Autotasking can generally be described as the automatic distribution of loop iterations to multiple processors. You can allow Autotasking to be fully automatic, that is, not requiring your intervention, or you can interact directly with the Autotasking system to enhance performance. For more information on Autotasking, see the CF90 Commands and Directives Reference Manual or Optimizing Application Code on UNICOS Systems.

2.1 Autotasking Expert System

The Autotasking expert system, ATExpert, is a tool developed to measure and graphically display the Autotasking performance of a job run on an arbitrarily loaded UNICOS system. This tool can predict speedups on a dedicated system by using data collected from one run on a nondedicated system. It shows where a program is spending its time and whether those areas are executed serially or with multiple processors. It also serves as a guide for improving Autotasking performance.

ATExpert has the following characteristics and capabilities:

- Works with multitasked codes.
- Instrumentation incurs CPU overhead.
- Requires recompilation, reloading, and execution.
- Instrumentation focuses on timing each parallel region identified by the Autotasking system. Times for starting, stopping, and performing parallel work in a parallel region are measured. ATExpert also measures the time to perform work between parallel regions (called preceding serial time) and following the last parallel region (called ending serial time).

Instrumentation is required by the Cray C compiler, the Cray C++ compiler, and the CF90 compiler. After compiling and loading the instrumented file, the autotasked job is executed. At program termination, the ATExpert library routine produces a file (called atx.raw) that contains collected performance data. You can view this data by using the ATExpert postprocessing tool.

- Overhead caused by timing is kept low (10% to 20%) by carefully locating calls to the real-time clock function and avoiding large numbers of subroutine calls. This permits fairly accurate computation of various kinds of Autotasking overhead and allows ATExpert to make accurate projections of
speedups that use $N$ processors; $N$ can range from 4 through 164. However, overhead increases for programs that have small-granularity parallel regions.

- Information can be accessed by using an X Window System graphical interface, a simple ASCII interface, or a report. The easiest way to view data collected during the run of an autotasked job is to use the X Window System graphical interface of the ATExpert postprocessing tool. (See Section 2.3, page 25, for information about using the X Window System interface. For information about the ASCII interface, see Section 2.4, page 44. For information about the report interface, see Section 2.4.2, page 45.)

- ATExpert gives accurate performance measurements of a program, subroutines, parallel regions, and loops in a given application. From these measurements, a program can be examined to see where further performance gains can be made.

  Serial sections of the run can be scrutinized to find more potential parallelism. Autotasking overhead can also be detailed to determine where bottlenecks are occurring. ATExpert will, at each level of analysis, graphically show you how the program has performed.

- You also can use ATExpert with other tools, such as profview and flowview, to get an even more accurate feel for where you can gain additional performance.

For complete information on how ATExpert run-time data is collected and analyzed, see Section 2.6, page 50.

### 2.2 Getting Started with ATExpert

**Note:** You should run ATExpert as a background process, thereby freeing the command line for other uses by adding an ampersand ($\&$) at the end of the command line, as shown. If you already have invoked ATExpert, or any other application, as an active (foreground) process, you can move it to the background by pressing `CONTROL-Z` and entering `bg` at the shell prompt (not within the ATExpert window). This command sequence stops the process and then runs it in the background.

To start ATExpert using the X Window System interface, enter the following command:

```
atexpert &
```

You also can include a file name on the command line to invoke ATExpert and open a specified file, as follows:
atexpert -f file

The file name you include on the command line specifies the raw-format output file that ATExpert will process. If you do not specify a file name, the default is atx.raw. See Section 2.6.3, page 58, for more information about the atx.raw file.

The preceding command line starts ATExpert in its X Window System mode; that is, if the DISPLAY environment variable is set, ATExpert automatically tries to execute in X Window System mode. If the DISPLAY environment variable is not set, the default is ASCII interactive mode, in which you are prompted for additional options and data is displayed in a more tabular format. (See Section 2.4, page 44, for information about using the ASCII interface.)

You also can use ATExpert with a command-line option that generates reports for a file (-r). (See Section 2.2.1, page 23, for complete command-line options, and Section 2.4.2, page 45, for information about report generation.)

Figure 2, page 22, shows the main ATExpert window generated by the X Window System interface with the atx.demo file opened.
Guide to Parallel Vector Applications

Figure 2. Main ATExpert Window
For descriptions of screen areas, see Section 2.3.1, page 25, through Section 2.3.1.3, page 27.

**Note:** ATExpert displays shown on your screen may differ slightly from those in this document due to variances among window managers.

### 2.2.1 atexpert Command Line

Command options are as follows for the atexpert command:

```
```

- **-b** Shows the preceding serial time versus the parallel times for the given speedup chart.
- **-f file** Specifies the raw-format file with which ATExpert will work. Default is `atx.raw`.
- **-h** Writes help to `stdout`. Specify this option only when using the ASCII or report interface.
- **-L** Specifies that ATExpert ignore the `DISPLAY` environment variable (if set), and forces execution of an ASCII interactive session. You are prompted for additional options after the interactive session is started. Specify this option only when using the ASCII interface.
- **-m** Specifies that ATExpert should plot maximum speedups rather than plotting dedicated speedups. Default is to plot dedicated speedups.
- **-N cpus** Specifies the number of CPUs (maximum of 16) for which ATExpert will extrapolate data from the autotasked program.
- **-o** Directs ATExpert to give the overhead breakdown (in clock periods) for the number of CPUs set.
- **-p** Specifies page mode to set a page break at each screenful of information. Specify this option only when using the ASCII interface.
- **-r type** Generates reports for the file based on `type` specified, as follows. The `-f` and `-r` option
combination is required for noninteractive environment use. You may specify any combination of the p, s, r, and l types or the R type. Specify this option only when using the report interface.

```
p   Program summary
s   Subroutine summary
r   Parallel region summary
l   Loop summary
R   Report formatted and sent to stdout
```

-v Specifies verbose mode, which produces observations about the program, subroutines or function, parallel regions, and loops. Specify this option only when using the ASCII or report interface.

### 2.2.2 Instrumenting Your Code

To use ATExpert with CF90 code, use the following command line:

```
f90 -exe file.f90
```

If you have not included Autotasking directives in your code, you must add -03 to the command line to set the level of optimization.

To use ATExpert with Cray C++ or Cray C code, use the following command lines, respectively:

```
cc -h atexpert file.c
cc -h atexpert file.c
```

If you have not included Autotasking directives in your code, you must add -03 or -h task3 to the command line.

**Caution:** Multistreaming programs will not execute when instrumented for ATExpert.
2.3 Using the X Window System Interface

When using ATExpert with the X Window System interface, you see several different windows. (Figure 2, page 22, shows the main window.) Most windows are active; that is, selecting an item within them causes some action to occur. The action may be to update data in existing displays or to pop up a new display with more detailed information. For example, selecting a subroutine name in the Autotasked Subroutines window pops up a window with parallel region data.

Section 2.3.1, page 25, through Section 2.3.4, page 42, provide details about the following components of the ATExpert graphical display:

- Main window
- File menu
- Display menu
- Options menu

Explanations and screens shown are based on the X Window System interface; if you are running an ASCII session, see Section 2.4, page 44. See Section 1.7, page 9, for more information on the X Window System interface.

2.3.1 Main Window

The main ATExpert window contains several display areas: the parallel region monitor, the speedup display, and the information panel.

2.3.1.1 Parallel Region Monitor

In Figure 2, page 22, the top portion of the main window is called the parallel region monitor. This display shows parallel times, as well as the serial time preceding the parallel regions, for parallel regions. (See Section 2.5.1, page 47, for information on determining serial time.)

