Enhancing Portability of HPC Applications across High-end Computing Platforms

Magdalena Sławińska, Jarosław Sławiński, Dawid Kurzyniec, Vaidy Sunderam

Dept. of Math and Computer Science, Emory University
400 Dowman Drive
Atlanta, GA 30322, USA
{magg, jaross, dawidk, vss}@mathcs.emory.edu

Abstract

Fast hardware turnover in supercomputing centers, stimulated by rapid technological progress, results in high heterogeneity among HPC platforms, and necessitates that applications are ported and adapted frequently. The cutting-edge nature of the hardware mandates customized performance tuning, which, coupled with continuously growing application complexity, makes the process inherently and increasingly challenging. In this paper, we analyze build procedures of a representative set of HPC applications, and attempt to identify commonalities that can be exploited to enhance cross-platform portability. We then propose a novel method for reducing non-portabilities while preserving high performance. The approach, based on profiles *that capture and isolate non-portable features at various levels, requires only a moderate amount of changes to existing makefiles. It leverages the expertise of system designers and administrators, and reduces burdens placed on application scientists. As a proof of concept, we discuss the application of our methodology to enhancing portability of the Milc application across heterogeneous HPC platforms.*

1 Introduction

Vigorous technological progress in the area of high performance parallel computing and networking results in fast hardware turnover in supercomputing centers and research labs around the globe. About a third of the entries from the Top 500 list drops out every six months [3] and is replaced by faster machines. Diversity within installations is therefore commonplace. For instance, machines presently utilized at the National Center for Computational Sciences (NCCS) at Oak Ridge Research Laboratory (ORNL) range from Cray X1E MSP, to Cray XT3 running Catamount microkernel, to SGI Altix SMP based on Intel Itanium2 processors, to a Linux-based visualization cluster. The typical life span of a single machine is 2 years, and new systems are deployed about twice per year. At the same time, applications running on these resources grow more and more sophisticated, and exhibit much longer life spans. Indeed, some current applications have been already ported to over a *hundred* different platforms during their life time [4, 6, 10].

The process of adapting HPC applications to emerging platforms is intrinsically challenging because of the cutting-edge nature of the target platforms. To maximize performance, dedicated tools, compilers, and specific optimization techniques must be used. Straightforward porting, even though often possible, may lead to significant performance degradation [5]. Furthermore, rapid technological development makes optimization a moving target, as each new compiler release may include a new set of optimization flags and techniques. Traditionally, this problem had been approached by application scientists through meticulous, manual, trial-and-error procedures, involving re-writing declarative parts of build files from scratch for every application on every new system. However, with applications increasing in complexity, this approach is no longer viable. By some estimates [12], build-related efforts can contribute to as much as 30% of the development costs, and it is clear that these efforts should not be repeated when the application is ported to a new platform. Furthermore, we note that tuning requires expertise at the hardware and system level – expertise that is a burden and but a necessary evil for application scientists as it detracts from their ultimate, application-related scientific objectives. We also note that automated build technologies recently gaining popularity in the HPC application community, notably GNU Autotools,

1-4244-0910-1/07/$20.00

do not provide a panacea to portability-related problems since they still require the user to provide system-specific configuration parameters and tuning flags.

In this project, we have attempted to analyze past application porting scenarios accumulated in their build files over the years, trying to identify potential traits and patterns with the hope that they can be exploited to simplify the process in the future. In Section 3, we present an analysis of build systems from a representative set of high-performance applications, highlighting discovered trends, commonalities, and differences across a baseline set of heterogeneous platforms. Our initial observations allow us to propose a new methodology for porting and tuning HPC applications. The methodology, described in Sections 4 through 6, is based on the concept of *profiles* that isolate build procedures from variable parameters. We capture platform-specific and system-specific intricacies (e.g. compiler flags and options) into hardware and system profiles that contain the aggregated knowledge of vendors and administrators. Application build files can then be shielded from specific toolkits and suites by referring to their abstract capabilities (e.g. "compile with loop unrolling", or "preserve strict FP semantics"), thus greatly improving portability. We demonstrate that our methodology requires only moderate modifications to existing build files, and we propose tools to facilitate the migration process (Section 6). Finally, in Section 7 we present the Milc application [1] use case to illustrate the viability of our approach.

