Document type: GWD-I
Category: Informational

Susanne M Balle, Hewlett Packard
Robert T. Hood, CSC-NASA Ames

Applications, Programming Models and Environments Area (AMPE)
User Program Development Tools for the Grid (UPDT)

March 17, 2004
Revised: March 17, 2004
Revised: July 9, 2004
Revised: July 14, 2004
Revised: September 13, 2004

GGF UPDT User Development Tools Survey

Status of this memo

This memo provides information to the Grid community regarding program development tools currently used by users developing programs on large scale systems and on the Grid. In addition, this memo provides information regarding what tools are needed in order to facilitate users' transition to the Grid.

Abstract:

Development tools such as debuggers and performance tuning tools are essential for users to understand why their application is not providing them the results they expected or why they are not achieving the expected performance. Cluster users are already accustomed to having these tools available to them. As users want to Grid-enable their applications or develop new applications they need help to avoid the pitfalls that Grid introduces. An adequate tools portfolio is essential in the program development cycle to maximize the programmer's productivity. In the Grid environment, very few tools are currently available to the user. The GGF UPDT RG surveyed users to determine what development tools users need to be able to write programs for the Grid.

The goals for this survey are (1) to understand the program development cycle on large scale systems, (2) to be able to determine where gaps exists in user program development tools and (3) to understand how users are currently handling program development on Grid—both on heterogeneous and on homogeneous systems. It is important for us to also get feedback from users about what tools are currently useful or what tools would be useful to have in the future. We are also interested in understanding the challenges encountered by developers when developing applications for large scale systems and the Grid

The main conclusions from the survey are that in order to get users to migrate their application to the Grid they need access to adequate Grid enabled building blocks and tools such as Grid enabled version of the libraries they use as building blocks in their current applications such as math libraries, Grid enabled middlewares and libraries, Grid-aware debuggers and Grid-aware performance tuning tools.

Author Contact Information:

Susanne M. Balle
Hewlett-Packard
MS ZKO2-3/N30
110 Spit Brook Road
Nashua, NH 03062
Susanne.balle@hp.com

Robert T. Hood
CSC--NASA Ames Research Center
M/S T27A-1
Moffett Field, CA 94035
rhood@nas.nasa.gov

Contents:

1 Introduction

Having a well thought-out tool portfolio to ease the development cycle on large scale systems and on the Grid is crucial if we want these architectures to be available to more users as well as ease the transition to the Grid. Development tools such as debuggers and performance tuning tools are essential for users to understand why their application is not providing them the results they expected or why they are not achieving the expected performance. Cluster users are already accustomed to having these tools available to them. As users want to Grid-enable their applications or develop new applications they need help to avoid the pitfalls that Grid introduces. Most users aren't currently experienced in developing programs for large scale homogeneous systems. Their task is further complicated since many Grids consist of heterogeneous systems. An adequate tools portfolio is essential in the program development cycle to maximize the programmer's productivity. We haven't been able to uncover any concrete data which could tell us if users of large scale systems and of Grids have adequate tools or what type of tools they would like to add to their current tool portfolio. In the Grid environment, very few tools are available to the user. The GGF UPDT RG surveyed users to determine what development tools they need to be able to write programs for the Grid.

The goals for this survey are (1) to understand the program development cycle on large scale systems, (2) to be able to determine where gaps exists in user program development tools and (3) to understand how users are currently handling program development on Grid—both on heterogeneous and on homogeneous systems. It is important for us to also get feedback from users about what tools are currently useful or what tools would be useful to have in the future. We are also interested in understanding the challenges encountered by developers when developing applications for large scale systems and the Grid

A pointer to the GGF User survey was distributed to several Grid mailing lists as well as directly to colleagues (including the Pittsburgh Supercomputing Center). Feedback was collected from February 2003 to October 2003. The following mailing lists were used:

- GGF UPDT RG (updt-rg@ggf.org)
- Globus discussion (developer-discuss@globus.org)
- MPICH G2 (mpich-g@globus.org)
- GGF Apps WG (apps-wg@gridforum.org)
- GGF APM RG (models-wg@gridforum.org)
- Etc.

We received 20 completed surveys in total. It is important to note that respondents could choose only to answer a subset of the questions. The first 10 respondents completed the "test" survey. We realized after having reviewed 10 respondents' answers that the survey needed clarifications in certain sections. Some respondents were confused about the definition of "design cycle" and when asked what tools they use, they listed debuggers, libraries, etc. We later reworded the survey in order to make this section clearer. We also reworked the survey by rearranging the order in which some of the questions were listed

to give it a better flow. The final survey was an on-line survey which made filling it out much faster and much easier.

The answers provided by the respondents have been compiled and are presented in this paper. It is important to understand that we are not doing explicit statistics but only rough estimations.

This paper is organized as follows: Section 2 gives a high-level description of the UPDT User survey, Section 3 summarizes the findings and conclusions from the survey and Section 4 presents the data from the completed surveys.

2 High-level description of the GGF UPDT User survey

The survey is divided into 7 parts:

1. User profile
2. Platforms
3. Design cycle
4. Program development cycle
5. Debugging cycle
6. Performance tuning cycle
7. Maintenance and administration

Along with the more technical aspects of the survey we wanted to understand the respondents' background. The user profile section and the platform section described below provide us with such information.

2.1 User profile

The user profile section deals with the respondents' research area, type of applications they develop, project size, and role within the project.

For more information see the survey in Appendix A

2.2 Platform

The platform section inquires the respondents about

(1) the system architectures they have at their disposal,
(2) details about their production systems,
(3) details about their production runs,
(4) details about their development systems,
(5) details about the OS's, available on their systems as well as
(6) if they have access to and use heterogeneous and or homogeneous systems

The Development cycle is described by the design cycle, the program development tools, the debugging cycle, and the performance tuning cycle sections of the survey.

2.3 Design cycle

In the design cycle section of the survey respondents were asked to describe the tools they use, to comment on their adequacy or inadequacy, if they are writing their applications from scratch or if they modify existing apps. We also asked the respondents to consider the major issues to take into account when writing application for Large Scale Systems (1000+ processors) and to prioritize the following: (1 being the most important and 6 being the least important)

(1) Scalability
(2) Robustness
(3) Fault tolerance
(4) Recovery mechanism
(5) Modularity
(6) Redesign of application or part of an application to get a more efficient load-balancing scheme.

The respondents were asked to prioritize the following considerations when designing applications for the Grid: (1 being the most important and 5 being the least important)

(1) Compatibility
(2) Flexibility
(3) Support for heterogeneous systems
(4) Security
(5) Single logon

In addition to the 5 factors mentioned above, respondents were also asked to add other factors to be considered.

For more information see the survey in Appendix A

2.4 Software development cycle

In the program development cycle segment of the survey, respondents are asked to describe the tools they use such as IDE, editors, languages, parallel programming paradigms, libraries, software packages, etc. They are also asked about their plans to have their application Grid-enabled as well as what Grid packages they use.

For more information see the survey in Appendix A

2.5 Debugging cycle

In the debugging cycle component of the survey, respondents are asked to describe the tools they use, to comment on their adequacy or inadequacy, to describe a typical debugging cycle, to list common debugging problems as well as to list new debugging challenges and problems cause by debugging their application on a Grid.

For more information see the survey in Appendix A

2.6 Performance tuning cycle

In the performance tuning section of the survey, respondents are asked to describe the tools they use, to comment on their adequacy or inadequacy, to describe a typical performance tuning cycle, to list common optimization problems as well as to list new challenges and problems cause by optimizing their application on a Grid.

For more information see the survey in Appendix A

2.7 Maintenance and administration

In the maintenance and administration segment of the survey, respondents are asked to describe the tools they use and to comment on their adequacy or inadequacy.

For more information see the survey in Appendix A

2.8 Other

This portion of the survey is meant for the respondent to give general feedback about the survey as well as to tell the author of the survey if they are interested in receiving the final report.

For more information see the survey in Appendix A

3 Conclusions

As mentioned earlier, the GGF UPDT survey was distributed and answers were compiled from February to October 2003. We received 20 completed surveys—10 respondents completed the "test" survey and 10 respondents completed the on-line final survey. Section 3.1 presents a summary of the results. In Section 3.2, we present the conclusions that we can make from the survey data.

3.1 GGF UPDT survey conclusions

We are interested in surveying users who develop applications for either large scale systems or for the Grid. Forty-five percent of the respondents answered that they are writing applications that target one thousand or more processes. Thirty percent of the respondents are expecting their application on less than one thousand nodes. We therefore feel that our sample selection is adequate.

User profile

The background of the respondents is very broad so we expected to collect unbiased survey samples. The respondents' work areas are CAD/CAM/CAE, Life Sciences, Physics, Earth Sciences, Performance evaluation, Compilers, Signal processing, Optic, Multimedia, General scientific computing, etc. Fifty percent of the respondents have a role as a technical project leader. The majority of the projects have 1 to 10 people.

We are confident that our sample size (=20) and our sample selection is adequate to achieve the goals of this survey.

See Section 4.1 for more details on the survey data collected.

Platforms

Sixty-two percent of the respondents use the same system for development and for production runs. In many cases, the type of systems depends on the project. As a general comment, developers use the machines that are available to them. The list of systems available to the respondents includes all the major vendors as well as Linux, various flavors of Unix, Windows, clusters and SMPs.

When asked if they are using heterogeneous systems, forty-five percent of the respondents answered affirmatively. We believe that some of the respondents misunderstood the question and though that the question meant if their application was portable -- can run on different system/OS/etc. One respondent specifically mentioned his participation in the SPACI (Southern Partnership for Advanced Computational Infrastructure) project. The SPACI system consists of: 3 HP Alphaserver SC (16 procs), 2 Origin 2000, IBM SP2 (16 procs), IBM SP3 (8 procs), Meiko CS2 (128 procs), 6 Beowulf clusters.

See Section 4.2 for more details on the survey data collected.

In order to understand the needs of the users to develop programs for large scale systems as well as for the Grid we need to go through the whole program development cycle.

Design cycle

Some respondents were confused about the definition of “design cycle” and when asked what tools they use, they listed debuggers, libraries, etc. We later reworded the survey in order to make this section clearer. By design phase we meant the steps developers takes or tools they use before they start programming.