The vertical axis shows speedup, and the area of work displayed in each box corresponds to the amount of work done. Therefore, the more area a box has, the more impact that region has on the program. Also, boxes with higher speedups show performance gains; boxes (especially boxes with a large area) that show poor speedups potentially lower the entire program performance. The dotted line that runs the width of the screen shows average program speedup. It is significant to note the line in relation to each region as it highlights the relative importance that a particular region has to the program as a whole.
Parallel regions are shown in original execution order. You can scroll this window horizontally, in case the number of parallel regions exceeds the number of regions that can be meaningfully displayed in a small area. The parallel region monitor is an active display. Therefore, if you click on a particular parallel region, which will then be highlighted to indicate it has been selected, the other displays and windows will be updated with information about the selected region.

See Section 2.3.32, page 32, for more information about parallel regions.

2.3.1.2 Speedup Display

In Figure 2, page 22, the speedup display (located below the parallel region monitor) shows performance speedup curves. This display shows speedup curves for programs, subroutines or functions, parallel regions, or loops. A diagonal line and two curves are displayed and indicate the following information:

- The diagonal line represents linear speedup, which is the speedup a selected portion of the program would achieve if the program was 100% parallel and Autotasking incurred no overhead.

- The two curves show ideal and expected speedups if the number of CPUs used for the portion of the program currently selected is increased.

The top curve shows ideal speedup predicted by Amdahl’s Law. (See Section 2.6.2, for a discussion of Amdahl’s Law). It represents the speedups you would expect given the serial time but with no overhead associated with Autotasking. It shows the effect that serial time has on overall performance.

The second curve shows expected dedicated speedup, which is the performance improvement in wall-clock time you would expect to achieve on a dedicated system with the use of Autotasking. Dedicated speedups are simply the values of \( tu / tn \); \( tu \) is the unitasking time, and \( tn \) is the parallel time for \( n \) CPUs for the portion of the program displayed. When possible, measured values are used for \( tu \) and \( tn \). If a measured value for \( tn \) does not exist or seems to be unusually large or small, an extrapolation algorithm that simulates Autotasking is used to generate a value of \( tn \). When loop speedups are displayed, you can identify the use of measured times by the appearance of heavy dots on the speedup curve at those numbers of CPUs. Extrapolated times do not have dots displayed on the curve.

The diagonal line and two curves partition the display into three areas, as shown in Figure 2, page 22. The first area, up to the dedicated speedup curve, shows the effect of parallel execution or parallel work. You want this area as
large as possible. The second area between the two curves represents overhead. Finally, the top area above the Amdahl’s Law curve shows the effect of serial code on overall performance. The last two areas should be as small as possible for peak parallel performance.

Within this display, you can click on any CPU point and a window will pop up showing a breakdown of the various sources of overhead for the selected number of CPUs used for the portion of the program currently displayed in the window. See Section 2.6.2, for a complete description of overhead costs and an example of the Overheads pop-up window that shows the breakdown of overhead.

2.3.1.3 Information Panel

The information panel is located at the bottom of the main window and provides you with information about the status of the tool. If an error or warning message is generated, it will be displayed here and you will hear a beep.

Note: As you work within ATExpert, you will see other displays in addition to the main window. These displays are created in pop-up windows as a result of selecting a menu option or clicking in an active window.

2.3.2 File Menu

The File menu selections let you perform operations such as opening files, opening a demo file, and exiting ATExpert. Figure 3, page 27, shows the File menu.

![Figure 3. File Menu](image)
File menu selections are described in Section 2.3.2.1, page 28, through Section 2.3.2.3, page 29.

2.3.2.1 Open... Selection

The Open... selection lets you open (load) files to be used by ATExpert. You can also activate this selection by using the keyboard accelerator sequence CONTROL-O.

The Open... selection creates an input panel on which you enter the name of the file to be opened. The input panel accepts shell wildcard conventions. Figure 4, page 28, shows the input panel for opening files.

![Figure 4. ATExpert Load File Input Panel](image)

The input line, near the top of the panel, shows the path to the current directory (that is, the directory from which you invoked ATExpert). To remove this path name, click on the Clear button or position the cursor on the input line and use the BACKSPACE or DELETE keys to remove the current path name. You also can click on the directory name from which you want to load a file
from the list displayed under the Directories column of the ATExpert Load File input panel.

The file name entered on the input line specifies a raw-format (atx.raw) output file for the session and loads it into ATExpert. This option starts ATExpert analysis, fills the main window, and pops up the Autotasked Subroutines window, which lists subroutines in the program and shows their parallel and serial times.

In addition to entering a file name on the input line and pressing the RETURN key or the Load button, you can double click on a file name in the Files column of the input panel to load that file.

The panel has the following buttons:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Loads the files entered on the input line. Pressing RETURN is the same as clicking on the Open button. Separate multiple file names with a space.</td>
</tr>
<tr>
<td>Filter</td>
<td>Updates the directory or file list based on current file path and filter options.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears the input line.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the input panel.</td>
</tr>
</tbody>
</table>

2.3.2.2 Open demo file Selection

The Open demo file option opens a sample atx.raw file for you to view.

2.3.2.3 Quit Selection

The Quit selection displays a confirmation window on which you can confirm your decision to exit. You can also choose this selection by using the keyboard accelerator sequence CONTROL-Q.

The confirmation panel has the following buttons:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confirm</td>
<td>Exits the tool and returns you to the operating system. Pressing RETURN is the same as clicking on the Confirm button.</td>
</tr>
</tbody>
</table>
Cancel

Cancels the request to quit. Pressing ESCAPE is the same as clicking on the Cancel button.

To suppress display of the confirmation panel, add the following command line to your .Xdefaults file:

atexpert*quitConfirm: never

2.3.3 Display Menu

The Display menu selections let you show information about various aspects of your code. Figure 5, page 30, shows the Display menu. Menu selections listed above the line are used to retrieve windows you have closed; those listed below the line are used to open new windows.

![Display Menu]

Figure 5. Display Menu

Display menu selections are described in Section 2.3.3.1, page 31, through Section 2.3.3.7, page 41.
2.3.3.1 Subroutines... Selection

The Subroutines... selection retrieves the window that lists autotasked subroutines or functions, and provides information about parallel and serial time spent within them. This window is created by default when an atx.raw file is loaded. You should use this selection only if you have previously closed the window. You can also activate this selection by using the keyboard accelerator sequence CONTROL-S.

Figure 6, page 31, shows sample output from the Subroutines... option.

![Autotasked Subroutines Display](image)

Figure 6. Autotasked Subroutines Display

Subroutine or function names are shown on the left of the display. To the right of each subroutine/function name the following information is displayed in bar graph format:
### Information | Description
---|---
**Preceding Serial** | Indicates the time from when the previous parallel region ended to the time that the next parallel region begins. This means that part of the preceding serial time may come from the end of the last subroutine or function. It also may very likely include time from calls made to other untasking subroutines or functions before coming to this subroutine or function’s first parallel region.

**Parallel** | Indicates the sum of parallel times of all parallel regions in this subroutine or function.

**Ending Serial** | Indicates the time from the end of the last parallel region until the completion of the program.

The Autotasked Subroutines display is an active window; selecting a subroutine or function by clicking on it pops up another display that shows the selected subroutine’s parallel regions. (See Section 2.3.3.2, page 32, for information about parallel regions.)

To close the display, click on the Close button.

### 2.3.3.2 Parallel Regions... Selection

The Parallel Regions... selection retrieves a window that displays timing information about parallel regions of code. This window is displayed if you select a subroutine or function name from the Autotasked Subroutines display; or you can use this selection to reopen the display if you have previously closed it.