2 Related Work

Portability and build-related aspects have been recognized to incur increasingly significant development costs in the high performance scientific computing applications arena [8]. The current common practice in addressing the cross-platform build issue has been simply to maintain separate versions of makefile "includes" for each architecture or even particular machines, containing predefined sets of experimentally selected configuration and tuning parameters [4, 6, 10]. Clearly, this approach makes code maintenance and development difficult, and does not cope well with porting codes to emerging architectures. Recently, a trend to migrate to GNU Autotools [15] can be observed. However, converting makefile-based applications to GNU Autotools requires significant effort [12], and introduces its own compatibility issues (e.g. requiring compatible versions at the user's and developer's sides [7]). Also, as a general-purpose technology, GNU Autotools does not fully address all aspects characteristic to the HPC domain, such as cross-compilation, use of restricted microkernels, and tuning to the *hardware* environment.

Another way to tackle the build problem is to provide a uniform environment that supports universal build procedures. In the HPC arena, Eclipse PTP [14] provides a standard GUI-oriented *development* and *run-time* interface to heterogeneous systems. Our project can be perceived as complementary to Eclipse PTP in that it supplies Eclipse plugins that enhance the IDE with additional support for the *build* process.

The other important aspect of the build problem is how to acquire and manage the information necessary for the successful build. The Environment Modules project [9, 13] provides users with a tool to manage their computational environments where configuration information for various packages and libraries installed on a given computer system is captured in the form of *modules* that can be *loaded* and *unloaded* dynamically and automatically to modify the user's environment variables such as `PATH` or `LD_LIBRARY_PATH`. We take this concept further and propose *profiles* to: (1) dynamically switch among different configurations; and (2) rectify the configuration information related to distinct domains such as hardware, system software, and the application itself, and separate it from the actual build files.

3 Study of Build Configuration

We have examined build processes in two groups of computational applications: 37 applications from LCF (*Leadership Computing Facility*) at ORNL [11] and 29 applications from OpenScience [2]. The results are summarized in Table 1.

Applications from the LCF group cover a wide spectrum of computational science: physics, nanoscience, climate, biology, chemistry, computer science, engineering. For the applications from OpenScience, we focused on two most numerous groups: *Chemistry* and *Bioinformatics* (*Life Science*). Information about the configuration/installation procedures was obtained from source codes (if they were available) or from web pages (installation instructions).

As Table 1 indicates, most applications use either GNU Autotools (38% LCF, 45% OpenScience) or proprietary makefiles (43% LCF, 35% OpenScience). The remaining applications rely on custom scripts (shell, Perl, Python) either for the actual build or to generate makefiles. To port build files to new platforms, the user is required to set appropriate environment variables, set/modify variables in makefiles, and, in a few cases, make changes in the include and source files (e.g. changing macro definitions, or fine-tuning low-level routines). The resulting sets of configuration parameters tuned for a particular platform or machine are usually retained with the application, providing a starting point for porting to a similar architecture in the future. In most cases, different applications on the same baseline platform tend to use similar tools, compilers, and command-line switches; nevertheless, significant differences occur in

Group	Build steering methods	LCF (37)		OpenScience (29)	
GNU Autotools	• environment variables • `./configure` parametrization • modification of configuration files or file structures (e.g. copy some configuration files into a given directory)	14	38.00%	13	45.00%
Shell scripts	• the same as for Autotools • script modification (hard because of many conditional structures)	3	8.00%	1	4.00%
Makefile(s)	• the same as for shell scripts • choice of predefined makefiles	16	43.00%	10	35.00%
Scripts → makefiles(s) or templates script → makefiles(s) templates repository of specific architectural 'profiles'	• a script generates makefile(s) and after it the user can run makefile that is made according to passed parameters. Scripts can use some repositories (as profiles) to create a proper makefile. • the same as for shell scripts	3	8.00%	3	10.00%
Other	• environment variables • modification in build files • interactive methods	1	3.00%	2	7.00%

Table 1. Installation procedures of two applications sets: LCF and OpenScience. Percentages may not sum up to 100% due to round-off errors.

build and source files due to: proprietary names of variables (e.g. `IRAT, debug`), references to macro definitions in header files (e.g. `# define HAVE_IEEEFP_H 1`), conditional expressions (e.g. `if [ $ debug ] ...`), or commented-out variables.