Thirty percent of the respondents are creating their application from scratch whereas thirty percent of the respondents modify existing codes. The most popular design tools were Matlab, UML, HTML, and taking advantage of a number of building blocks e.g. energy evaluation procedures, local minimization procedures, etc. Thirty percent of the respondents found that the tools were adequate.

Scalability	1.18
Robustness	3.08
Fault Tolerance	3.36
Modularity	3.45
Recovery Mechanism	4.00
Redesign	4.27

Figure 1: Properties, from Figure "When designing applications for the large scale systems …"in Section 4.3, scaled according to their weighted ranking.

We asked the respondents to prioritize Scalability, Robustness, Fault tolerance, Recovery mechanism, Redesign, and Modularity when designing application for large scale systems (see Figures in Section 4.3). Figure 1 lists the properties according to their weighted ranking and illustrates clearly that Scalability came in as a clear number one. A low ranking means that this consideration was a high priority in the eyes of the respondents. The respondents felt that Scalability was a major issue to consider when developing applications for large scale systems. From the data we collected, it is difficult to conclude something about Recovery mechanism, Robustness, Modularity, Redesign, and Fault tolerance. Their rankings seem to indicate that the respondents gave them similar importance which indicate that these issues seem to be important to the respondents but not in any particular order.

We also asked the respondents to prioritize "Compatibility", "Flexibility", "Heterogeneity", "Security", and "Single Login". Figure 2 illustrates the properties, from Figure "When designing applications for the Grid "in Section 4.3, scaled according to their weighted ranking. The respondents prioritized "Heterogeneity", "Flexibility", "Compatibility", and "Security" as the most important considerations to take into account when developing applications for the Grid. We are surprised to see that "Single Login" is last. We feel that "Single Login" is one of the basic building blocks for a Grid to work well for scientists.

Flexibility	2.27
Heterogeneity	2.36
Compatibility	2.50
Security	2.50
Single Login	4.10

Figure 2: Properties, from Figure "When designing applications for the Grid …" in Section 4.3, scaled according to their weighted ranking.

Respondents mentioned the following other factors to take into consideration in a Grid environment:

- Software licensing (1),
- interoperability (2),
- resource management (1),
- performance (1),
- non-standard (1), and
- non-uniform (1)

See Section 4.3 for more details on the survey data collected.

Software development cycle

Users are writing distributed MPI programs, mixed MPI and threads programs as well as using other parallel programming paradigms such as HPF. They primarily use languages such as C/C++ and FORTRAN but Java, PERL, Python, Tcl/tlk are being used as well. 40% of the respondents mix languages in their application. Users take advantage of mathematical libraries such as Lapack and ScalaPack and others when possible.

Thirty-five percent of the respondents are looking into Grid-enabling their applications:

- 15% have already started,
- 10% expect to start within 6 months,
- 10% expect to start within 12 months, and
- 10% expect to Grid-enable their application in more than 12 months.

Of the respondents who are currently running on the Grid (35%) 20% use MPICH G2 and 25% use the Globus toolkit (GTK).
Of the respondents (25%) who use GTK -- 60% use GTK 2 and 40% use GTK 3.

See Section 4.4 for more details on the survey data collected.

Debugging cycle

The debugging cycle survey shows that print statements and gdb (see Appendix B) are the most used debugging tools. Totalview (see Appendix B) and "other" share third place. Respondents using specialized threading tools such Visual threads (see Appendix B) and/or specialized mpi-aware tools such as Visual MPI (see Appendix B) checked the "other" checkbox.

Fifty percent of the respondents found the tools available to them to be adequate. The respondents pointed out that the most valuable tools are flexible and simple. Having

access to MPI aware tools such as Totalview and Vampir/VampirTrace (see Appendix B) was noted as being very valuable as well.

The respondents mentioned that (1) Remote debugging, (2) Arithmetic, (3) Irreproducibility, and (4) Dynamic Issues are additional debugging issues created by a Grid environment.

The data shows that most developers are using standard debugging techniques to debug applications on large scale systems as well as on the Grid. These techniques and tools are adequate in a non-Grid environment. Developers circumvent not having Grid-aware debuggers by login in and attaching to local processes.

See Section 4.5 for more details on the survey data collected.

Performance tuning cycle

The performance tuning cycle survey lists the following tools as the most used performance tuning tools: gprof, compilers, and Vampir/VampirTrace (see Appendix B). It is important to note that the respondents didn't seem to agree on one or a couple of specific tools to use. These tools seem to be very system and vendor dependent. Fifty percent of the respondents found the tools available to them to be adequate.

It is important to note that only Vampir/VampirTrace is an MPI-aware tool therefore the data indicate that most users are only doing single processor optimization. Otherwise they maybe concentrating on the individual processor's performance and not on performing load-balancing or on distributed memory optimizations.

From the results presented in Section 4.6, we believe that many of the respondents are not using tools to optimize their parallel application but instead optimize each of the local processes. Some respondents did use system-wide tools such as Compaq's DCPI (see Appendix B) and oprofile (see Appendix B) which allow the developer to get a better understanding of the bottlenecks in their parallel applications.

When asked about additional performance tuning problems created by the Grid, respondents mentioned: (1) Variable latency, (2) Bandwidth, (3) CPU speed, (4) Everything!

The data shows that most developers are using standard optimization techniques to tune applications on large scale systems as well as on the Grid. Developers work around the problems created by running on the Grid by login in and tuning each process separately. This is not always possible but works when one can login to the individual nodes.

See Section 4.6 for more details on the survey data collected.

Maintenance and administration cycle

In the maintenance and administration cycle, CVS and RCS were deemed adequate. Other tools used were UNIX scripts and PBS. In many cases the person working on the code is also in charge of keeping the files.

See Section 4.7 for more details on the survey data collected.

3.2 Executive summary

After having processed the survey data, we are confident that our sample size (=20) and our sample selection is adequate to achieve the goals of this survey. We expect the conclusions to be unbiased.

The survey points out that, users are using all the systems they have at their disposal. They are no longer customizing code to a specific architecture — their applications have to be portable. We can therefore conclude that as the Grid becomes more popular, "heterogeneousness" is going to become an important factor to consider when developing Grid-enabling applications, middleware softwares and tools.

The most popular design cycle tools are Matlab, UML, HTML, and taking advantage of a number of already existing building blocks. The respondents were asked to rank specific factors to consider when designing applications. Scalability was rated the most important factor when designing applications for large scale systems. The rest of the factors – Robustness, Fault tolerance, Recovery mechanism, and Modularity – seem to be important but didn't get priorities assigned to them consistently.

The respondents prioritized "Heterogeneity", "Flexibility", "Compatibility", and "Security" as the most important considerations to take into account when developing applications for the Grid. We are surprised to see that "Single Login" is last. We feel that "Single Login" is one of the basic building blocks for a Grid to work well for scientists. In addition to the factors mentioned above, developers listed the following new factors to consider: (1) Software licensing, (2) Interoperability, (3) Resource Management, and (4) etc.

Thirty-five percent of the respondents had plans to develop a Grid-enabled application within a year. That number shows that the Grid in still new and not widely adopted. Unfortunately we are not able from the data gathered to understand why so few people are running or developing programs for the Grid. Our guess is that developing applications for the Grid is too hard and too time consuming. By surveying the landscape for Grid-aware tools, we realize that few robust tools are currently available.

The conclusion, from the software development cycle survey, is that respondents are using the common programming paradigms (see Lee et al. [1]) to program for the Grid. Users are writing distributed MPI programs, mixed MPI and threads programs as well as using other parallel programming paradigms such as HPF. They primarily use languages

such as C/C++ and FORTRAN but Java, PERL, Python, Tcl/tlk are being used as well. Users take advantage of mathematical libraries such as Lapack and ScalaPack and others when possible. The latter means that when researchers want to migrate to the Grid they need these libraries to be Grid-enabled.

From a debugging standpoint, most developers are using standard debugging techniques to debug applications on large scale systems as well as on the Grid. These techniques and tools are adequate in a non-Grid environment. Developers circumvent not having Grid-aware debuggers by login in and attaching to local processes. This is not always possible but works when one can login to the individual nodes. As the Grid gets more widely adopted, it is important to have debuggers that are Grid-aware since users will not be able to debug their processes by login in to the local node. Grid-aware debuggers such as P2D2 (Hood et al. [2]), Ygdrasil (Balle et al. [3]), etc., already exist and are being improved to meet Grid-aware requirements and users' needs.

The survey data shows that most developers are using standard optimization techniques to tune applications on large scale systems as well as on the Grid. The respondents didn't seem to agree on one or a couple of specific tools to use. These tools seem to be very system and vendor dependent. Fifty percent of the respondents found the tools available to them to be adequate. The respondents pointed out that they would like to see automatic performance tuning tools a la "ATLAS" (see Appendix B) as well as processor simulator tools. Developers work around the problems created by running on the Grid by login in and tuning each process separately. This is not always possible but works when one can login to the individual nodes. Grid-aware performance tuning tools such as Marmott [4], Dimemas [5], etc., are important for users that want to tune the performance of their application on the Grid.

If we want users to migrate from large scale clusters to the Grid, it is important that we make the tools they need and are accustomed to available on the Grid. Even though it is not currently obvious that scientists will run distributed MPI applications, for non-embarrassingly parallel jobs or for jobs without very little communication, across Grids we still need to make the Grid-enabling middlewares and libraries available to them so that they can, with minimal changes, make their application run on the Grid – assuming a working Grid. It is important that projects, such as the MPICH G2 project [6], Globus [7], GridLab [8], Cactus [9], PACX-MPI [4], etc., lead the way by providing the necessary middleware so that users can take advantage of the Grid without having to invest a huge amount of time and resources. As we get more experience with the Grid in general, it will be important that vendors integrate Grid-enabling capabilities for application development into their offerings.

In summary in order to get users to develop or migrate their application to the Grid they will need access to:

1. Grid enabled version of the libraries they use as building blocks in their current applications such as math libraries, etc.
2. Grid-enabling middlewares and libraries
3. Grid-aware debuggers

4. Grid-aware performance tuning tools

4 GGF UPDT survey data

In this Section, we present the questions and answers from the respondents.