Figure 7, page 33, shows sample output from the Parallel Regions... selection.
2.3.3 Loops... Selection

The Loops... selection retrieves the window that displays timing information about loops in a selected region of code. Unlike the output from the Subroutines... and Parallel Regions... selections, however, output from the Loops... selection shows unitasked time versus autotasked time. This window is displayed if you select a parallel region from the Parallel Regions display; or you can use this selection to reopen the display if you have previously closed it. You can also activate this selection by using the keyboard accelerator sequence CONTROL-L.

Figure 8, page 34, shows sample output from the Loops... selection.
Figure 8. Loops Display

Loop names appear on the left side of the Loops display. Generally, loop names are the names from the original source file.

With the C and C++ compiler, loops do not have names, but are identified by line numbers. (See the Cray C/C++ Reference Manual, for more information.)

To close the display, click on the Close button.

2.3.3.4 Case statements... Selection

The Case statements... selection provides an additional level of detail about autotasked code. (CASE structure is indicated whenever a CASE label appears in the Loops display.) Selecting one of the CASE labels opens a window that details individual unitasked times for each CASE. This information lets you see how work is distributed across the cases.

Figure 9, page 35, shows sample output from the Case statements... selection.
2.3.3.5 Observations... Selection

The Observations... selection offers interpretations of ATExpert data and makes suggestions for improving performance. These observations also are available through the -v option of the atexpert command line.

Figure 10, page 36, shows sample output from the Observations... selection.
Guide to Parallel Vector Applications

Figure 10. Observations Display

The Observations display has the following buttons:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>Updates the speedup graph to reflect data for the currently selected program, subprogram, parallel region, or loop. You will use this button only if you have configured the Observations... selection to not update graphs and observations at the same time. (See Section 2.3.3.5.1, page 37.)</td>
</tr>
</tbody>
</table>
2.3.3.5.1 Configuring Observation Options

When you click on the Config button of the Observations display, the following pop-up input panel is displayed, which lets you tailor observations made about your code (refer to Figure 11, page 37):

![Figure 11. Observation Configuration Options Display](image)

The Observation Configuration Options display contains the following options:
Guide to Parallel Vector Applications

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction Parallel</td>
<td>Specifies the desired parallel fraction. Default is 0.80.</td>
</tr>
<tr>
<td>Threshold</td>
<td>Specifies the value used as a measure of minimum granularity. Default is 2000.</td>
</tr>
<tr>
<td>Fraction Overhead</td>
<td>Specifies the minimum acceptable Autotasking overhead fraction. Default is 0.10.</td>
</tr>
<tr>
<td>Number of CPUs</td>
<td>Specifies the number of CPUs for which Autotasking rules are being applied. Default is the number of CPUs on the system where the program executed.</td>
</tr>
<tr>
<td>Update graphs and observations together</td>
<td>When selected (highlighted), the speedup graph is updated automatically with code constants currently selected in the Observations window.</td>
</tr>
</tbody>
</table>

For additional information about the meaning of options available through the Observation Configuration Options display, see Section 2.3.3.5.2, page 38.

2.3.3.5.2 How Observations Are Determined

Observations are derived from the application of Autotasking rules, as follows:

- Observations are made with respect to the following formula:

  \[ T_p = ST + Tu/N + O \]

  This formula states that the parallel execution time of a program or any portion of it \((T_p)\) is a function of the following variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ST)</td>
<td>Serial time preceding a parallel region or following the last parallel region at the end of the program; that is, the amount of work outside parallel regions.</td>
</tr>
<tr>
<td>(Tp)</td>
<td>Wall-clock time at the given level when using (N) CPUs in parallel regions.</td>
</tr>
<tr>
<td>(Tu)</td>
<td>Total unitasked time of work inside parallel regions and loops.</td>
</tr>
</tbody>
</table>
\[ N \quad \text{Number of CPUs for which the Autotasking rules are being applied.} \]

\[ O \quad \text{Total overhead from all sources associated with parallel execution.} \]

- Observations try to bring about maximum parallel performance by maximizing the time of work inside parallel regions \((Tu)\) and minimizing serial time \((ST)\) and overhead \((O)\). ATExpert selects rules and makes observations based on the following formula:

\[
\frac{Tu}{(Tu + ST)} > FP
\]

\[
\frac{O}{(Tu/N)} < FO
\]

\(FP\) is the desired parallel fraction; default is 0.8. \(FO\) is the minimum acceptable Autotasking overhead fraction; default is 0.1.

\(FP\) and \(FO\) are configuration parameters. Increasing \(FP\) has the effect of identifying areas in a program in which \(ST\) is dominant. Decreasing \(FO\) has the effect of focusing more attention on overhead.

- The rule base is organized to focus primary attention on \(ST\) and secondary attention on \(O\) because you are most likely to improve program performance by making serial portions of the program parallel. Except for load imbalance, overhead costs are relatively fixed, and you can do little to change them other than to increase the granularity of parallel work inside parallel regions.

- Autotasking becomes effective only when some minimum amount of parallel work exists. This is called the granularity of parallel work. The rule base uses a threshold value as a measure of minimum granularity.

- All times and overheads are weighted by the appropriate sample counts or trip counts.

2.3.3.6 Report... Selection

The Report... selection displays, in text format, the data from which the graphical displays are derived. You can also activate this selection by using the keyboard accelerator sequence CONTROL-E.

Figure 12, page 40, shows a sample report.
Figure 12. Report Display

The report contains header information that describes the environment in which your program executed. Then, for each subroutine or function, the report lists all parallel regions and the loops within individual regions.

Overhead types listed in the report are explained in Section 2.6.2. Overhead types and other notations are indicated on the report, as follows:
<table>
<thead>
<tr>
<th>Type/notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>Indicates minimum times and counts of overhead and nonoverhead items.</td>
</tr>
<tr>
<td>ave</td>
<td>Indicates average times and counts of overhead and nonoverhead items.</td>
</tr>
<tr>
<td>max</td>
<td>Indicates maximum times and counts of overhead and nonoverhead items.</td>
</tr>
<tr>
<td>Nu</td>
<td>Indicates the number of times the unitasked version of code was executed.</td>
</tr>
<tr>
<td>Tu</td>
<td>Indicates the time required to execute the unitasked version of code.</td>
</tr>
<tr>
<td>N n / S n pairs</td>
<td>Indicates the numbers of samples taken and the speedup gained at a specified number of CPUs, as shown in the following example:</td>
</tr>
<tr>
<td></td>
<td>N8  S8</td>
</tr>
<tr>
<td></td>
<td>1    7.5</td>
</tr>
<tr>
<td></td>
<td>This pair indicates that one sample was taken at 8 CPUs and that the speedup at that number of CPUs was 7.5.</td>
</tr>
<tr>
<td>~ (tilde)</td>
<td>The tilde (~) symbol following any number indicates that an extrapolated number was used instead of a measured one.</td>
</tr>
</tbody>
</table>

The Report display has the following buttons:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>Saves the report to a user-specified file</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the display without doing anything</td>
</tr>
</tbody>
</table>

2.3.3.7 Source... Selection

The Source... selection lets you view a source file. When selected, this selection displays the Load File input panel on which you can specify the file you want to view. When the file is loaded, the first occurrence of any currently selected subroutine, parallel region, or loop is highlighted in the source file.

You cannot edit the Source display; however, you can use CONTROL-S to search for a string within the source file.
## 2.3.4 Options Menu

The Options menu lets you specify the format and content of the ATExpert display. Figure 13, page 42, shows the Options menu.