In summary, porting of build files of HPC makefile- as well as Autotools-based applications is rather cumbersome and requires specific knowledge from domains of the hardware, operating system and the application itself. In the following sections, we propose a novel approach to address this problem and reduce deployment-related burdens placed on application scientists.

4 Through Profiles to Portability

Experiences from our "build" research show that in general, information about configuration and installation parameters could be extracted from current build files and stored in separate configuration files that we call *profiles*. We identify several different profile types, focusing on aspects of *hardware*, *system*, *user-level*, and *application* configuration, and facilitating separation between different areas of expertise (Figure 1). *Hardware profiles* store information about processor type, processor speed, available cache memory, interconnect type, etc. that could be provided by hardware vendors. *System profiles*, prepared by site administrators, contain data related to a given system deployment: paths to compilers, libraries, building tools, categorized switches for a given compiler (optimization, debug, ...), and so on. *User profiles* capture user preferences and settings by augmenting or partially overriding system defaults, much in the same way that user-level `/home/user/.profile` files complement system-wide `/etc/profile` files for shell configuration. While system and user profiles describe *how* the software is installed, *application profiles* answer the question *what* to install. They are provided by developers and store application-specific tuning preferences, macro definitions or specific variables and may override or select among values defined at the system and user level. Ideally, application profiles refer to abstract, standardized capabilities rather than individual tools and command-line switches, so that porting an application to a new platform requires minimal effort providing that hardware, system and/or user profiles contain correct values.

As an example, consider a makefile-based application (Figure 2). In order to make the configuration portable, the developer must identify non-portabilities in makefiles such as compilers' names, optimization parameters, macro definitions, paths, etc. Next, he or she has to move the con-

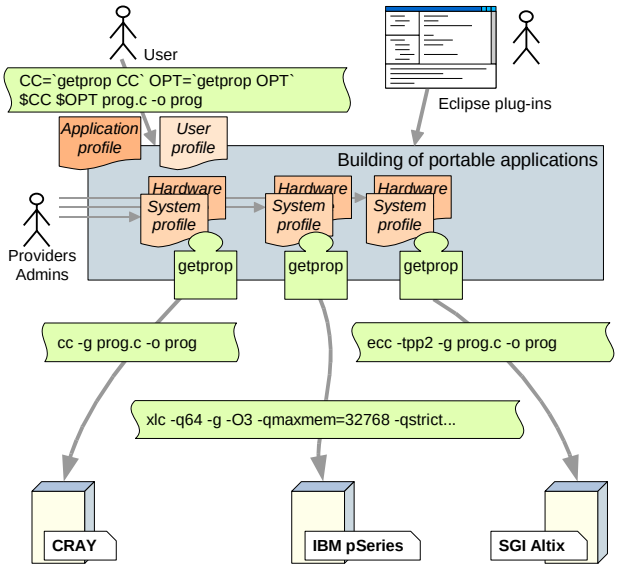

Figure 1. The process of making the application build portable

figuration parameter values to appropriate profiles, and substitute their use in build files with the appropriate data retrieval expressions. Then, further configuration (including tuning and porting to new architectures) may be performed by modifying the appropriate profiles.

The described methodology allows adoption of existing build mechanisms, such as proprietary makefiles, with relatively little effort (as the modifications are confined and have to be made only once) and partially relieves computational scientists from providing full suite of configuration parameters, since most of them are supplied by vendors and administrators in the hardware and system profiles. Still, the end-user can easily customize the build at various levels through user profiles.