As mentioned earlier the survey is divided into 7 parts:
1. User profile
2. Platforms
3. Design cycle
4. Program development cycle
5. Debugging cycle
6. Performance tuning cycle
7. Maintenance and administration

The results presented below are based on 20 respondents. It is important to remember that the respondents only had to answer the questions they wanted to answer. Nothing in the way the survey was setup was forcing them to fill out all the fields. The percentages presented below were done based on 20 respondents.

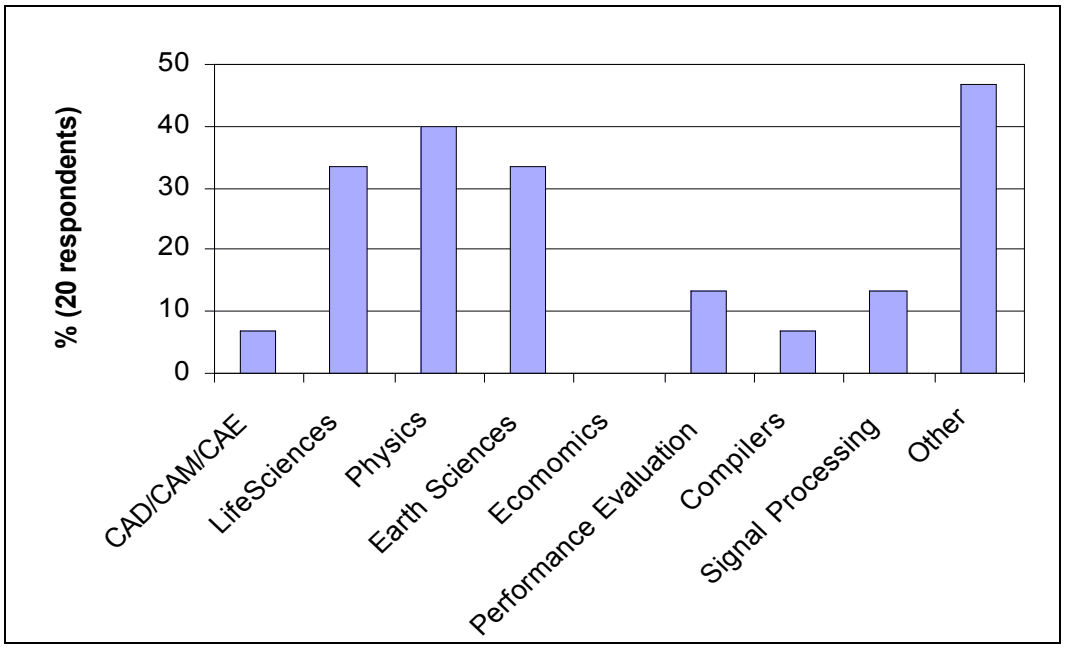
In the following sections you will see (#) next to a specific answer. (#) indicates the number of respondents that gave that answer.

4.1 User profile

1. Are the applications you are developing or have developed designed to run on Large Scale Systems (1000+ processors)?
   - 9 Yes 6 No

2. What area are you working in?

CAD /CAM /CAE	Life Sciences	Physics	Earth Sciendes	Economics	Performance Evaluation	Compilers	Signal Procesing	Others
1	5	6	5	0	2	1	2	7

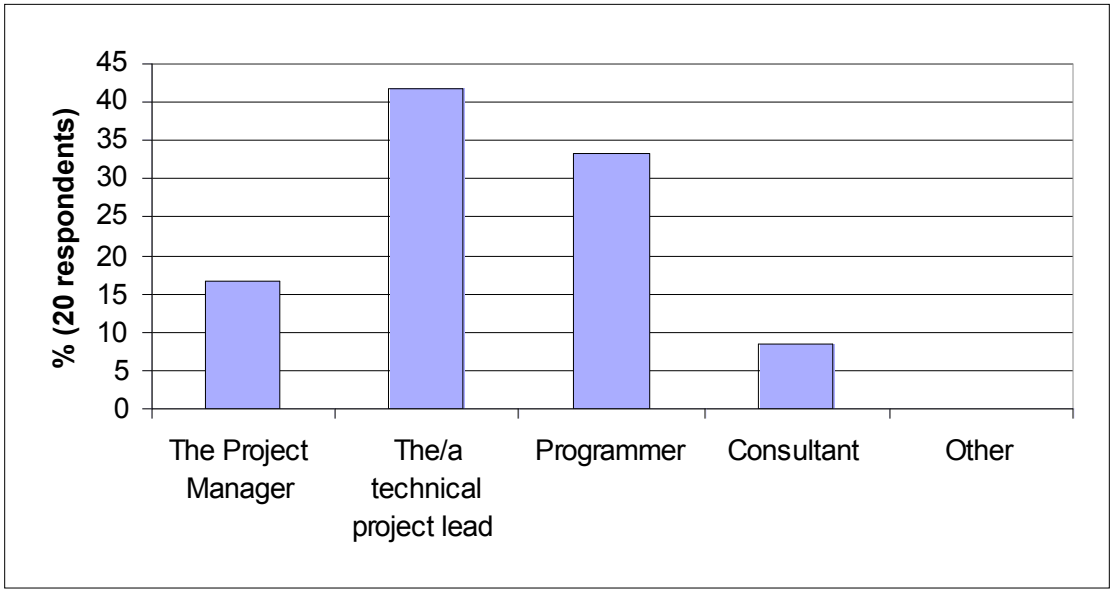


1. Others:
    a. General Scientific computing
    b. Optic
    c. Multimedia

3. Are you or your group the owner of the application you develop?
    - Yes: 5 No: 6

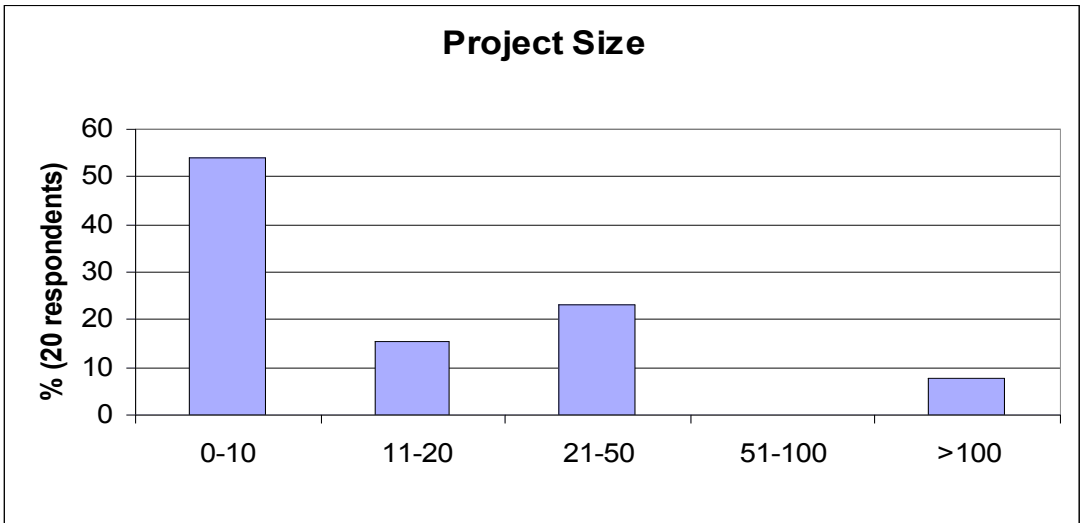
4. Your role within the project:

Project Manager	The/a technical project lead	Programmer	Consultant	Other
4	10	8	2	0



5. How many people are involved in the project?

0—10	11—20	21—50	51—100	> 100
7	2	3	0	1



4.2 Platforms

This part of the survey is carried out to better understand what type of machines, developers are using when writing their applications on as well as the size of the development machines and of the production machines. We were also interested in understanding what percentage of the respondents was using their production system for development as well.

1. What platforms do you have access to?
   - Users use any machine they can get access to
     1. HP AlphaServer SC (Lemieux @ PSC) (>3)
        - Production run 512, 100-600, …
     2. HP Itanium (1),
     3. IBM SP (2),
     4. Fujitsu VX (1),
     5. SGI Origin (2),
     6. Itanium Cluster (2),
     7. Power4 Cluster (1),
     8. IBM p690 (1),
     9. 32-processor Linux Cluster (3),
     10. 256-processor Pentium cluster (1),
     11. 384-processor Windows 2000 cluster (1)

2. Are you developing your programs/applications on the same machine as the one you will be running your production runs on?
    - Yes: 10 No: 6

    - 62.5% uses the same system for development and production
    - Varies with the project

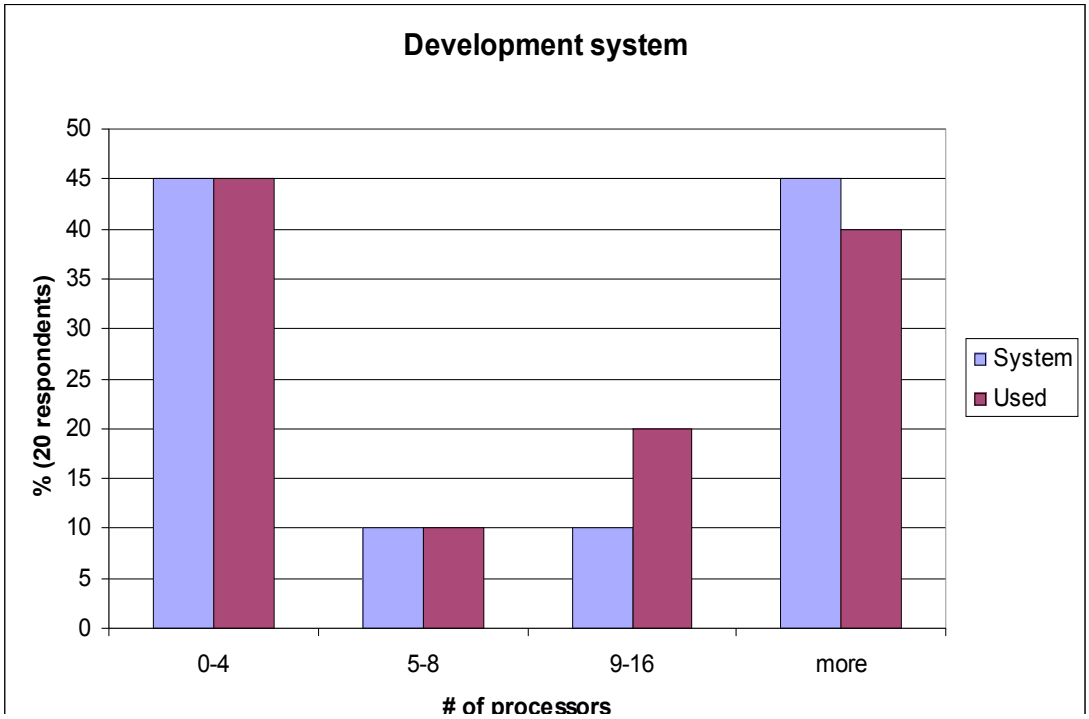
3. Are you running your program on heterogeneous systems?
    - Yes 7 No 4
        - The authors of the survey believe that some respondents thought this question meant if their apps run on different systems/OS/etc.
        - One specific example was given by one of the respondents which shows that this respondent is in fact using heterogeneous systems.
            a. SPACI (Southern Partnership for Advanced Computational Infrastructure)
                - 3 HP Alphaserver SC (16 procs), 2 Origin 2000, IBM SP2 (16 procs), IBM SP3 (8 procs), Meiko CS2 (128 procs), 6 Beowulf clusters