![Options Menu](image1)

**Figure 13. Options Menu**

Configuration specifications selected using the Configure data... selection override ATExpert defaults. You can also activate this selection by using the keyboard accelerator sequence CONTROL-C.

Figure 14, page 42, shows configuration options.

![Configuration Options Display](image2)

**Figure 14. Configuration Options Display**
The Configuration Options display contains the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of CPUs</td>
<td>Specifies the number of CPUs with which ATExpert analyzes your program. If you change this selection, you must reload the file. Default is the number of CPUs on the system in which the program executed.</td>
</tr>
<tr>
<td>Plot Numbers</td>
<td>Specifies whether to display numbers on the graph. Default is on.</td>
</tr>
<tr>
<td>Threshold Test</td>
<td>Instructs ATExpert to treat all loops that failed threshold tests as if they had enough work to pass the test. If you change this selection, you must reload the file. Default is on.</td>
</tr>
<tr>
<td>Filter Serial Time</td>
<td>Specifies that ATExpert ignore the effects of serial time and displays only the effects of parallel times. If you change this selection, you must reload the file. Default is off. The following filters are available:</td>
</tr>
<tr>
<td>Program</td>
<td>Filters by program serial time</td>
</tr>
<tr>
<td>Subroutine</td>
<td>Filters by subroutine or function serial time</td>
</tr>
<tr>
<td>Parallel Regions</td>
<td>Filters by parallel region serial time</td>
</tr>
<tr>
<td>Plot Speedup</td>
<td>Specifies type of lines to plot in the speedup display. The following lines are available:</td>
</tr>
<tr>
<td>Dedicated</td>
<td>Specifies plotting of dedicated line</td>
</tr>
<tr>
<td>Maximum</td>
<td>Specifies plotting of maximum line</td>
</tr>
<tr>
<td>Overhead</td>
<td>Specifies plotting of overhead line</td>
</tr>
</tbody>
</table>

To close the display, click on the Close button.
2.4 Using the ASCII Interface

If you do not use the X Window System, ATExpert provides a simplified ASCII interface. Although more limited than the X Window System version, this interface does present atx.raw file information. (See Section 2.2.1, page 23, for command-line options to specify the ASCII interface.)

2.4.1 Displaying Data

To use the ASCII interface to ATExpert, you first must instrument your program to generate timing data and produce an atx.raw file. (See Section 2.2.2, page 24, for information about compiler commands for instrumenting your code.) Then, you must choose the options you would like from the command line. In the following simple case, you could specify the following interactive command line:

atexpert -L -b -f atx.raw

This command line causes ATExpert to load the instrumentation file and to display a program performance curve, as shown in Section 2.4.1, page 44. (This is the same graph, that is shown in Figure 2, page 22, the X Window System interface.)

Welcome to the Autotasking(tm) Expert System.

Program summary: atx.raw

<table>
<thead>
<tr>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>CPU 4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Enter parallel section to analyze:

After ATExpert is executing, entering a question mark (?) at the prompt prints available interactive options. Available selections are as follows:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-b</td>
<td>Toggles the histogram breakdown. This gives more information about preceding serial and parallel times for subroutines/functions, parallel regions, and loops.</td>
</tr>
<tr>
<td>all</td>
<td>Plots all inner loops.</td>
</tr>
<tr>
<td>help</td>
<td>Prints the help file information.</td>
</tr>
<tr>
<td>list</td>
<td>Lists all parallel subroutines/functions, regions, and loops.</td>
</tr>
<tr>
<td>1[oops]</td>
<td>Plots all inner loops. You also can specify this option by using the letter 1.</td>
</tr>
<tr>
<td>p[rog[ram]]</td>
<td>Plots the program summary. You also can specify this option by using the abbreviations p or prog.</td>
</tr>
<tr>
<td>quit</td>
<td>Quits the ATExpert session.</td>
</tr>
<tr>
<td>r[eg[ions]]</td>
<td>Plots the summary of all parallel regions. You also can specify this option by using the abbreviations r or reg.</td>
</tr>
<tr>
<td>s[ub[routines]]</td>
<td>Plots the summary of all parallel subroutines/functions. You also can specify this option by using the abbreviations s or sub.</td>
</tr>
<tr>
<td>[subroutine name][region name][loop number]</td>
<td>Specifying subroutine name, region name, and loop number plots the specified subroutine or function, region, and/or loop.</td>
</tr>
<tr>
<td>-o</td>
<td>Toggles the overhead breakdown for the number of CPUs requested.</td>
</tr>
<tr>
<td>-m</td>
<td>Plots the maximum curve rather than the dedicated curve.</td>
</tr>
<tr>
<td>?</td>
<td>Prints a help screen.</td>
</tr>
</tbody>
</table>

2.4.2 Generating Reports

ATExpert provides several noninteractive reports. You can use the -r option to generate reports, which are then directed to stdout. This option is especially useful if you are confined to a noninteractive interface.
To get a simple report of program speedup, use the following ATExpert command:

```
atexpert -L -rp -f atx.raw
```

To get a report on program and subroutine speedups, overhead information for each, and have the report redirected to a file, use the following command:

```
atexpert -L -rps -o -f atx.raw > atx.report
```

If a full report of speedups for all loops, parallel regions, subroutines, and the program is needed, use the following command:

```
atexpert -L -rpsrl -f atx.raw
```

If you need a formatted version of the atx.raw file, use the following command:

```
atexpert -L -rR -f atx.raw
```

See Section 2.2.1, page 23, for a list of all report types available.

### 2.5 Interpreting ATExpert Graphs

You can make many interpretations from information that ATExpert displays. Section 2.5.1, page 47, through Section 2.5.7, page 49, discuss some initial interpretations you can make to help you get started using ATExpert. These are the first things most people look for when ATExpert displays program speedup curves. After you become familiar with these, you can begin to make your own, more detailed, interpretations and can interpret ATExpert’s wealth of performance data more comfortably.

The two key questions ATExpert is trying to answer for you are, "What improvement did using the Autotasking option give me?’’ and "What can I do to improve performance?’’

The first speedup chart you see (shown in Figure 2, page 22, as the Speedup Display) shows the expected speedup for your entire program. If speedup is significantly less than you expected, Section 2.5.1, page 47, through Section 2.5.7, page 49, offer some suggestions you can follow to determine why your program is not running as well in parallel as you might have expected.
2.5.1 Determining Serial Time

The area between the top (Amdahl’s Law) curve and the diagonal line in the speedup window represents serial time in your program. If this area is large, you will want to look at large areas of preceding serial time before parallel regions and see whether they can be autotasked by the use of directives.

2.5.2 Finding Large Serial Time

ATExpert provides the following three ways to find large serial time portions in your program:

1. The parallel region monitor at the top of the display area shows serial time as small, dark boxes with a height of 1. (The 100 is not displayed.) Click the mouse over a long box and ATExpert immediately changes the display to the corresponding parallel region that follows this serial code. Subroutine or function and parallel region names in the corresponding windows are highlighted. Click on the parallel region name and the associated loop identifiers are shown in a new loop window.

2. Select the Observations selection from the Display menu. Within the observation text is a section that lists the five top serial portions of your program.

3. Look for long bars of preceding or ending serial time in the Subroutine window and click on a subroutine or function name to display parallel regions in the subroutine or function. Click on the parallel region names with long bold bars to find the first loop in the parallel regions.

   Note: The code preceding the first loop of the highlighted parallel region is the serial code to investigate. Remember that serial code extends all the way back to the previous parallel region, which might be in another subroutine or function. Serial time includes all the code that is executed after the previous parallel region. It may include many subroutine or function calls and returns, plus I/O and other system or library calls.