5 Profile Design

The basic role of profiles is simply to provide key-value mappings for configuration parameters. Given the need to manage large collection of such mappings and to reflect relationships between them, a tree structure comes as a natural organizational choice. We have decided to use XML for the internal profile representation, due to its well-known advantages such as self-documentation, ease of processing, and support for "low-profile" editing using the simplest command-line tools. The actual key-value mappings are represented in the profile by the "Item" elements. Filesystem-like hierarchies can be built using the "Group" and "Link" elements, analogous to directories and symbolic links in a Unix file system. Two important features in the proposed mapping scheme are *profile inheritance* and *dynamic recursive resolution*. Profile inheritance allows profiles to extend upon one another. For instance, a user profile may extend upon the system profile, and may selectively override individual items or even whole tree branches from its parent. Dynamic recursive resolution allows the item (or group) element values to refer back to other items and groups within the profile. For instance, the value for the "/build/CFLAGS" group can be defined as "${OPT} ${INCLUDE} ${LIB} ${ARCH} ${WARN}", a collection of subgroup definitions. The resolution of ${/build/CFLAGS} would then lead to recursive evaluation and concatenation of "build/CFLAGS/OPT", "/build/CFLAGS/INCLUDE", etc. The ${OPT} expression above defines a (here, relative) reference to a named group (or item) in the profile. Run-time substitution, combined with profile inheritance, introduces an important feature: it allows the user to selectively override *sub-expressions* from the parent profile. For instance, it is possible to modify the optimization options of the C compiler by redefining the "/build/CFLAGS/OPT" group alone in the user profile, leaving other options intact. In fact, this simple model turns out to be powerful enough to be able to support conditional expressions, and to allow system profiles to virtualize configuration options into abstract capabilities, as will be demonstrated in Section 7.

6 Eclipse Portlug-in Tool

As a proof-of-concept we have developed the plugin for Eclipse we dubbed Portlug-in (*Port*ability P*lug-in*). As we mentioned in Section 2, we chose Eclipse due to the particular relevance of Eclipse PTP to the scientific and HPC computing community.

With the Portlug-in wizard the user (i.e. someone who ports an application) creates the new Portlug-in project. Next, the tool allows the user to import the application's source code, and it aids him/her in identifying non-portabilities (e.g., hard-coded variable values) in existing makefiles. These hard-coded values can then be substituted by `getprop` tool invocations. The `getprop` program allows extraction of data from profiles based on the supplied parameter name; to do that, it performs the necessary recursive substitutions and traverses the profile inheritance graph as needed. Finally, the tool aids the user in uploading modified source and configuration files (scripts, makefiles, etc), as well as the appropriate profiles, to the remote site.

7 Experimental Evaluation

In order to verify the viability of our approach, we converted the Milc application (version 6.20sep02) [1] ac-