4. Development system
    - Number of processors in the system:

0—4	5—8	9—16	More
9	2	2	9

    - Number of processors used:

0—4	5—8	9—16	More
9	2	4	8



5. Production system

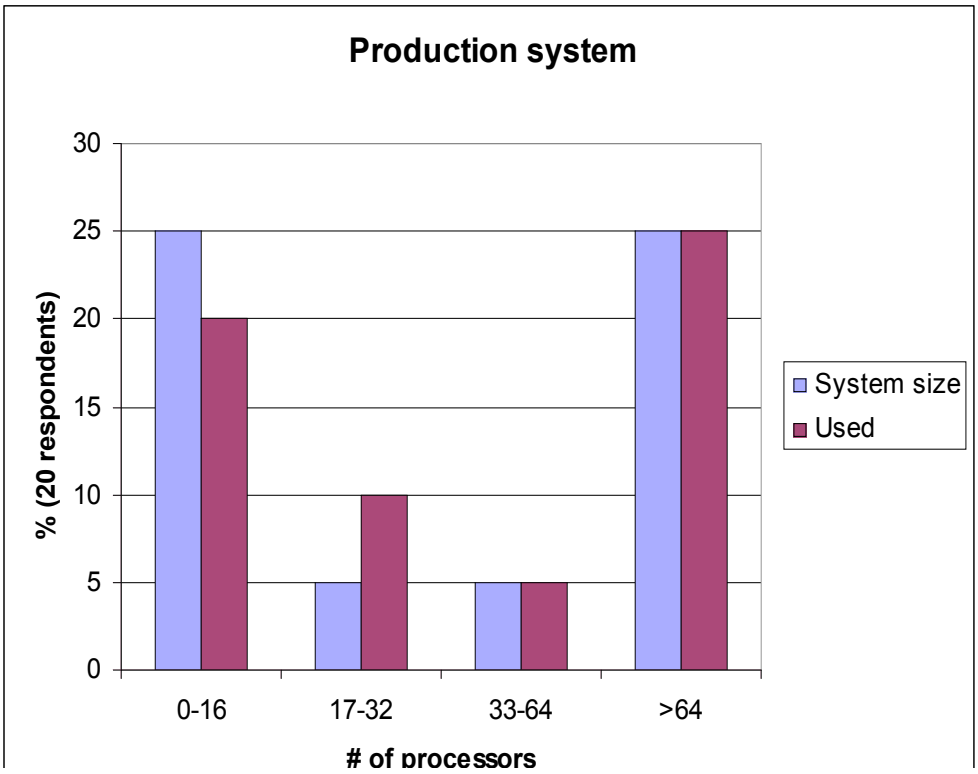
    1. What OS it it running?
        - UNIX version (11)
            - AIX (version 4.3.3), Tru64, HPUX, Solaris, IRIX (version 6.5)
        - Linux version (9)
            - (2.4.19, 2.4.20) (1)
            - Red Hat 6.0 (1), 7.2 (1)
        - Windows (5)

    2. Number of processors in the system

0—16	17—32	33—100	More
5	1	1	5

    3. Number of processors used:

0—16	17 – 32	33 – 100	More
4	2	1	5



4. Examples of production runs:

- Lemieux (HP AlphaServer SC @ PSC) (3)
  - Production runs
    - 4-600 processors
    - 512 processors
- 246-processor Pentium cluster (1)
  - Production runs 4-64 processors
- 384-processor Windows 200 cluster (1)
  - Production runs 20-100 processors
- 1392-processor UNIX system (1)
  - Additional details not available
  - Typical runs 1-256 processors

4.3 Design cycle

What tool(s) do you use?

- Visual Studio (1),
- Matlab (3),
- UML (1),
- Dev. Studio (1),
- HTML (1)
- Taking advantages of a number of "building blocks" blocks e.g. energy evaluation procedures, local minimization procedures, etc.

Are the tools adequate or inadequate? Why?

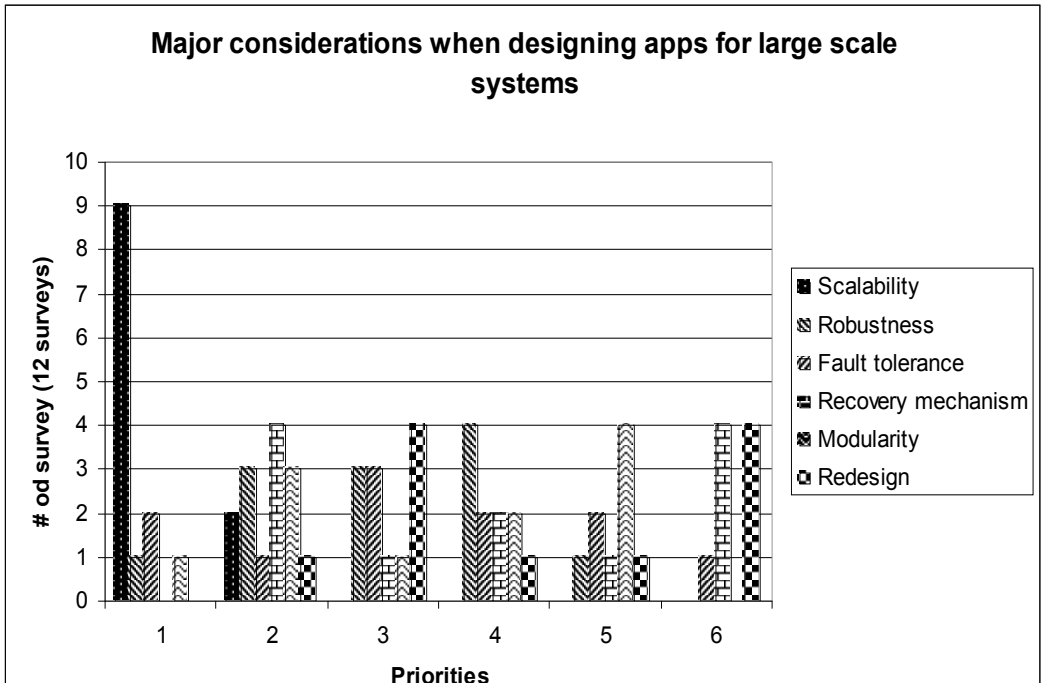
- Adequate (6)

Are you starting the design of your application from?

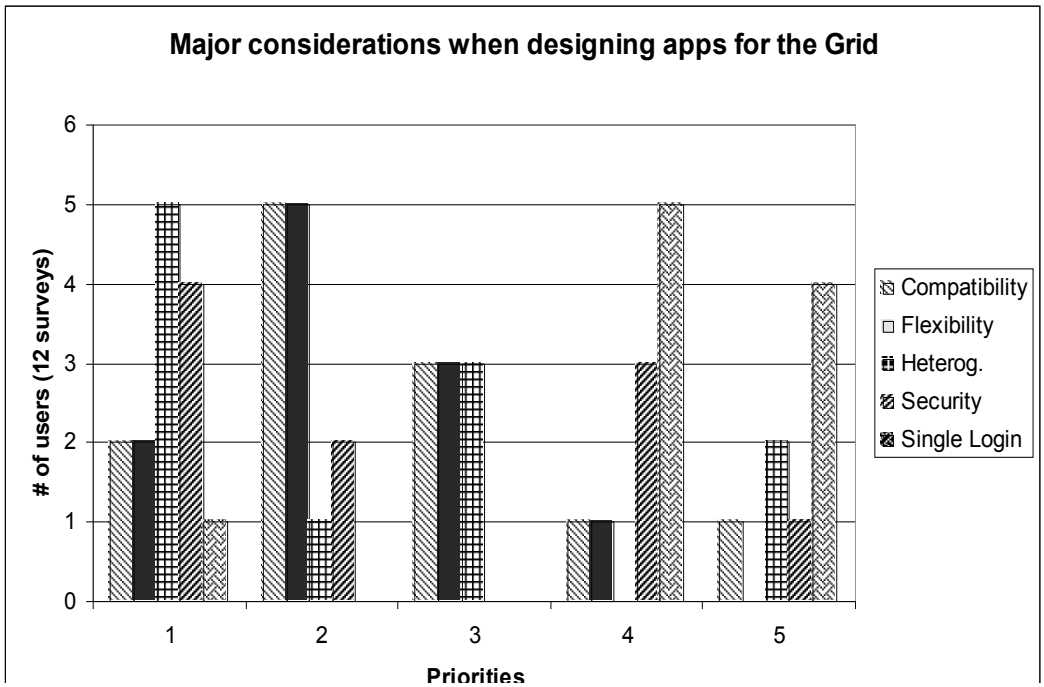
- Scratch (6)
- Modifying existing code (6)

Some respondents were confused about what "Design cycle" means.