2.5.3 Determining and Evaluating Key Performance Areas

To improve your program, you must find the key performance areas in your program and evaluate how effectively these key performance areas are being autotasked or run in parallel.

The dominant performance areas of your program are displayed in the parallel region monitor section at the top of the display area. The large shaded boxes
represents areas with large amounts of parallel work. The height of a box in the parallel region monitor section indicates parallel speedup. Therefore, you want parallel boxes to be tall. Long, flat parallel boxes point to key parallel regions that are not achieving a high speedup. In particular, you should investigate those boxes with a height that falls below the dashed line across the parallel region monitor section. Click the mouse on these boxes. Instantly, ATExpert relocates you to the subroutine or function and parallel region in question, with the corresponding names highlighted. Investigate these to see what is happening and what can be done to improve speedups.

2.5.4 Determining Parallel Overhead

The area between the top curve (Amdahl's Law) and the lower curve (dedicated speedup) represents overhead. If this area is large, overhead is hindering parallel performance. Click the mouse on the CPU area of the speedup display. A detailed breakdown of various sources of overhead will appear in a new window.

Section 2.6.2, gives descriptions of each type of overhead.

2.5.5 Interpreting Speedup Curves That Level Off or Plateau

A plateau on a speedup curve can mean several things. If the Amdahl's Law curve also plateaus near the dedicated speedup curve, serial time in your program is dominating performance. You should concentrate your optimization efforts on the large serial portions of your program.

However, if the Amdahl's Law curve shows acceptable speedups, this indicates a possible large load imbalance problem or parallel regions with small-granularity work. To determine which of these problems exist, pop up the Overhead display by clicking on one of the curved lines in the speedup display. Compare the size of load imbalance overhead to other sources of parallel overhead. (See Section 2.6.2, for information about overhead cost and Figure 15, page 53, to see a sample Overheads display.)

Sometimes you can resolve a load imbalance problem either by reorganizing a nested loop structure or by selecting an intermediate level loop to be autotasked. To improve a CASE structure, put larger cases first or break them into smaller cases.

If the problem seems to be small-granularity parallel work, investigate the corresponding key performance areas for ways to bring parallel execution to a higher level; either by bringing it to an outer level loop or to a higher level.
subroutine or function. Inline expansion of a routine also may generate higher-level parallel execution.

If it is difficult to remove a speedup curve plateau, you may want to consider setting the NCPUS environment variable to control the number of CPUs used in your program. This will make more effective use of the CPU resources on your system.

2.5.6 Determining Dominant Autotasked Subroutines

To determine the dominant subroutine or functions that were autotasked, look at the list of subroutine or function names in the Autotasked Subroutines display. These are all of the routines that contain parallel regions. If a dominant routine that you know about is not listed, it either was not executed or it was not autotasked.

The length of parallel bars in this window shows the amount of parallel work within each listed routine; therefore, longer light bars represent the dominant routines. If these bars do not correspond to your expectations, check the corresponding serial bars. If they are long, this indicates that some dominant loops in these routines are not being autotasked or could be autotasked better. Click on the routine names to bring up its parallel regions, and then click on parallel region names to locate the code by its loop identifiers.

2.5.7 Interpreting Bars and Messages

A vertical bar next to a subroutine or function name or a variable work loop message in the information panel section of the main window (as shown in Figure 2, page 22) both indicate variable work loops in your program. Briefly, a variable work loop is a parallel loop in which the time per concurrent iteration varies considerably from one execution of a parallel region to the next. Section 2.6.4, page 62, describes these loops in more detail.

ATEXpert usually handles variable work loops fairly well. However, situations do arise when variations in time are so large that ATEXpert’s algorithms break down and start producing bad speedups. When speedup curves for these loops are displayed, ATEXpert prints a warning message that indicates you should be wary of the speedup values.
2.6 Understanding ATExpert Concepts

Section 2.6.1, page 50, through Section 2.6.6, page 64, explain the following ATExpert concepts:

- Instrumentation
- Overhead costs
- at:raw output file
- Variable work loops
- Dedicated and maximum speedup curves
- Software anomalies

2.6.1 Instrumentation

Instrumentation records the number of unitasked, scalar iterations for each loop; tallies the number of concurrent iterations executed in each parallel execution of a loop; and stores other data associated with a parallel region or loop. (For information on how to instrument your code, see Section 2.2.2, page 24.)

Instrumentation, including calls to the real-time clock function, is added to your code to record the following times:

- Begin parallel region time (BP). This is the time just before the parallel region is entered. Only the master CPU records this time.
- Begin control structure time (BCS). This is the time just before the control logic for each parallel loop in a parallel region is entered. Each master and slave CPU records this time.
- Top of concurrent iteration loop (Top). This is the time when a CPU is about to begin its first concurrent iteration. Each master and slave CPU records this time. A top time is also recorded by the master before it executes each loop in the serial version of a parallel region’s loops.
- Bottom of concurrent iteration loop (Bottom). This is the time when a CPU finishes its last concurrent iteration. Each master and slave CPU records this time.
- End parallel region time (EP). This is the time just after the parallel region is ended. Only the master CPU records this time.
When an instrumented program terminates, data that has been gathered and summarized is written to a file named at.x.raw. See Section 2.6.3, page 58, for a description of the contents of this file. You can assign another name to the file, rather than at.x.raw. Before executing the instrumented program, set the ATXFILE environment variable, as follows:

Standard shell (sh) or Korn shell (ksh):

`% ATXFILE=filename; export ATXFILE`

C shell (csh):

`% setenv ATXFILE filename`

### 2.6.2 Overhead Cost

ATExpert defines overhead as the difference in speedup between that predicted by Amdahl’s Law for multiprocessing and that measured or projected from an actual Autotasking run. Amdahl’s Law for multiprocessing is shown in the following equation:

\[
S(N) = \frac{1}{f_s + \frac{f_p}{N}}
\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S(N))</td>
<td>Maximum expected speedup from multitasking</td>
</tr>
<tr>
<td>(N)</td>
<td>Number of processors available for parallel execution</td>
</tr>
<tr>
<td>(f_p)</td>
<td>Fraction of a program that can execute in parallel</td>
</tr>
<tr>
<td>(f_s)</td>
<td>Fraction of a program that is serial = 1 - (f_p)</td>
</tr>
</tbody>
</table>

The speedup from multitasking, \(S(N)\), is in terms of wall-clock time, not CPU time.

When ATExpert uses this formula, it computes the fraction of time spent in parallel code, \(f_p\), from exploited parallelism rather than parallelism that might actually exist in a program. The distinction is important, because a program might contain parallelism at a very high level, but Autotasking cannot exploit it
because it cannot perform dependence analysis across procedure boundaries. A user, however, can insert directives to exploit parallelism at a high level. ATExpert would reflect this by computing larger values of $S(N)$ for the program as a whole and for the subroutines and functions that contain the inserted directives.

ATExpert treats unexploited parallelism in a program as serial time. Although ATExpert can guide you to areas of your program with significant amounts of serial time, it cannot detect parallelism inside library routines (such as those in libsci) or in routines that were not compiled using ATExpert instrumentation. Thus, calls to library routines outside parallel regions are treated as serial time, even though parallel execution may occur inside the routines.

For a given portion of your program, $f_p$ is computed as the time to execute the parallel pieces of the program in unitasked mode, divided by the time to execute the entire portion in unitasked mode. Thus, for example, $f_p$ for a parallel loop is 1.0 because the entire portion of code is parallel. A subroutine or function, program, or parallel region has serially executed code outside of the loops. $f_p$ for these is less than 1.0.