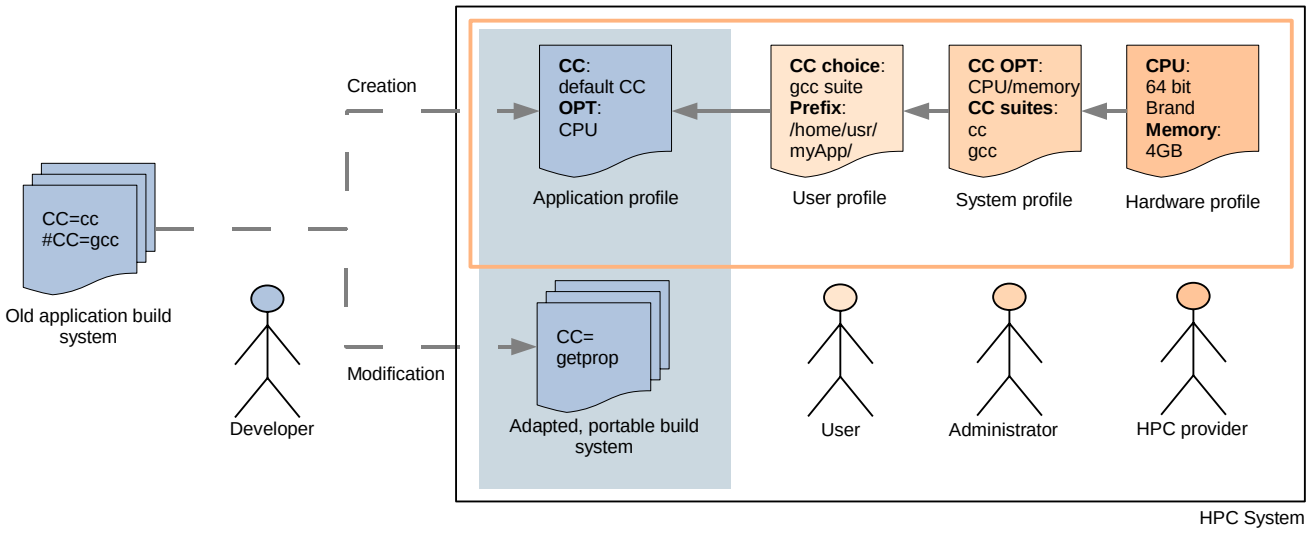

Figure 2. Application, user, system and hardware profiles

```
...
# 1. Compiler
CC      = gcc    # (cc89 gcc xlc gcc pgcc cl g++)
...
#-------------- SUN SPARC -----------
#for Ultra
#OCFLAGS= -fast -dalign -libmil -fsimple=2 -fns
...
# 4. Code alternatives
# usually unchanged
CODETYPE= -DFAST
...
# Complete set of compiler flags - do not change
CFLAGS  = ${OCFLAGS} ${CODETYPE}
...
prefetch.o: prefetch.c
${CC} -g ${OCFLAGS} -c prefetch.c

include Make_template
#!!! we have to change the header file still
# (comment-out define HAVE_IEEEFP)
```

Figure 3. The original makefile for building Milc libraries

```
...
CC       = `getprop applications/milc/CC`
...
OCFLAGS  = `getprop applications/milc/OCFLAGS`
...
CCFLAGS  = `getprop applications/milc/CCFLAGS`
...
# usually unchanged
CODETYPE = `getprop applications/milc/CODETYPE`
...
CFLAGS   = ${OCFLAGS} ${CCFLAGS} ${CODETYPE}
...
prefetch.o: prefetch.c
  ${CC} -g ${OCFLAGS} -c prefetch.c

include Make_template
```

Figure 4. The modified makefile for building Milc libraries with `getprop` expressions

cording to the profile-based methodology in order to make it directly portable between the following heterogeneous computer systems: IBM SP4 (SuSE Linux 9.1 x86-64, gcc|pgcc + mpich), Dell Intel (RedHat Linux 8.0, pgcc|gcc + lammpi), and Sun (SunOS 5.10, cc|gcc + mpich). In the course of this experiment, we succeeded in producing Milc builds equivalent to those resulting from hand-tuned makefiles. For brevity, we demonstrate the system profile details only for the SunOS Sparc platform.

Figure 3 shows a fragment of the original makefile of Milc libraries. First, as we mentioned in Section 4, non-portabilities such as compiler names, flags, etc., must be identified in this makefile and substituted with `getprop` expressions (Figure 4). Figures 5 and 6 present the resulting system and application profiles, respectively. Although at first glance they might look complicated, we note that only a single, platform-independent application profile is needed, so that it can be written once and distributed as a part of the application. Further, the system profile is non-application-specific, so it can be supplied by the site administrator; the burden on the application scientist is thus minimized. It is helpful to think of the profiles in the context of a registry organized as a tree where relevant branches are stored in sub-profiles. The retrieval of actual values from a set of profiles treated as a single entity is the task of `getprop`.

The system profile (Figure 5) contains two main branches /sys and /config. While the system group contains actual *values* concerning the system such as compiler suites (paths, flags for debug, optimization, etc), the config branch describes which alternative configuration variant should be applied by specifying abstract capabilities (e.g. abstract optimization options). The application profile (Figure 6) states *which* flags and options should be applied to the given application (/apps). Similar to the system profile /config, the application profile /config fine tunes the optional parameters.

