomeric forms of amino acids and sugars, a fact known as "homochirality of life". Many pharmaceuticals are chiral molecules of which one enantiomer has a beneficial effect whereas the other may have severe side effects.

"At the heart of it all is chiral recognition or chiral discrimination. A handshake is possible between two left or two right hands, but not between left and right hand. On the molecular level, there is an energy difference between the interaction of, for example, the right handed form of a chiral molecule and the two enantiomeric forms of another substance.

"How can this subtle energy difference have such a profound effect in nature? The CISS-1 experiment involved computing the interaction energy of two chiral molecules at many different separations and orientations. The specific system consisted of 1,2-propyleneimine (an aziridine derivative) and hydrogen peroxide. The calculations were carried out on two different hydrogen peroxide enantiomers. An analysis of the results gives a "chiral recognition surface" that identifies the active sites of the chiral components."

To calculate the potential energy surfaces, a three-dimensional (spatial) grid was created with the fixed molecule at the centre. Calculations of potential energy between an enantiomer and the fixed molecule were performed at each point of the grid to yield the entire energy surface (for example, Figure 1). Because of the nature of the calculation, the grid may be arbitrarily dense, resulting in a potentially infinite number of points. However, the grid was chosen so as to provide adequate granularity for interpreting the potential energy surface while still being computationally feasible (even with a large number of computers participating in computing results). Each point may require several hours of computing time on a modern workstation. In computational terms, this is an ideal parallel problem which can be distributed onto as many processors as points needed and is therefore ideally suited for the CISS experiment.

The commercial product MOLPRO was used for the electronic structure calculations [9]. The computations for each single point were performed using a state-of-the-art level of theory. For this particular experiment, a single MOLPRO job requires a site-specific amount of main memory and between 0.5 GB and 1 GB of temporary disk space. On an average contemporary processor (such as an AMD Athlon CPU running at 1.3 GHz with 256 KB cache available in the Department of Chemistry at the University of Alberta), MOLPRO takes approximately four hours to compute one data point.

4 The CISS-1 Experiment

4.1 CISS-1 Environment

Porting any application to a heterogeneous set of computing platforms is always a labour-intensive and frustrating experience — CISS-1 was no exception. Some of the minor and majors issues that had to be addressed included:

- Hardware. A variety of different hardware platforms were used, each with their own set of peculiarities (see Table 1).

- Operating system. A variety of operating systems were encountered, each with their own set of peculiarities.

- Compilers. MOLPRO had particular requirements with respect to C and Fortran compilers. Often, the widely-available GNU compilers were not able to build the code correctly and we had to use either the Portland Group or Intel compilers, if available.

- Batch schedulers. CISS-1 used systems running PBS, Sun Grid Engine (SGE), and IBM's LoadLeveler. On systems that did not have a batch scheduler, we used a version of Trellis placeholders that did not depend on a local scheduler (the zero-infrastructure placeholders).

- Quotas. Many sites had explicit or implicit disk quotas which had to be removed. In many cases, we only found out about the quotas the hard way.

- Network. Some sites had firewalls or gateway nodes. Assistance was needed to communicate securely through them.

- Disk. Disk was located in different places on different systems. Some disk was local, some was shared across a storage area network (SAN). The chemistry application required roughly 1 GB of temporary storage for each job. For CISS-1, our experience was that local disk was much faster than a shared, parallel file system.

- Secure shell. Some sites used the commercial version of SSH, which turned out to have some incompatibilities with the open-source OpenSSH (e.g., the on-disk format of the public/private keys).

4.2 The Throughput Experiment

On November 4, 2002, the CISS-1 experiment combined 1,376 CPUs from 20 different HPC systems (16 different institutions, 18 different administrative domains) across Canada to cooperatively execute the computational chemistry parameter space study (Table 1) [6]. The experiment received a lot of media exposure, and the Web site received over 30,000 hits in the 24-hour period [7]. In our opinion, the most noteworthy metric of CISS-1 is the 18 different administrative domains, since that represents an unprecedented amount of (human) cooperation among Canadian HPC centres and one of the largest cross-domain metacomputers reported to date.