For example, a subroutine contains a parallel region that requires 10,000 clock periods in unitasked mode. This subroutine also contains 5,000 clock periods of serial time outside the parallel region. For this example, $f_p = 0.67$ (10000/15000) and $S(8) = 2.42$. This is the ideal speedup, as predicted by Amdahl’s Law for this subroutine. Amdahl’s Law assumes no overhead is associated with multiprocessing.

Serial time is not considered overhead by the preceding definition. Instead, serial time merely affects the value of $f_p$ used in Amdahl’s Law. For ATExpert, $f_p$ reflects the amount of parallelism detected and exploited by the Autotasking software. It may not be an accurate measure of your program’s potential parallelism.

Overhead can occur from many sources within parallel regions. Each source contributes to a lower speedup than that predicted by Amdahl’s Law. The Overheads display in Figure 15, page 53, shows overhead totals for a sample program.

Note: To generate an Overheads display, click the mouse on the speedup display. For more information about determining parallel overhead, see Section 2.54, page 48.
The following sources of overhead are measured and displayed by ATExpert (screen abbreviations are shown in parentheses):

- Begin Parallel (bp) overhead: This overhead is the time that the master task requires to initialize parallel execution in a parallel region. During this time,
the master task is performing threshold tests, if necessary, testing whether Autotasking or microtasking is occurring at a higher level, setting up shared registers, and unparking any slave CPUs that may be waiting in the Autotasking library.

In terms of measured times, \( b_p \) is the time from the beginning of the parallel region to the first control structure.

- **Slave Arrival (sa) overhead** - Slave arrival time is the time it takes for the first slave CPU to reach the first control structure in the slave routine. It is measured from the time recorded by the master at the end of its begin parallel code.

- **Wait Slave (ws) overhead** - This overhead is the time that the master task must wait for slave CPUs to return to the Autotasking library before it can resume serial execution outside the parallel region. It varies depending on whether the master or a slave process takes the last concurrent iteration.

- **Begin Control (bc) structure overhead** - This overhead is the time it takes to compute the number of concurrent iterations in the next parallel loop, to establish synchronization parameters for parallel execution, and to detect whether a master or slave process has any work to do in the parallel loop. In this sense, work means concurrent iterations.

In terms of measured times, \( b_c \) is the time from the beginning of the control structure to the top of the concurrent iteration loop.

- **Convoy Time (cv) overhead** - This overhead is the time required to get all remaining slave CPUs (after the first) to the first control structure in the slave routine.

Adding \( sa \) and \( cv \) overhead represents the latency in getting all slave CPUs into the slave routine and ready to take a concurrent iteration. If this overhead is too high, it is possible for all concurrent iterations to have been taken by the time the last slave CPUs arrive.

- **Iteration Overhead (io)** - Iteration overhead is the additional cost incurred during the execution of concurrent iterations. This includes the time to obtain a concurrent iteration and to test whether the last iteration has already been started. Usually, the time to execute this scalar code is covered in the hardware by the completion of the final vector instructions in an inner loop. However, instances (such as reduction loops) occur when this is not true.

Iteration overhead also may include hardware delays from the use of multiple CPUs, such as additional memory bank conflicts or shared-register contention. These are difficult to measure directly. The change in iteration
overhead as more CPUs execute in a parallel loop is used as an estimate for these.

- Loop Sync (ls) overhead - Loop synchronization overhead is the time required for all participating CPUs to finish their last concurrent iteration after the first CPU finishes its final concurrent iteration. This synchronization ensures that all of the results from the previous parallel loop are in memory before starting the next parallel loop.

ATExpert tries to decompose loop synchronization overhead into components associated with load imbalance, slave arrival latency, and convoy time because these more closely reflect the cause of loop synchronization delays. Thus, ATExpert seldom displays a nonzero value for ls overhead.

The ls overhead is a measure of the quality of the ATExpert algorithm to decompose Autotasking overhead into its separate components. Because this algorithm is based on measured times, ls sometimes reflects noisy times rather than an incorrect algorithm. In other situations, ls can be nonzero because of the reasons listed for "other" overhead (for example, a negative value of ls can result when the compiler generates higher performing parallel code than the corresponding serial code in the three-code model).

- Load Imbalance (li) overhead - Load imbalance occurs when the work (that is, concurrent iterations) cannot be distributed evenly across the participating CPUs. If no other sources of overhead exist, ATExpert computes li overhead as the difference between a perfect loop time partitioning for N CPUs and the discrete distribution of the given number of concurrent iterations (Ilers) across N CPUs. The formula that ATExpert uses is as follows:

\[
li = \frac{tu}{Ilers} \times \text{ceil}(Ilers / N) - \frac{tu}{N}
\]

The tu variable is the unitasking time for the loop. (The formula is a little more complex for vector concurrent loops or CASE structures because the time per concurrent iteration or case varies.) The term \(\text{ceil}(Ilers / N)\) represents the maximum number of iterations that at least one CPU must take. Basically, this formula says that whenever the number of iterations is not exactly divisible by the number of CPUs, a load imbalance exists.
Suppose a loop that contains 16 iterations executes in time $tu$. The perfect distribution of parallel time for $N$ CPUs is for each CPU to execute for time $tu / N$. For values of $N$ equal to 1, 2, 4, 8, and 16, the iterations are exactly divisible by $N$, and each CPU will execute in time $tu / N$ (if no other overhead exists). However, for $N$ equal to any other value, 16 iterations cannot be partitioned evenly. If $N$ equals 5, for example, one CPU must take four iterations and the rest will take three iterations. This creates a load imbalance of the following:

$$li = \frac{tu}{16} \times 4 - \frac{tu}{5} = \frac{tu}{20}$$

Thus, load imbalance is contributing a 5% overhead when executing this loop with five CPUs.

Load imbalance overhead is sometimes hidden by the effects of $sa$ and $cv$ sources of overhead. ATEExpert displays the remaining influence of load imbalance, after $sa$ and $cv$ have been considered.

- **Inter Loop (il)** overhead - Inter Loop time is the time required to reach the top of the next control structure after all CPUs have finished their last iteration of the previous control structure. It applies only when two or more parallel loops exist within a parallel region or one parallel loop is executed multiple times in a parallel region.

- **Wait Send (wt)** overhead - A quantity that reflects the wall-clock time impact that wait/send has on the region of code being viewed. For example, if a lot of work is surrounding the wait/send, processors may be scattered throughout and the wait/send may have little impact.

- **Other overhead** - Other overhead is a measure of the inaccuracy of the ATEExpert instrumentation and algorithms. It results when the ATEExpert extrapolation algorithm produces an estimated parallel time that is smaller than the measured time for a given number of CPUs. This algorithm uses measured values of times and overhead when possible. The following are known to be causes of what ATEExpert shows as "other overhead."

  - "Noise" introduced in the timings because of running on a loaded system, with unrelated jobs running in other CPUs. Memory bank conflicts from these other jobs affect both the unitasking and parallel execution times of loops.
- The ATEexpert extrapolation algorithm contains some simplifying assumptions that are sometimes not valid. As a result, the model sometimes produces parallel times that are more ideal than real.

- The ATEexpert library routine summarizes the instrumentation data gathered during the execution of a parallel region. Only minimum, maximum, and average times are eventually written into the atx.raw file; thereafter, ATEexpert generally uses only the minimum times. This occasionally introduces variations between unitasking times and parallel times with different numbers of CPUs. ATEexpert assumes that all measured times in the atx.raw file are valid, especially if many samples are taken. When plugged into the extrapolation algorithm, small variations may show up as "other overhead."