What are the major considerations to take into account when writing application for Large Scale Systems (1000+ processors)? Please prioritize the following (1 = most important, 6 = least important)



When designing applications for the Grid what are the major considerations to take into account other than the ones listed above: (1 = most important, 6 = least important)



- Other factors to take into consideration in a Grid environment:
  - Software licensing (1),
  - interoperability (2),
  - resource management (1),
  - performance (1),
  - non-standard (1), and
  - non-uniform (1)

4.4 Software development cycle

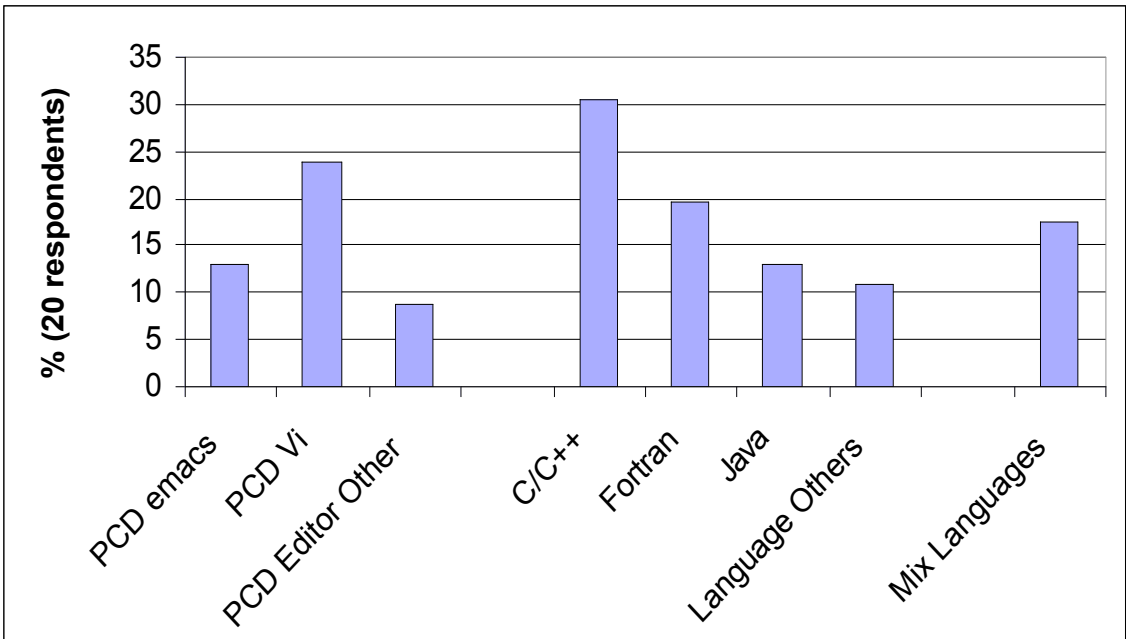
What integrated IDE are you using?

- Netbean (0)
- Jbuilder (0)
- Other (5)
  - Visual studio (2), JDE (sun) (1), Linux IDE (1), Websphere (1)

If no specific IDE is used what editor are you using?

Emacs	Vi	Other
6	11	4

- Other (editor)
  - Nedit, VisualSlic, kEdit, CMS, Xedit, gedit, joe, VC, ultraedit



What language(s) do you use? Do you mix programming languages?

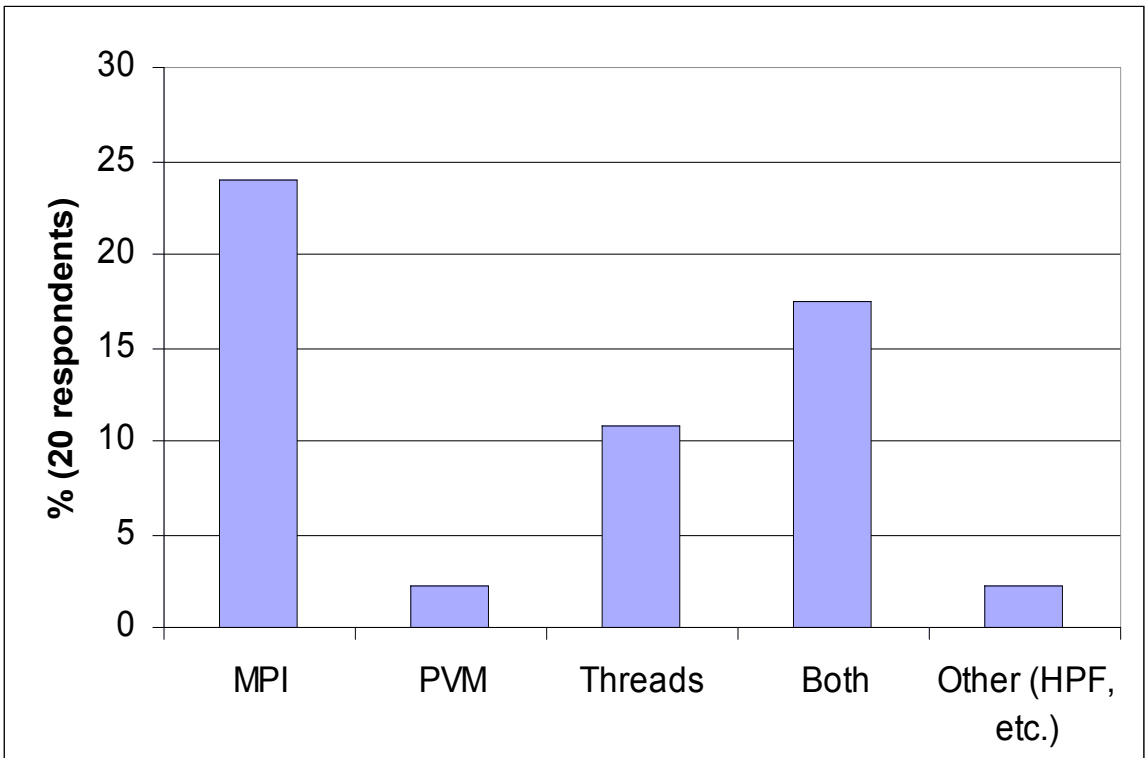
C/C++	Fortran	Java	Other	Mix Languages
14	9	6	5	8

- Other (languages)
  - Perl, Matlab, Cobol, assembler, Python, tcl/tkl, HPF
- Mixed languages
  - C Fortran (4), Java C++ (3)

What type of parallel programs do you usually write?

MPI	PVM	Threads	Both	Other
11	1	5	8	1

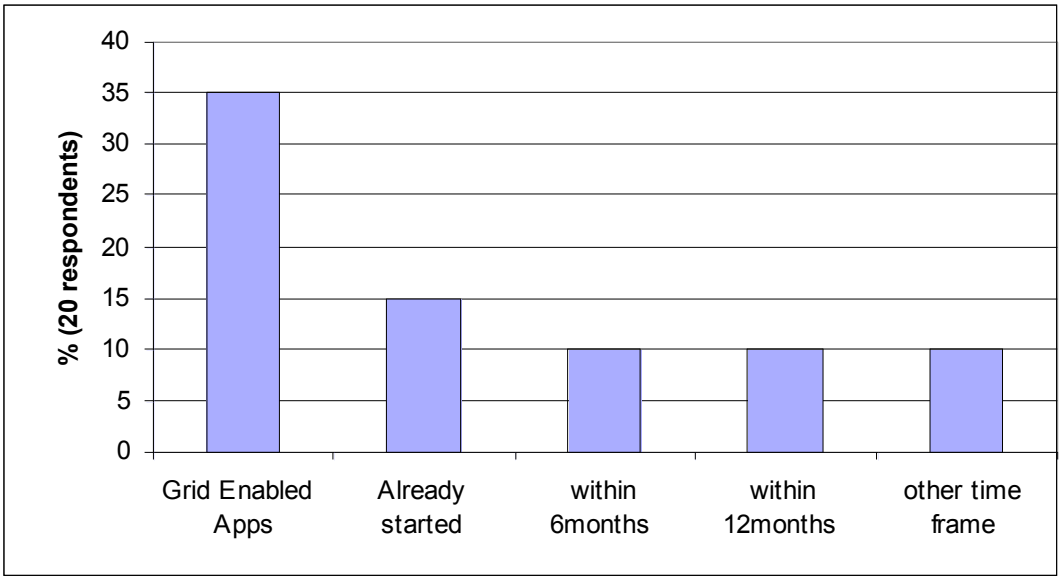
- Other (parallel languages)
  - HPF, Shmem



Are you looking onto Grid enabling your applications?

- Yes (7) No (3)

Already started	Within 6 months	Within 12 months	Other time frame
3	2	2	2



Software packages:

What software packages do you use?

1. Math Libraries

Lapack	Scalapack	PetSc	Vendor Math Libraries
6	3	0	6

- Other (Math Libraries)
  - WSMP, SuperLu, ARPACK/PARPACK, METIS/ParMETIS, FFTW

2. MPI

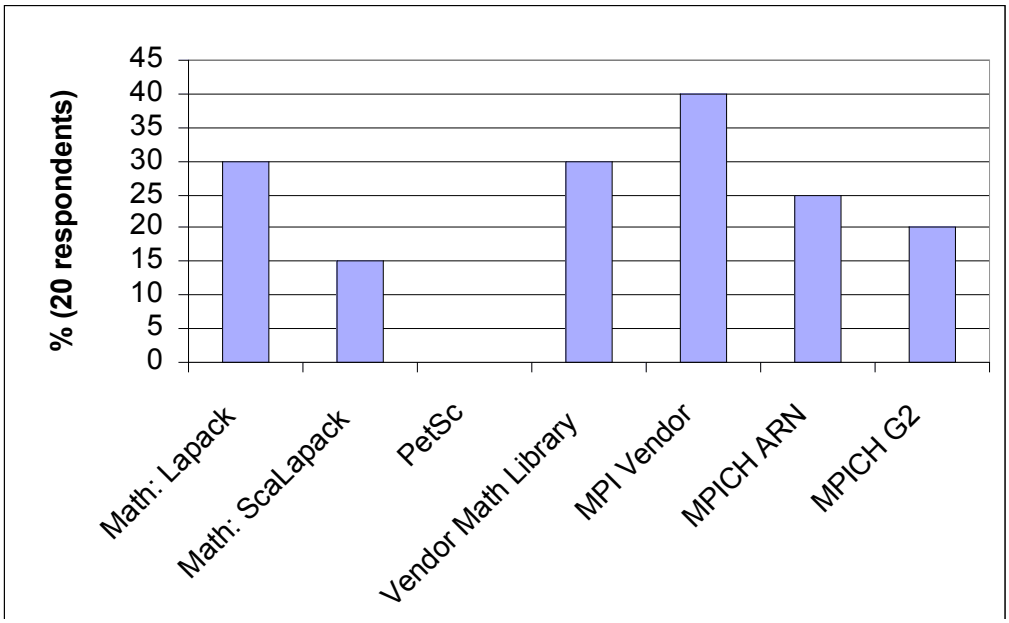
Vendor MPI	MPICH ANL	MPICH G2
8	5	4

- Other (MPI)
  - Customized MPI (1), pyMPI (1)