The CISS-1 experiment was performed as a 24-hour throughput test (i.e., capacity computing). For convenience, TrellisWeb was used to manage placeholder scheduling. Starting on November 4, 2002 at 12:00 midnight, and ending on November 5, 2002 at 12:00 midnight, placeholders were executed on the systems shown in Table 1. The two days prior to November 4, 2002 were used as a ramp-up period to finish integrating several sites and bring all sites up to full capacity, but only the computations performed during the 24-hour period will be discussed here. Several other institutions offered additional computing resources for CISS-1, but (regrettably) a lack of time prevented us from integrating them into the experiment.

A high-level overview of the throughput of the CISS-1 experiment is shown in Table 2. Performance varies from site to site because of differing numbers of processors, placeholders launched at each site, and how well-suited MOLPRO was for the particular hardware architecture at a given site. For example, since MOLPRO uses a substantial amount of temporary disk space during a computation, systems with fast, local disk drives (e.g., clusters) performed much better than systems with shared, parallel file systems.

A total of 7,593 work units (jobs) were completed by CISS-1 in the official 24-hour period.[1] The top four sites, in terms of throughput, were the CLUMEQ cluster in Montreal (`stokes.clumeq.mcgill.ca`) with 2,162 units, the cluster at Simon Fraser University in Burnaby

[1] Many more units of work were completed during the development of CISS, the ramp-up period, and after November 4, 2002. Over 27,000 units of work have been completed in total.

No.	Site	Description	Local Scheduler	PH
1	athlon-cluster.nic.ualberta.ca	x86 Linux Cluster	PBS	32
2	aurora.nic.ualberta.ca	SGI Irix	PBS	236
3	brule.cs.ualberta.ca	x86 Linux Cluster	SGE	26
4	bugaboo.hpc.sfu.ca	x86 Linux Cluster	Zero-Infrastructure	192
5	chromosome1.ocgc.ca	SGI Irix	PBS	96
6	deeppurple.sharcnet.ca	Alpha Linux Cluster	Zero-Infrastructure	22
7	driftwood.iam.ubc.ca	x68 Linux Cluster	PBS	16
8	gnome.usask.ca	x86 Linux Cluster	Zero-Infrastructure	32
9	hammerhead.sharcnet.ca	Alpha Linux Cluster	Zero-Infrastructure	22
10	herzberg.physics.mun.ca	SGI Irix	Zero-Infrastructure	20
11	maci-cluster.ucalgary.ca	Tru64 Alpha Cluster	PBS	176
12	mercury.sao.nrc.ca	x86 Linux Cluster	PBS	48
13	minerva.uvic.ca	IBM AIX SP	LoadLeveler	128
14	monolith.uwaterloo.ca	IBM AIX P-Series	Zero-Infrastructure	16
15	myri.ccs.usherbrooke.ca	x86 Linux Cluster	Zero-Infrastructure	8
16	p4-cluster.nic.ualberta.ca	x86 Linux Cluster	PBS	26
17	stokes.clumeq.mcgill.ca	x86 Linux Cluster	PBS	248
18	symphony.unb.ca	IBM AIX SP	LoadLeveler	2
19	white.cs.umanitoba.ca	x86 Linux Cluster	Zero-Infrastructure	22
20	zodiac.chem.ubc.ca	x86 Linux Cluster	PBS	8
			Total	1376

Table 1
CISS-1 Sites (PH = placeholders; typically one PH per CPU)

(`bugaboo.hpc.sfu.ca`) with 1,575 units, the various clusters and machines at the University of Alberta in Edmonton with 1,275 units, and the MACI cluster at the University of Calgary (`maci-cluster.ucalgary.ca`) with 1,065 units.

When examining these numbers, several points should be kept in mind:

1. Most sites possessed much more computing power than that used for CISS-1. Therefore, the throughput numbers should be viewed as lower bounds on the HPC resources at the various sites.

2. Unfortunate, but normal, hardware/software failures at a few sites affected their throughput. For example, a failed hard disk at the University of Calgary and a reboot at the University of Alberta led to a gradual restart of the placeholders and computations at both sites. Without the failures and reboots, several sites would have achieved better throughput. As an aside, we are fairly pleased with our software's ability to deal with faults.