- A related source of inaccuracy occurs when loop iterations are not always exactly identical in terms of work performed. IF-THEN-ELSE logic, gather/scatter memory reference patterns, and varying inner-loop iteration counts can contribute to small variations in measured times from one sample to the next. These may not be sufficient to warrant being called variable work loops, but they may cause ATEexpert to produce an overhead value that is unrealistically too small or too large. When plugged in to the extrapolation algorithm, estimated parallel times for different numbers of CPUs may be inaccurate as a result.

- The compilers do not produce identical code in the three parts of the three-code model. Thus, unitasked code may run faster or slower than the master's parallel code or the slave code. The master's parallel code also may run faster or slower than the slave's parallel code. In the worst case, one may be executing in vector mode and the other in scalar (report this situation by filing a Software Problem Report (SPR) against the compiler product you are using). ATEexpert assumes that the compiler generates identical code with identical performance characteristics for each part of the three-code model. Thus, minimum times are assumed to be valid for any number of CPUs and any part of the three-code model. Plugging these into the extrapolation algorithm may produce times that are inconsistent with the code that the compiler generated.

- In loops that fail a threshold test, ATEexpert forces the times to be the unitasked times, plus some time to perform the threshold test. However, when overheads are computed for the loop, the difference between the forced time and Amdahl's Law appears as "other overhead." Amdahl's Law, in this case, is simply unitasked time divided by the number of CPUs. As the number of CPUs increases, the difference, appearing as other overhead, increases.
2.6.3 *atx.raw* Output File

The *atx.raw* output file allows a postprocessing tool to access data and interpret information easily. You should access the *atx.raw* file by using the ATExpert postprocessing tool. This section briefly describes each of the line types in the header. All times are in clock periods.

Header information:

<table>
<thead>
<tr>
<th>Line type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP</td>
<td>Total number of disconnects in parallel code and the average number of clock periods per disconnect.</td>
</tr>
<tr>
<td>DT</td>
<td>Execution date and time of this instrumentation. Also the elapsed time and CPU time of this execution.</td>
</tr>
<tr>
<td>DS</td>
<td>Total number of disconnects in serial code and the average number of clock periods per disconnect.</td>
</tr>
<tr>
<td>MX</td>
<td>Maximum and average CPUs used in parallel regions and the maximum memory usage for this run.</td>
</tr>
<tr>
<td>NP</td>
<td>Number of parallel regions. The number of times parallel regions were run parallel and the number of times the parallel regions were run unitasked.</td>
</tr>
<tr>
<td>SY</td>
<td>System ID, operating system release, and system type.</td>
</tr>
<tr>
<td>VS</td>
<td>Version of ATExpert.</td>
</tr>
</tbody>
</table>

Parallel region information:

<table>
<thead>
<tr>
<th>Line type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP</td>
<td>Minimum, maximum, and average time to begin parallel execution. It is the time from the entry of a parallel region to the beginning of the first control structure.</td>
</tr>
<tr>
<td>PR</td>
<td>Parallel region information. This line contains the master routine name, the parallel region (slave routine) name, the number of times the parallel region was executed, the number of parallel loops in the parallel region, the number of CPUs that</td>
</tr>
</tbody>
</table>
entered during parallel execution, and the number of disconnects that occurred for which the ATE expert library could not determine where to subtract the associated wall-clock time (the times are rejected unless they are the only samples taken).

PS

Minimum, maximum, and average preceding serial time, which is the interval between the end of the previous parallel region and the start of the current parallel region and the name of the preceding parallel region. As such, it can include subroutine calls, returns, nonparallel loops, and any other type of serial code. For the first parallel region in a subroutine, it is not the time from the top of the routine.

SA

Minimum, maximum, and average time for the first slave CPU to arrive in the slave routine. Slave arrival overhead is described in Section 2.6.2.

Parallel work (loop/case) information:

<table>
<thead>
<tr>
<th>Line type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>Minimum, maximum, and average begin control structure times. The begin control structure time is the interval from the entry to a control structure to the top of the concurrent iteration loop. For CASE control structures, the BC lines contain the times to reach the corresponding CASE.</td>
</tr>
<tr>
<td>DO</td>
<td>DO loop or CASE identifier, and the number of untasked, scalar iterations, and the number of concurrent iterations. The identifier usually is the Fortran DO loop statement number (label). However, if one does not exist, particularly for a CASE, the subroutine line number is used. The DO line differs somewhat from other lines, because different data can be contained for different types of parallel work. For example, the following tags can appear on the DO line:</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VECTOR</td>
<td>Indicates a parallel vector loop</td>
</tr>
<tr>
<td>CASE</td>
<td>Indicates a CASE control structure</td>
</tr>
<tr>
<td>SAVELAST</td>
<td>Implies that the master task must take the last concurrent iteration</td>
</tr>
<tr>
<td>VECRED</td>
<td>Indicates a parallel vector reduction loop</td>
</tr>
<tr>
<td>GUIDED #</td>
<td>Indicates guided scheduling with a minimum number of iterations</td>
</tr>
</tbody>
</table>

If these tags do not appear on the DO line, the loop iterations are assumed to be distributed one at a time or in chunks to the participating CPUs.

Three DO lines with the same identifier inside the same parallel region designate a variable work loop. See Section 2.6.4, page 62, for more information.

| IL       | Minimum, maximum, and average interloop times.                             |
| IO       | Minimum, maximum, and average iteration overhead times. The maximum value is not the maximum iteration overhead measured; rather it is the minimum value measured when the maximum number of CPUs was working. This is used to give ATExpert an estimate of memory or shared-register contention during the concurrent execution of a parallel loop. See Section 2.6.2, for a description of iteration overhead. |
| LS       | Minimum, maximum, and average loop synchronization times. See               |
Section 2.6.2 for a discussion of loop synchronization overhead.

**PD**  Minimum, maximum, and average processor delta times, which is the interval of time between arriving CPUs at the top of a control structure.

**TU**  Number of parallel samples run with *n* CPUs and the minimum corresponding parallel time. For variable work loops, the three *Tn* lines represent the average, minimum, and maximum times. If a *Tn* line does not exist for a given value of *n*, no samples were taken with that number of CPUs working.

**TU**  Number of untasked samples run and the minimum untasked time. For variable work loops, the three *TU* lines represent the average, minimum, and maximum times, respectively.

**WS**  Minimum, maximum, and average wait slave times. See Section 2.6.2 for a discussion of wait slave overhead.

BP, SA, BC, PD, IO, LS, IL, and WS lines are written to the atx.raw file only if reasonable, nonzero times were measured for them.

Other information:

<table>
<thead>
<tr>
<th>Line type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC</td>
<td>A warning error condition code and associated messages string.</td>
</tr>
<tr>
<td>ES</td>
<td>Time from the end of the last parallel region until the completion of the program and the name of the last parallel region executed.</td>
</tr>
</tbody>
</table>
INFO

Additional information to be used to interpret the error condition.

2.6.4 Variable Work Loops

*Variable work loops* are defined as parallel loops in which ATExpert detects significant variations in the per iteration wall-clock times among different executions of the loops. These loops probably will contain *IF-THEN-ELSE* statements with varying degrees of truth, inner loops with varying iteration counts, calls to concurrent routines with varying execution paths, and/or vector reduction loops with varying vector lengths.

Variable work loops are a difficult problem for ATExpert, because it cannot produce one measured time that accurately depicts loop performance. When a variable work loop is identified, ATExpert prints three timing records for the loop, rather than the usual one. The three records correspond to the average, minimum, and maximum times measured for the loop.

ATExpert does not generate variable work loop information for parallel case structures. Only minimum times are used for parallel cases. This may be changed in a later release of ATExpert.