In order to effectively switch among relevant parts of the system profile, e.g. to choose between different compiler suites, we utilize previously described features of the profile, namely: *links* and *dynamic recursive resolution.* For instance, consider resolving the value of OCFLAGS. The application profile points to /sys/build/currentSuite/CFLAGS that leads to <Link> (/sys/build/currentSuite) in the system profile. At this point we recursively resolve the reference /config/sys/build/suite (still in the system profile) that leads to the value ./suites/gnu. By continuing this process, we finally arrive at the CFLAGS value of -O2. As mentioned before, the combination of recursive resolution with inheritance allows powerful semantics, including conditional evaluation. For instance, consider the resolution of /apps/milc/CCFLAGS. The user profile refers back to the "headers" section in the system profile to detect whether a particular header file is present in the system. If so, the additional macro definition (-DHAVE_IEEEFP), meaningful to the application at the source file level, is added to the compiler flags.

8 Summary

In this paper, we address the problem of HPC software portability across heterogeneous high-end machines. Our research shows that many applications continue to use proprietary makefiles or shell scripts as their build technology. We propose a novel, lightweight approach to enhancing legacy application portability through the use of profiles which group and organize configuration parameters, facilitating their reuse across applications and platforms. With the Milc application use case, we demonstrate that migration to profiles is a relatively simple process, requiring only focused, one-time modifications in legacy build files. Since profiles encapsulate specific knowledge about the hardware platform, computer system, and application requirements, the application scientist that uses, ports, and tunes the application at hand is mostly relieved from the burden of the low-level expertise needed for achieving the optimal build. Nevertheless, users can still customize the build process according to their needs through user profiles.

Our future research will concentrate on refining the definition and structure of profiles. We also intend to perform experiments with LCF applications at ORNL. Additionally, we intend to evaluate the feasibility of our methodology for Autotools-based applications.

References

[1] The MILC Code (version: 6.20sep02). http://www.physics.utah.edu/\%7Edetar/milc/milcv6.html.

[2] The OpenScience Project. http://www.openscience.org/links/, 2006.

[3] Top500 lists. http://www.top500.org/lists, 2006.

[4] WRF v. 2.1.2. http://www.mmm.ucar.edu/wrf/users/download/get_source2.html, 2006.

[5] Computing Research Association. Report of workshop on the roadmap for the revitalization of high-end computing. Washington, D.C., June 2003. Available at http://www.cra.org/Activities/workshops/nitrd/.

[6] CPMD Consortium. CPMD ver 3.11.1. http://www.cpmd.org/cpmd_download.html.

[7] M. B. Doar. *Practical Development Environments, Chapter 5*. O'Reilly, Oct 2005.

[8] P. F. Dubois, G. K. Kumfert, and T. G. W. Epperly. Why Johnny can't build. *Computing in Science and Engineering*, 5(5):83–88, Sep/Oct 2003.

[9] J. L. Furlani and P. W. Osel. Environment modules project. http://modules.sourceforge.net/.

[10] Gordon research group at Iowa State University. The General Atomic and Molecular Electronic Structure System (GAMESS). http://www.msg.ameslab.gov/GAMESS/GAMESS.html, 2006.

[11] D. B. Kothe. National Leadership Computing Facility – Bringing Capability Computing to Science. CAMS High End Computing for Nuclear Fussion Science and Engineering Workshop, Feb 2006. http://www.inl.gov/cams/workshops/highendcomputing/d/national_leadership_computing_doug_kothe.pdf.

[12] G. K. Kumfert and T. G. W. Epperly. Software in the DOE: The hidden overhead of "the build". Technical Report UCRL-ID-147343, Lawrence Livermore National Laboratory, 2002.

[13] NERSC. Modules approach to software management. http://www.nersc.gov/nusers/resources/software/os/modules.php.

[14] Parallel Tools Platform, 2006. http://www.eclipse.org/ptp.

[15] G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor. *GNU Autoconf, Automake and Libtool*. New Riders publishing, 2000. Available at http://sources.redhat.com/autobook/.