3. The MOLPRO application was not substantially tuned for the architecture at each site. When possible, we used the recommended build procedures from the developers of MOLPRO, but the emphasis in CISS-1 was on the metacomputing aspects of the experiment and not on optimizing MOLPRO for each system.

Overall, CISS-1 consumed approximately 2.94 years (25738:58:40) of computing time across all sites. The mean time required to compute a work unit was approximately 3.39 hours (3:23:23). However, if we instead assume that a work unit takes approximately four hours to compute on a contemporary x86 cluster processor located in the Department of Chemistry at the University of Alberta, the total computation comes out to about 3.46 years worth of computation (7,292.94 work units at four hours per work unit). These numbers could be even better, but (as discussed above) a number of faults resulted in several hundred processors being unavailable for a few hours.

Most sites exhibit a large amount of variability (as evidenced by standard deviation σ) due to the fact that MOLPRO required a variable amount of time depending on the particular data point being computed. Other sources of variability include heterogeneous CPU clock rates (e.g., the various SGI Origins at the University of Alberta and the different Alpha-based nodes at `maci-cluster.ucalgary.ca`) and contention for shared storage systems.

Some of the systems, such as `athlon-cluster.nic-.ualberta.ca`, `brule.cs.ualber-ta.ca`, and `bugaboo.hpc.sfu.ca` had relatively steady and predictable rates of execution. Although computational ability varies among the sites, the overall rate within these sites stayed fairly consistent. This is largely due to the fact that these sites employed the use of local scratch disks present at each node (all three are clusters). In contrast, sites such as `aurora.nic.ualberta.ca` and, to some extent `maci-cluster.ucalgary.ca`, exhibited spikes in work completion due to the use of a parallel file system (as with `aurora.nic.ualberta.ca`), or

No.	Site	Number of Jobs	Total CPU Time	MeanTime Per Job	σ
1	athlon-cluster.nic.ualberta.ca	307.75	751:12:52	2:26:27	0:46:20
2	aurora.nic.ualberta.ca	437.17	4530:30:47	10:21:46	4:34:15
3	brule.cs.ualberta.ca	433.31	958:13:50	2:12:41	0:33:10
4	bugaboo.hpc.sfu.ca	1575.22	3984:29:37	2:31:46	0:36:55
5	chromosome1.ocgc.ca	136.79	1310:10:33	9:34:41	3:40:41
8	gnome.usask.ca	159.13	729:58:24	4:35:14	1:29:49
10	herzberg.physics.mun.ca	107.37	441:04:44	4:06:29	1:15:01
11	maci-cluster.ucalgary.ca	1064.61	2919:52:07	2:44:34	1:03:15
12	mercury.sao.nrc.ca	397.61	941:16:43	2:22:02	0:43:51
14	monolith.uwaterloo.ca	356.09	383:37:03	1:04:38	0:09:06
15	myri.ccs.usherbrooke.ca	73.02	161:45:13	2:12:54	0:38:28
16	p4-cluster.nic.ualberta.ca	96.56	579:38:53	6:00:10	2:47:13
17	stokes.clumeq.mcgill.ca	2162.47	5625:23:44	2:36:05	0:47:12
19	white.cs.umanitoba.ca	57.46	479:57:20	8:21:10	2:37:03
	other	228.36	1941:46:53	8:30:11	7:38:53
	Overall	7592.94	25738:58:40	3:23:23	2:40:54

Table 2
CISS-1 Throughput: Number of Jobs Completed All times are reported as H:MM:SS.

some cluster nodes sharing the same scratch disk (as with `maci-cluster.ucalgary.ca`). Because of the temporary storage requirements of MOLPRO, and because such storage is heavily used, MOLPRO jobs may contend with one another in a shared file system situation. `aurora.nic.ualberta.ca` experienced a severe I/O bottleneck at the parallel file system that resulted in high contention and high mean execution time.