Variable work loops are identified by a small vertical bar preceding the loop identifier, parallel region name, or subroutine or function name. A warning message also is printed after ATExpert has read an *atx.raw* file that contains a variable work loop.

When the speedup graph is displayed for a variable work loop, error bars also appear along the speedup curve. This represents ATExpert’s best guess at the possible range of speedups that you can expect for the known variations of measured times.

The algorithm used to detect variable work loops is not perfect. Noise in the measured times can cause ATExpert to identify a variable work loop when one does not exist. The reverse also is true, but it is less common. Also, because of the possibly noisy times, ATExpert skips the variable work loop algorithm for very small granularity loops (unitasking times of a few thousand clock periods).

As a result of having a very limited set of measured times for a variable work loop, the ATExpert extrapolation algorithm may produce very inaccurate times. If the variation of times is such that the extrapolation algorithm may break down, a warning message is printed to caution users about the reliability of the displayed performance data.
Users are asked to use speedups from variable work loops with caution. If possible, compare the information ATExpert is showing you with other knowledge you may have about your program. Run your instrumented program more than once, and check whether ATExpert produces the same speedups; if not, be very skeptical of the quality of the ATExpert measured values or extrapolation algorithms. Use its results only as a relative guide to your program’s Autotasking performance, not as an absolute measure.

### 2.6.5 Dedicated and Maximum Speedup Curves

ATExpert tries to predict dedicated speedups from data collected on a nondedicated system; this is the default. However, ATExpert provides an option to show maximum speedups, which are defined as those computed by using minimum times in all speedup calculations. The use of minimum times, however, may not accurately reflect actual execution paths taken by the program.

Dedicated speedups use a formula in which minimum times are used in all calculations, except when adding in preceding serial time. For this calculation, a fraction of the difference between the average and minimum preceding serial time is added in. The rationale for this dedicated speedup calculation is that preceding serial code often contains alternative paths. Examples of alternative execution paths are as follows:

- I/O performed every so many time steps
- Initialization occurring on the first invocation of a routine and never thereafter
- Data-driven algorithms

When alternative paths are taken, serial code performance can vary greatly. Using minimum times to predict expected speedups for a subroutine or function or program can yield wildly inaccurate results. For example, ATExpert predicts a program speedup of 4.3 when displaying the maximum curves for an important application program. The dedicated curves show a speedup of 2.3, which is much closer to the measured speedup of 2.1.

If alternative paths are not taken in preceding serial code, average and minimum times usually are close in value and the effect of adding the difference is minimal. Very little difference should exist between maximum and dedicated curves for programs with no alternative paths (or none of significance).

The ATExpert hardwired dedicated speedup algorithm seems to work well for most application programs. Feedback from users indicates that predicted
speedups consistently are close to those actually measured on a dedicated system. Instances will occur, however, when the ATExpert hardwired algorithm does not produce an accurate speedup prediction. For example, if huge amounts of I/O are performed infrequently, ATExpert’s algorithm may break down. In this situation, I/O may dominate the program’s real performance. However, ATExpert’s algorithm may be more biased toward minimum preceding serial times. Therefore, it will predict larger speedups than you actually may obtain on a dedicated system.

A quick way to check whether ATExpert may be predicting inaccurate dedicated speedups is to compare them with the maximum speedups. If the difference between speedups is great, the ATExpert dedicated speedup algorithm may not be correctly accounting for the actual alternative paths taken through the program’s serial code.

You can use the parallel region monitor section to identify areas of the program in which the dedicated speedup algorithm is having trouble. These areas will be those in which preceding serial time (shown as boxes with a height of 1) changes dramatically when switching between maximum and dedicated curves. You can inspect these areas of the program to see what effect, if any, alternate paths may be having on the program’s performance.

A future release of ATExpert may provide more user control over the dedicated speedup algorithm as a configuration option.

### 2.6.6 Software Anomalies

Known anomalies exist with ATExpert. This section details the known problems and discusses potential areas of misunderstandings.

- The atexpert command - ATExpert measures performance of Autotasking, not microtasking or macrotasking.

- Dedicated speedups - ATExpert’s calculation of dedicated speedups seems to be accurate for most programs; however, situations do exist in which the predicted dedicated speedups may be inaccurate. (See Section 2.6.5, page 63, for a more complete description of when this problem may occur.)

Also, ATExpert’s predictions are based on the environment from which the measurements were made. If the MPDEDICATED environment variable is set, it makes predictions based on this value, because the value is part of the environment. ATExpert cannot predict speedups for when this variable is set from measurements taken when it is not set.
• Extrapolation - The extrapolation system is a smart system, but it does need as much data as possible to make correct extrapolations. The more timing data collected for a program, the more accurate the extrapolation will be for areas that did not get all of the CPUs from the system that it could effectively use.

• I/O - If you have heavy I/O in your program, the preceding serial times in those areas is minimum time.

• Multiple disconnects - The ATEXpert report file contains the number and length of disconnects. The higher the number of disconnects, the less accurate the numbers may be. ATEXpert does handle disconnects, and it subtracts the disconnect time from the timed information, but large numbers and lengths of disconnects still can cause some problems.

• Preceding serial time - The preceding serial time is defined as the time from the end of the last parallel region to the beginning of the next parallel region. In the case of the first parallel region, this is the time from program startup to the beginning of the first parallel region. In all cases, the serial time can cross subroutine calls and is not to be mistaken for time from the beginning of the subroutine.

• Sharply defined peaks and valleys in the speedup curve - Sometimes ATEXpert’s speedup curves may have significant peaks and valleys. Peaks and plateaus usually are reasonable, however, deep valleys or sharp peaks are not. These occur when measured overhead values either are not available and must be estimated, or they are inaccurate (possibly because of too few samples). In either case, extrapolated parallel times to fill in gaps in measured times may be too large (or small) compared with adjacent measured times. The result is a speedup curve with sharply defined peaks and valleys. A future release of ATEXpert will address this problem. In the interim, rerunning an instrumented program often eliminates the peaks or valleys, because better overhead values are measured or the gaps are filled in.

• Single-CPU instrumentation - ATEXpert can produce valid speedup predictions even when the instrumented program is run on only one CPU. The quality of this prediction, however, is proportional to the granularity of work in parallel regions. The predictions that ATEXpert makes should improve as granularity increases.

• Threshold tests - Threshold tests that the AutoTasking software generates can affect ATEXpert’s interpretation of timing measurements. For example, if iteration counts vary from one invocation of a parallel region to the next, a threshold test can force unitasking samples to be taken for low iteration
counts and Autotasking samples to be taken for large iteration counts. This can lead to larger than expected speedups or the detection of variable loops. ATE Expert tries to detect threshold tests and to interpret the results accordingly, but it is not perfect. If a threshold test may be giving misleading results, ATE Expert will print a warning message.

- Uninstrumented multitasked libraries - Libraries that are microtasked or autotasked without ATE Expert instrumentation will be measured as preceding serial time. This includes libsci calls.

- Variable work - When variable work occurs at the loop level, it also can affect the projected program speedup. See Section 2.6.4, page 62, for a detailed explanation of variable work loops.

- Variable iteration loops - ATE Expert can display inaccurate speedups for a program dominated by loops with varying numbers of concurrent iterations. ATE Expert generally uses the minimum times from running the maximum number of concurrent iterations. This tends to bias overall speedups, however, toward larger values. Actual program speedups may be less than those predicted by ATE Expert. If the average number of concurrent iterations is more than 20% different than the maximum, ATE Expert will print a warning message.

- Wait Slave overhead - The wait slave overhead can be misleading if all slaves return to the Autotasking library before the master task finishes its last iteration, which typically is not the case. See Section 2.6.2 for a discussion of wait slave overhead.