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://dcl.mathcs.emory.edu/hwb/profile.xsd">
  <Group name="sys">
    <Group name="build">
      <Link name="currentSuite" ref="${/config/sys/build/suite}"/>
      <Group name="suites">
        <Group name="gnu">
          <Group name="CFLAGS" value="${OPT} ${INCLUDE} ${LIB} ${WARN}">
            <Group name="OPT" value="${/config/sys/build/CFLAGS/OPT}">
              <Item name="Optimize" value="-O2"/>
              <Item name="Debug" value="-O0 -g"/>
              <Item name="ARCH" value="-m64"/>
            </Group>
          </Group>
        </Group>
        <Group name="SunWS11.0">
          <Group name="CFLAGS" value="${OPT} ${INCLUDE} ${LIB} ${WARN}">
            <Group name="OPT" value="${/config/sys/build/CFLAGS/OPT}">
              <Item name="Optimize" value="-fast -fsimple=2 -fns -libmil -dalign"/>
              <Item name="Debug" value="-g -C"/>
              <Item name="ARCH" value="">
            </Group>
          </Group>
          <Group name="CC" value="${PATH}">
            <Item name="PATH" value="/development/WS11.0/SUNWspro/bin/cc"/>
          </Group>
        </Group>
      </Group>
      <Group name="headers">
        <Item name="IEEEFP" value="true"/>
      </Group>
    </Group>
  </Group>
  <Group name="config">
    <Group name="sys">
      <Group name="build">
        <Item name="suite" value="./suites/gnu"/>
        <Group name="CFLAGS">
          <Item name="OPT" value="${Optimize} ${ARCH}"/>
        </Group>
      </Group>
    </Group>
  </Group>
</Profile>
```

Figure 5. The system profile for SunOS (for clarity reasons non-relevant parts are skipped)

```
<Profile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://dcl.mathcs.emory.edu/hwb/profile.xsd">
  <Group name="apps">
    <Group name="milc">
      <Item name="CC" value="${/sys/build/currentSuite/CC}"/>
      <Item name="OCFLAGS" value="${/sys/build/currentSuite/CFLAGS}"/>
      <Item name="CODETYPE" value="${/config/application/DFAST}"/>
      <Group name="CCFLAGS" value="${headers/IEEEFP}">
        <Group name="headers">
          <Group name="IEEEFP" value="${${/sys/build/headers/IEEEFP}}">
            <Item name="true" value="-DHAVE_IEEEFP"/>
          </Group>
        </Group>
      </Gropu>
    </Group>
  </Group>
  <Group name="config">
    <Group name="apps">
      <Group name="milc">
        <Item name="DFAST" value="-DFAST"/>
      </Group>
    </Group>
  </Group>
</Profile>
```

Figure 6. The application profile for Milc v. 6.2

Biographies

Magdalena Sławińska is a postdoctoral fellow at the Mathematics and Computer Science Department at Emory University, Atlanta, U.S.A. She received her diploma in Computer Science from Gdańsk University of Technology, Poland, in 1999, and a Ph.D. in Computer Science from the same university, in 2005. Her research interests include high-performance computing, collaborative computing, and middlewares for resource sharing.

Jarosław Sławiński received his diploma in Computer Science from Gdańsk University of Technology, Poland, in 1999. He then went on to a few industry positions from a programmer to an IT department manager. Currently, he is a research associate at the Mathematics and Computer Science Department at Emory University, Atlanta, U.S.A. His research interests include new hardware solutions, and software engineering.

Dawid Kurzyniec received MS degree in Computer Science from AGH University in Kraków, Poland, in 2000. He is currently a Ph. D. candidate in the department of Math and Computer Science at Emory University, Atlanta. His research interests include heterogeneous distributed systems, concurrent processing, and security. He is the author of over 20 conference and journal publications related to distributed metacomputing.

Vaidy Sunderam is Dobbs Professor of Computer Science, and Chair of the Department of Mathematics and Computer Science at Emory University. His research interests are in parallel and distributed computing systems, software tools and architectures for metacomputing, high-performance message passing environments and infrastructures for collaborative computing. His prior and current research efforts focus on system architectures for heterogeneous computing middleware, collaboration technologies, and fault tolerant distributed systems. His work is funded by the National Science Foundation and the U.S. Department of Energy, and he has published over 140 articles in scholarly journals and refereed conferences. Professor Sunderam teaches courses in parallel processing, computer organization, distributed systems and software both at the beginning and advanced levels, and advises graduate theses in the area of computer systems. He is a recipient of the Emory Williams teaching award.