4.3 CISS-2

CISS-2 was run over the Christmas holidays, from December 23, 2002 to January 2, 2003. In contrast to CISS-1, two applications were used: a molecular dynamics application from Dr. Peter Tieleman's group (University of Calgary) and a physics simulation from Dr. David Sénéchal's group (Université de Sherbrooke). Also, in sharp contrast to CISS-1, we did not request exclusive access to the computing resources; CISS-2 was designed to use whatever cycles might be available during the holidays. A subset of the sites from CISS-1 were used for CISS-2, including: CLUMEQ (at McGill University), the Hospital for Sick Children (Toronto), Memorial University of Newfoundland, Simon Fraser University, and the Universities of Alberta, Calgary, New Brunswick, Saskatchewan, and Victoria.

The CISS-2 experiment is still being analyzed. Since the machines were not dedicated for CISS-2, it is more difficult to analyze the throughput in a meaningful way. However, it is known that CISS-2 completed more than the originally-planned workload. In fact, more jobs were created and added during the experiment's time window and, in the end, CISS-2 completed almost twice the original number of computations.

5 Concluding Remarks

Although there was room for improvement, both CISS-1 and CISS-2 were successes, as judged by many metrics. CISS showed that there is a need for large-scale high-performance computing in Canada. The experiments produced new research results in computational science [8,16]. Finally, CISS achieved national and international media exposure, furthering the campaign for an increase in the commitment of funding for HPC in Canada.

Perhaps most importantly, CISS helped foster a new social infrastructure for sharing computational resources in Canada. The sociology behind the sharing of multi-million dollar facilities worked well, but it continues to need work and attention. This project convinced some of the skeptics, but not all of the skeptics. We will continue to build the cross-institution bridges and contacts required for future CISS experiments. More HPC centres can be, and must be, included for future efforts.

To ensure a successful continuation of the CISS initiative, several things have to happen:

1. CISS should become a regular event on the Canadian research calendar. Computational scientists should be encouraged to tackle problems on the scale of CISS. Whether CISS should be scheduled as a few days per month, set aside specifically for CISS, or whether it should be irregular, based on demand, remains to be seen.

2. Given the large commitment of computing resources involved, CISS should only endeavor to undertake the highest-quality research projects. This requires a proper refereeing process to evaluate the impact of the research and the need for the resources.

3. Substantially more resources need to be devoted to the CISS effort; for example, CISS could be taken

over and administered by C3.ca under the TASP program.

Worldwide, there are many, large grid computing and metacomputing projects. Computational science and high-performance computing have become vital, complementary approaches to theoretical and experimental science. When appropriate, we should borrow inspiration and technology from other projects. When it makes sense, we should develop our own approaches and infrastructure solutions. But, if Canada does not define and commit to its own strategy for HPC, we risk putting artificial limits on the imagination and innovation of our scientists.

Acknowledgments

Financial support was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), C3.ca Association Inc, and Alberta's Informatics Circle of Research Excellence (iCORE). Canada Foundation for Innovation (CFI) funding supported the acquisiton of most of the HPC resources used in this research.

References

1. TOP500 Supercomputer Sites. `http://www.top500.org/`.
2. C3.ca Association Inc. `http://www.c3.ca/`.
3. Globus Project. `http://www.globus.org/`.
4. Trellis Project. `http://www.cs.ualberta.ca/~paullu/Trellis`.
5. Christopher Pinchak, Paul Lu, and Mark Goldenberg. Practical Heterogeneous Placeholder Scheduling in Overlay Metacomputers: Early Experiences. In *Proc. 8th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 85–105, Edinburgh, Scotland, UK, July 24, 2002. Also published as Springer-Verlag LNCS 2537 (2003), pages 205–228. Available at `http://www.cs.ualberta.ca/~paullu/`.
6. Christopher Pinchak. Placeholder scheduling for overlay metacomputing. Master's thesis, Department of Computing Science, University of Alberta, 2003.
7. CISS – The Canadian Internetworked Scientific Supercomputer. `http://www.cs.ualberta.ca/~ciss`.
8. Yunjie Xu, Aiko Huckauf, Wolfgang Jäger, Paul Lu, Jonathan Schaeffer, and Christopher Pinchak. The CISS-1 experiment: *ab initio* study of chiral interactions. In *39th International Union of Pure and Applied Chemistry (IUPAC) Congress and 86th Conference of The Canadian Society for Chemistry*, Ottawa, Ontario, Canada, August 10–15, 2003.
9. MOLPRO Quantum Chemistry Package. `http://www.molpro.net/`.
10. SETI@home. `http://setiathome.ssl.berkeley.edu/`.
11. RC5 Project. `http://www.distributed.net/rc5`.
12. Condor. `http://www.cs.wisc.edu/condor`.
13. Grid Canada. `http://www.gridcanada.ca/`.
14. UVic Grid Testbed. `http://grid.phys.uvic.ca/`.
15. Mark Goldenberg. TrellisDAG: A system for structured DAG scheduling. Master's thesis, Department of Computing Science, University of Alberta, 2003.
16. Justin L. MacCallum, Parag Mukhopadhay, Huaqiang Luo, and D. Peter Tieleman. Large scale molecular dynamics simulations of lipid-drug interactions. In *17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS)*, Sherbrooke, Quebec, Canada, May 11–14, 2003.