3. PVM (0)

4. Grid software
- GT2 (3), GT3 (2)

5. Other
- OpenMP,
- 2D/3D graphical programming library or tool, and
- Charm++/Converse



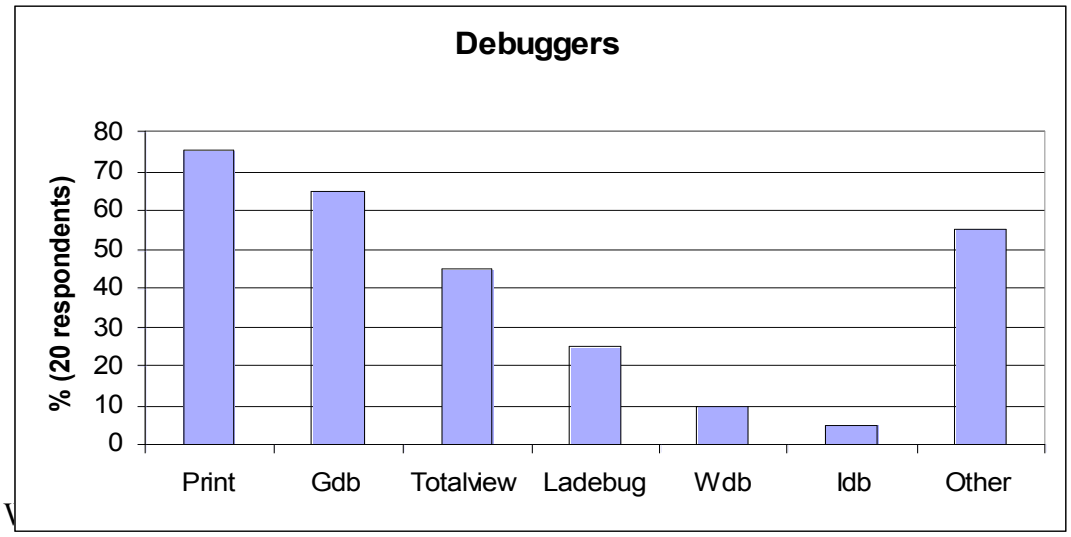
4.5 Debugging cycle

1. When debugging programs, which are targeted to run on thousands of processors do you use different tools for different number of processors?
   - Yes (4) No (8)

2. What tool(s) do you use?

Print statements	Gdb	Totalview	Compaq Ladebug	HP Wdb	Intel Idb	Other
15	13	9	5	2	1	11

   - Other: (debuggers)
     - Visual threads, Visual MPI, Visual Studio, DDD (1 proc), pdb



3. V
   a. Third degree ()
   b. Other ()
      i. valgrind
      ii. memcheck

4. Are the tools adequate?
   - Adequate (10) inadequate (4)

5. Comments:
   - "Good memory debugging tools tend to be expensive"
   - "Totalview too expensive"

6. Please describe a typical debugging session: (Several examples were described; they can all be summarized by the following)
   a. While (1)
      - { stare at code; edit code; build code; look at output;}

b. Debugging by comparing results for the parallel program with a serial program
c. Some types of problems we know that the algorithm should scale. If it doesn't it is a bug in the algorithm or in the hardware

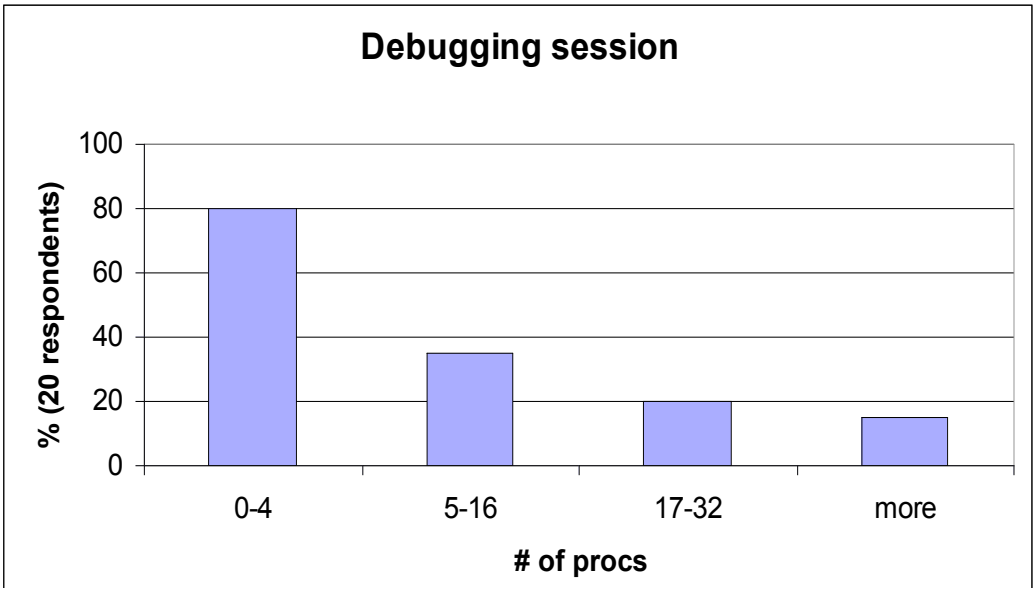
7. Which tools are the most valuable?
   - flexible and simple
   - MPI aware tools (such as Totalview and Vampir)

8. What data do you collect?
   - Variables
   - Callstacks

9. What additional tools/features would be on your wishlist?
   - Gprof with PAPI sampling

10. How many processes do you typically use when you are debugging your program?

0—4	5—16	17—32	More
16	7	4	3

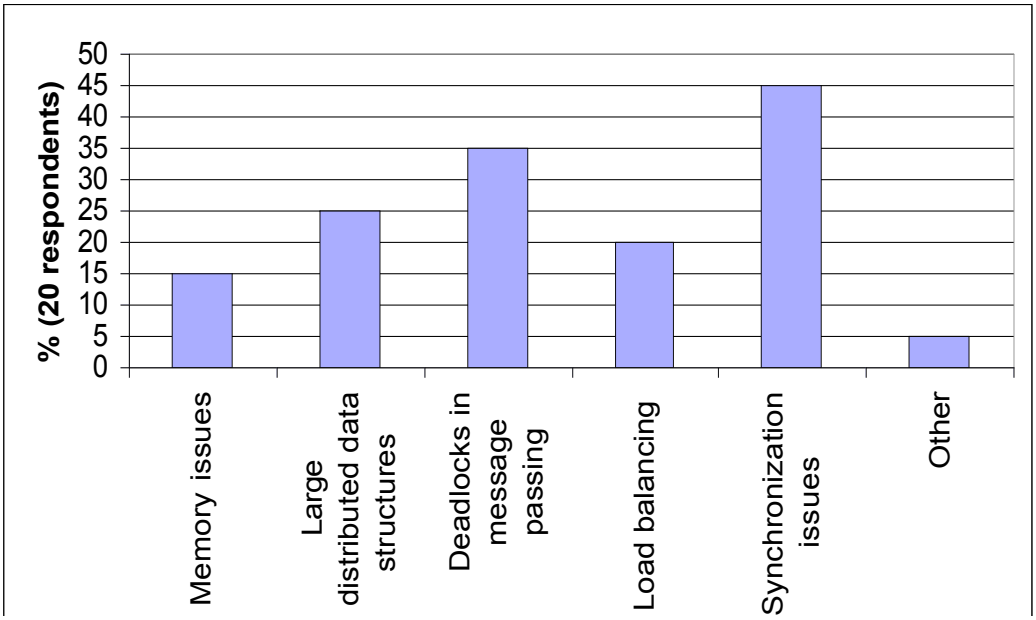


It is important to note that in the Figure "Debugging session" respondents were allow to check several of the categories. This explains why the total numbers of answers exceeds the total number of respondents.