Appendix

We thank the scientists who helped make CISS-1 and CISS-2 a success by contributing their applications to the experiment: CISS-1: Wolfgang Jäger, Aiko Huckauf, and Yunjie Xu (University of Alberta); CISS-2: Peter Tieleman and Justin MacCallum (University of Calgary); David Sénéchal and Dany Plouffe (Université de Sherbrooke). Thank you to all of the researchers who interrupted their own work to give us exclusive use of the computational resources for CISS-1.

C3.ca's vision, support, and leadership was vital to CISS. We hope to continue and expand C3.ca's role in future experiments. We gratefully acknowledge the role that Canarie and Netera have played in high-performance networking, with respect to CISS and the MACI Project in Alberta.

We thank the authors of MOLPRO, P. Knowles and H. Werner, for permission to use their software for CISS-1.

We acknowledge the contributions of Rob Lake, George Ma, Victor Salamon, Jeff Siegel, Jeremy Handcock, Nolan Bard, Mark Lee, Vanessa Chung, Yaling Pei, and Danny Ngo to the Trellis Project.

The following people graciously committed time and/or resources to make CISS-1 and CISS-2 a success. Please accept our apologies for any inadvertent omissions.

Mark Andrews, Hospital for Sick Children; Bruce Attfield, C3.ca; Roman Baranowski, University of British Columbia; Virendra C. Bhavsar, University of New Brunswick; David Bickle, Carleton University; Kayla Bonham, University of Calgary; Jonathan Borwein, Simon Fraser University; Mehdi Bozzo-Rey, Université de Sherbrooke; Chip Campbell, Hospital for Sick Children; Colin Cherry, University of Alberta; Scott Delinger, University of Alberta; Harold Esche, University of Calgary; Darrell Fraser, Aurora College; Robert Fridman, University of Calgary; Peter Graham, University of Manitoba; Wagdi G. Habashi, McGill University; Mark Hahn, McMaster University; Jason Hlady, University of Saskatchewan; Marc La France, University of Alberta; Colin Leavett-Brown, University of Victoria; Waldemar Lysz, University of Alberta; Chris MacPhee, University of New Brunswick; Paul Masiar, C3.ca; Gabriel Mateescu, National Research Council; Gary Molenkamp, SharcNet; John Morton, University of Guelph; Fred Perry, Memorial University of Newfoundland; Doug Phillips, University of Calgary; Wayne Podaima, National Research Council; Andrew Pollard, C3.ca; Darryl Reid, Memorial University of Newfoundland; Sean Seeley, University of New Brunswick; Ron Senda, University of Alberta; Martin Siegert, Simon Fraser University; Luc Simard, McGill University; Rob Simmonds, University of Calgary; Amik St-Cyr, McGill University; Mark Thachuk, University of British Columbia; Brian Unger, University of Calgary; Alain Veilleux, Université de Sherbrooke; Chris Want, University of Alberta; Kevin Watts, University of Alberta; Norris Weimer, University of Alberta; Douglas Weir, University of Waterloo; Len Zaifman, Hospital for Sick Children.