11. What are the most common problems you debug on large-scale systems?

Memory	Large	Deadlock in	Load	Synchronization	Other

issues	distributed data structures	message passing	balancing	issues	
3	5	7	4	9	1



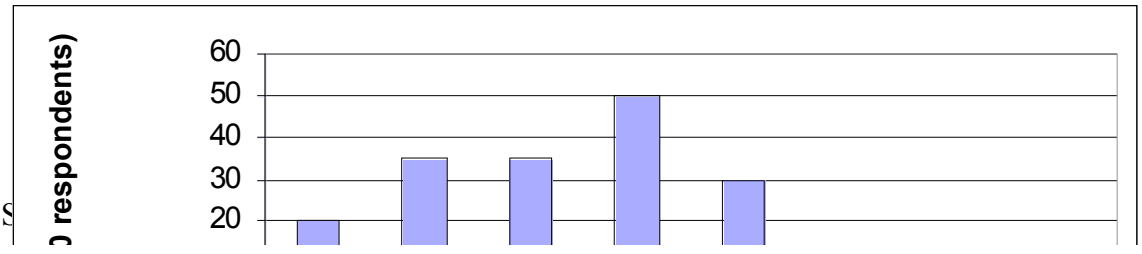
12. If your application is Grid-enabled how do you currently debug your program?
    a. Debugging by attaching to the local processes
    b. Debug the same way as we debug multi-process programs

13. What new problems does the Grid cause?
    a. Remote debugging
    b. Arithmetic
    c. Irreproducibility
    d. Dynamic issues

4.6 Performance tuning cycle

1. What tool(s) do you use?

Known benchmarks	Compilers with code annotations	Vampir/ VampirTrace	Gprof	PAPI	Paradyn	Vtune	Other
4	7	7	10	6	1	0	2



- Other: (performance tools)
  a. Visual Thread,
  b. Visual MPI,
  c. gperf,
  d. Xemsys toolkit,
  e. perfmon,
  f. WBTK,
  g. Hpm,
  h. DCPI,
  i. Atom,
  j. MPI tracers,
  k. XemSys
  l. Tuner,
  m. XemSys Analyzer,
  n. Intel's optimization report,
  o. Speedshop (SGI),
  p. memprof, and
  q. Paragraph

2. Are the tools adequate?
   - Adequate (10) Inadequate (2)

3. Comments:
   - Inadequate
     a. Vampir: "too expensive for large systems and too little development going on"
     b. Gprof: "Only wall clock, no PAPI information available, hard to interpret the output"
     c. "Very poor idea of what is limiting the processor"

   - Adequate

- "The Intel IA64 compiler will tell you how many cycles and what percentage of resources each loop in the code was taking"

Example of adequate tuning session:

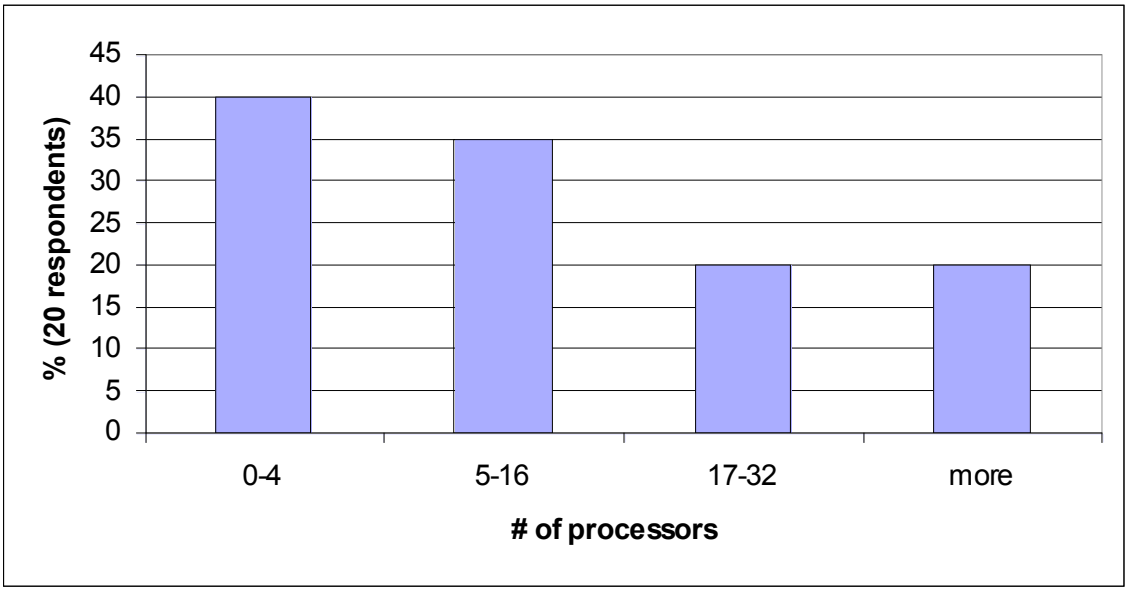
a. Performance specifications/benchmarks to set expectation
b. Compilers with code annotations to see how the compiler analyzes the code
c. Pixie to get operation counts
d. Compaq DCPI for low-impact profiling to discover where the time is spent
e. Compaq Atom tools for post-compile instrumentation to time the code

4. What date do you collect?
   a. hardware counters (4),
   b. CPU and wallclock (3),
   c. message queues (1),
   d. function overhead (1),
   e. message granularity (1),
   f. recv Wait time (1),
   g. Wall Clock timings for subroutines and critical sections (1),
   h. runtime (1)

5. What additional tools/features would be on your wishlist?
   a. Processor simulator which would show where time is spend (1),
   b. Automatic performance tuning a la "ATLAS" (1),
   c. Automatic performance monitoring a la "PAPI" (1),
   d. Gprof + PAPI hooks (1)

6. How many processes do you typically use when you are tuning your program?

0—4	5—16	17—32	More
8	7	4	4



7. If your application is Grid-enabled how do you currently debug your program?
   - Tune each machine separately

8. What new problems does the Grid cause?
   a. Variable latency
   b. Bandwidth
   c. CPU speed
   d. Everything!

4.7 Maintenance and administration

1. What tool(s) do you use?
   a. Unix scripts (1)
   b. CVS (4) adequate
   c. RCS (2) adequate
   d. PBS (1)

2. Comments:
   - In many cases the person working on the code keeps the files

4.8 Other

Comment from one for the respondents:
"It is indeed curious that scientists who presumably have been drilled in the methodology of rigorous" scientific inquiry for their particular discipline […] are often very casual when reporting on the performance of the computers they employ in their research." R. Hockney, "The art of Computer Benchmarking"

5 Acknowledgements

The authors thank Sergiu Sanielevici (Pittsburgh Supercomputing Center), Jie Song (Sun APSTC), Pierre Lagier (Fujitsu), and the users who completed the survey.

6 References

[1] C. Lee and al., A Grid Primer Programming Primer, GWD-1 (Informational), AMPE, GGF, August 2001 (http://www.eece.unm.edu/~apm/docs/APM_Primer_0801.pdf)
[2] R. Hoods and G. Jost, A Debugger for Computational Grid Applications, (http://www.nas.nasa.gov/Groups/Tools/Projects/P2D2/)
[3] S. M. Balle, J. Bishop. D. Lafrance-Linden, and H. Rifkin, Ygdrasil: Aggregator Network Toolkit for the Grid, May 2002
[4] R. Keller, B. Krammer, M.S. Müller, M.M. Resch, and E. Gabriel, MPI Development Tools and Applications for the Grid, , GGF8 Workshop on Grid Applications and Programming Tools, GGF8 , June 2003

[5] R.M. Badia, J. Labarta, J. Gimémez, and F. Escalé, Dimemas: Predicting MPI applicatins behavior in Grid environments, GGF8 Workshop on Grid Applications and Programming Tools, GGF8 , June 2003
[6] MPICH G2: a grid-enabled implementation of the MPI v1.1 standard (http://www.hpclab.niu.edu/mpi/)
[7] The Globus Toolkit: an open source software toolkit used for building grids (http://www-unix.globus.org/toolkit/)
[8] Cactus: an open source problem solving environment designed for scientists and engineers. (www.cactuscode.org)
[9] GridLab: A Grid Application Toolkit and Testbed (www.gridlab.org)

Intellectual Property Statement

Full Copyright Notice

IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

Appendix A: GGF UPDT survey
User Program Development Tools Survey for Large Scale Systems and the Grid

Prepared by: Susanne M. Balle (Susanne.Balle@hp.com)
Revised by:
Jie Song (SUN APSTC)
Pierre Lagier (Fujitsu)
Sergiu Sanielevici (Pittsburgh Super-computing Center)
Robert T Hood (CSC-NASA Ames)

Global Grid Forum UPDT RG
Last updated: 08/19/2003

Please send the completed survey back to Susanne.Balle@hp.com

---

User Profile
Platforms
Design Cycle
Program Development Cycle
Debugging Cycle
Performance Tuning Cycle
Maintenance and administration

User Profile:

Are the applications you are developing or have developed designed to run on Large Scale Systems (1000+ processors)?

1. Yes
2. No

What area are you working in?

1. MDCAD/ MCAM/MCAE
2. Life sciences
3. Physics
4. Earth Sciences
5. Economics
6. Performance Evaluation
7. Compilers

8. Signal Processing
9. Others
    a. Please specify _____

Are you or your group the owner of the application you develop?
1. Yes
2. No

If the program you develop is part of a larger project please specify your role.
1. Are you:
    a. The project manager
    b. The/a technical project lead of the project
    c. Programmer
    d. Other: please specify _____

2. How many people are involved in the project?
    a. 0-10
    b. 11-20
    c. 21-50
    d. 51-100
    e. 101+

Additional comments:

Platforms:

Systems:

Are you developing your programs/applications on the same machine as the one you will be running your production runs on?
1. Yes
2. No

Your development system:
1. What OS is it running?
    a. UNIX
        i. Please specify _____
    b. Linux
        i. Please specify _____
    c. Windows XP/2000/Me
    d. Other
        i. Please specify _____
2. How many processors does it have?
    a. 0—4
    b. 5--8

   c. more
3. How many processors do you use?
   a. 0—4
   b. 5--8
   c. more

If the development system is different for the production system

How many "production" systems do you have access to?

1. Your production system(s)
   a. What OS is it running?
      i. UNIX
         1. Please specify _____
      ii. Linux
         1. Please specify _____
      iii. Windows XP/2000/Me
      iv. Other
         1. Please specify _____
   d. How many processors does it have?
      i. 0—16
      ii. 17—32
      iii. 33--64
      iv. more
   e. How many processors do you use?
      i. 0—16
      ii. 17—32
      iii. 33-64
      iv. more

Are you running your program on **heterogeneous** systems?
- By this we mean if you have access to different systems/clusters with the different OS/processors/etc.
- Please specify systems characteristics (Vendor/ OS/ processor number)

Additional comments:

Design Cycle:

What tool(s) do you use?
- Are they adequate or inadequate?
  - Why?
- Which ones have you used in the last 3 months?
- Comments

Are you starting the design of you application
1. From scratch or
2. Are you modifying an existing program/application to fit your needs?

What are the major considerations to take into account when writing applications for Large Scale Systems (1000+ processors)?
Please prioritize the following: (1=most important 6=least important)

- Scalability
- Robustness
- Fault tolerance
- Recovery mechanism
- Modularity
- Redesign of application or part of an application to get a more efficient load-balancing scheme.

Please add to the list anything else you can think off as well as why this is important to you.

When designing applications for the Grid what are the major considerations to take into account other than the ones listed above: (1=most important 5=least important)

- Compatibility
- Flexibility
- Support for heterogeneous systems
- Security
- Single Logon

Please add to the list anything else you can think off as well as why this is important to you.

Additional comments:

Program Development Cycle:

What Integrated Development Environment (IDE) are you using?
1. NetBean
2. Jbuilder
3. Other: please specify _____

If no specific IDE is used what editor are you using?
1. Emacs
2. Vi
3. Other: please specify _____

Languages:

What language(s) do you use?
1. C/C++
2. Fortran
3. Java
4. Other: please specify _____

Do you mix programming languages? If so which ones and how important is it?

What type of programs do you usually write?
1. Message passing programs
2. Threaded programs
3. both (1) and (2)?
4. Other. Please specify _____

Is your application Grid-enabled? If yes what Grid software is it using?

Software packages:

What software packages do you use?
1. Math libraries? Please specify what library you are using.
    a. Lapack,
    b. SCALapack,
    c. Petsc,
    d. Vendor library,
    e. Other: please specify _____
2. MPI
    a. Vendor specific MPI
    b. MPICH from Argonne National Lab
    c. MPICH Globus flavor
    d. Other: please specify _____
3. PVM
4. Grid Software:
    a. Globus. Please specify version.
    b. Other: please specify _____
5. Other: please specify _____

Additional comments:

Debugging Cycle:

When debugging programs, which are targeted, to run on thousands of processors do you use different tools for different number of processors?

- What tool(s) do you use?
    1. Print statement
    2. gdb
    3. Totalview

4. other: please specify _____
- What tool are you using to debug memory problems?
    1. Third degree
    2. Other: please specify _____
- Are they
    1. Adequate
    2. Inadequate

- Which ones have you used in the last 3 months?
- How do you usually use them?
    - Please describe a typical debugging session
- Which tools are the most valuable?
- What do you like about the tool?
- What additional tools/features would be on your "wishlist"?
    - What features would make your life easier?

How many processes do you typically use when you are debugging your program:
1. 1-4
2. 5-16
3. 32-100
4. more

What are the most common problems you debug on large-scale systems?
1. Memory usage problems,
2. Large distributed data structures
3. Deadlocks in message passing
4. Load balancing
5. Synchronization issues
6. other: please specify _____

If your application is Grid-enabled how do you currently debug your program?
1. Yes
2. No

What new problems does the Grid cause?

Additional comments:

Performance Tuning Cycle:

- What tool(s) do you use?
    1. Known benchmarks to predict performance
        1. Compilers with code annotations feature
        2. Vampir/VampirTrace
        3. gprof

4. PAPI
5. Paradyn
6. timers such as MPI_WTIME
7. Vtune
8. other: please specify _____

- Are they
    1. adequate
    2. inadequate
    - Which ones have you used in the last 3 months?
    - How do you usually use them?
        - Please describe a typical performance tuning session
- What data do you collect?
- Which tools are the most valuable?
- What additional tools/features would be on your "wishlist"?
        - What features would make your life easier?

How many processes do you typically use when you are tuning your program:

1. 1-4
2. 5-16
3. 32-100
4. more

What new problems does the Grid cause?

If your application is Grid-enabled how do you currently tune your program?

Once you are "happy" with your program and are running in production mode how many processors are you usually running on:

1. 1-4
2. 5-16
3. 32-100
4. more

Additional comments:

Maintenance and administration

What tool(s) do you use?

- Are they adequate or inadequate?
    - Why?
- Which ones have you used in the last 3 months?
- Comments

Other

Any additional comments (about the survey , about relevant tools, etc.)

Please send the completed survey back to Susanne.Balle@hp.com

Appendix B: Glossary of tools from the GGF UDPT survey report

ARPACK/PARPACK: ARPACK is a collection of Fortran77 subroutines designed to solve large-scale eigenvalue problems. PARPACK is a parallel version of the ARPACK library. http://www.caam.rice.edu/~kristyn/parpack_home.html

ATLAS: The ATLAS (Automatically Tuned Linear Algebra Software) project is an ongoing research effort focusing on applying empirical techniques in order to provide portable performance. At present, it provides C and Fortran77 interfaces to a portably efficient BLAS implementation (www.netlib.org/blas)., as well as a few routines from LAPACK (www.netlib.org/lapack).

ATOM: ATOM uses the target application program, an instrumentation file and an analysis file to create a new executable, that when run collects analysis data for a wide variety of purposes. http://h30097.www3.hp.com/developerstoolkit/tools.html

Charm++: CHARM is a machine independent parallel programming system. Programs written using this system will run unchanged on MIMD machines with or without a shared memory. Charm++ is the C++-based parallel object oriented language having all features of Charm, which supports multiple inheritance, late bindings, and polymorphism. http://charm.cs.uiuc.edu/

Converse: Converse is an interoperable runtime system for parallel programming. http://charm.cs.uiuc.edu/research/converse/

CVS: CVS is the Concurrent Versions System, the dominant open-source network-transparent version control system. http://www.cvshome.org/

DCPI: In the Advanced Development Kit for Compaq Alpha platforms DCPI permits continuous low-overhead profiling of entire systems, including the kernel, user programs, drivers, and shared libraries. http://h30097.www3.hp.com/dcpi/

DDD: GNU DDD is a graphical front-end for command-line debuggers such as GDB, DBX, WDB, Ladebug, JDB, XDB, the Perl debugger, or the Python debugger. http://www.gnu.org/software/ddd/

FFTW: FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine and sine transforms, the DCT and DST). http://www.fftw.org/

Gdb: GNU debugger. http://sources.redhat.com/gdb/

Gprof: GNU profiler. http://www.gnu.org/manual/gprof-2.9.1/gprof.html

HPF: High Performance Fortran. http://www.crpc.rice.edu/HPFF/

Hpm: Hpm is the Hardware Performance Monitor toolkit.
http://www.sdsc.edu/SciApps/IBM_tools/hpm.html

HTML: HyperText Markup Language www.w3.org/MarkUp/

Idb: Intel Debugger. http://www.intel.com/software/products/compilers/fwin/

JBuilder: Borland(r) JBuilder(r) is the leading, cross-platform environment for building industrial-strength enterprise Java(tm) applications. **http://www.borland.com/jbuilder/**

JDE: Sun ONE Studio 5, Standard Edition. http://wwws.sun.com/software/sundev/jde/

MATLAB: MATLAB is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language. http://www.mathworks.com/

Memcheck: GNU memory debugging tool.
http://www.gnu.org/directory/devel/debug/memcheck.html

Memprof: GNU memprof is a tool for profiling memory usage and finding memory leaks. http://www.gnome.org/projects/memprof/

METIS/ParMetis: METIS is a family of programs for partitioning unstructured graphs and hypergraphs and computing fill-reducing orderings of sparse matrices. http://www-users.cs.umn.edu/~karypis/metis/

Ladebug: Ladebug from Hewlett Packard is a symbolic source-level debugger with a choice of command-line or graphical user interface.
http://h18000.www1.hp.com/products/software/ladebug/

Oprofile: OProfile is a system-wide profiler for Linux systems, capable of profiling all running code at low overhead. OProfile is released under the GNU GPL.
http://oprofile.sourceforge.net/news/

PAPI: PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. http://icl.cs.utk.edu/projects/papi/

Paradyn: Paradyn is a tool for measuring and analyzing the performance of sequential, parallel, and distributed programs. http://www.cs.wisc.edu/~paradyn/

Paragraph: Paragraph is a performance visualization tool for MPI.
http://www.csar.uiuc.edu/software/paragraph/

PBS: PBS is a flexible batch queuing system. http://www.openpbs.org/about.html

Pdb: Pdb is a portable binary database, http://pact.llnl.gov/

Perfmon: Performance Monitoring Tool.
http://www.hpl.hp.com/research/linux/perfmon/perfmon.php4

PyMPI: This package builds on traditional Python by enabling users to write distributed, parallel programs based on MPI message passing primitives.
http://sourceforge.net/projects/pympi

RCS: Revision Control System (RCS) manages multiple revisions of files.
http://www.gnu.org/software/rcs/rcs.html

SHMEM: The SHMEM library provides a shared-memory model for programming parallel computer systems. http://www-csag.ucsd.edu/projects/hpvm/doc/hpvmdoc_83.html#SEC90

Speedshop: an integrated package of performance tools
http://archive.ncsa.uiuc.edu/SCD/Perf/Tuning/Tips/speedshop.html

SuperLu: SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines.
http://crd.lbl.gov/~xiaoye/SuperLU/

Totalview: Debugger for complex code. http://www.etnus.com/Products/TotalView/

UML: The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software.
www.rational.com/uml/index.jsp

Valgrind: open-source memory debugger for x86-GNU/Linux.
http://developer.kde.org/~sewardj/

Vampir/VampirTrace: tool to visualize and analyse of MPI Programs.
http://www.pallas.com/e/products/vampir/index.htm

Visual Threads: is a diagnostic tool you use to analyze and refine multithreaded applications.
http://h21007.www2.hp.com/dspp/tech/tech_TechSoftwareDetailPage_IDX/1,1703,5062,00.html

Visual MPI: Visual MPI is a graphical tool for debugging and analyzing MPI applications. Visual MPI automatically diagnoses common problems such as deadlock,

MPI API usage errors, and thread-related errors.
http://h21007.www2.hp.com/dspp/files/unprotected/mpi/hpux/B6060-96014.pdf

Vtune: Performance Analyzer. http://www.intel.com/software/products/vtune/

WBTK: WBTK is a a set of benchmarking for memory performance.
http://www.epicea64.org/Articles/2002.10.13.wbtk.pdf

Wdb: The HP WDB debugger is an HP-supported implementation of the GDB debugger. It supports source-level debugging of object files written in HP C, HP aC++, Fortran 90, and FORTRAN77 on HP-UX Release 11.0 and later.
http://h21007.www2.hp.com/dspp/tech/tech_TechSoftwareDetailPage_IDX/1,1703,1662,00.html

Websphere: software platform for e-business. http://www.ibm.com/websphere

WSMP: Watson Sparse Matrix Package (Version 1.9.8, July 31, 2003) http://www-users.cs.umn.edu/~agupta/wsmp.html

Xemsys toolkit: Profiling tools.
http://www.caps-entreprise.com/en/index.php3?p=xnews