



BERT 77

FEATURES GUIDE

HPC Design
1008 Woodmere Drive
Bethlehem, PA 18017
www.hpc-design.com

CONTENTS

| | |
|--|-----------|
| BERT 77 FEATURES..... | 3 |
| WHY IS BERT NEEDED?..... | 4 |
| BERT DETERMINES EFFICIENCY..... | 5 |
| HOW TO USE BERT..... | 6 |
| HOW TO START THE XBERT USER INTERFACE..... | 7 |
| XBERT MAIN SCREEN..... | 8 |
| XBERT MAIN MENU BAR..... | 9 |
| HOW TO CHANGE PARALLEL COMPUTER SYSTEM PROFILE..... | 9 |
| HOW TO ADD FORTRAN FILES..... | 11 |
| HOW TO SAVE A PROJECT..... | 12 |
| HOW TO ANALYZE A PROGRAM..... | 12 |
| HOW TO EXAMINE RESULTS..... | 13 |
| HOW TO INTERPRET THE TREE GRAPH..... | 13 |
| HOW TO SEE PARALLEL PROGRAM CODE..... | 14 |
| HOW TO SEE LOOP INHIBITORS..... | 15 |
| HOW TO CHANGE NUMBER OF PROCESSORS..... | 16 |
| HOW TO USE THE BERT FLOW CHART..... | 16 |
| IMPORTANT POINTS..... | 18 |
| Why are some programs not converted?..... | 18 |
| Why is the estimate of execution time incorrect ?..... | 18 |
| What are recursive variables ?..... | 18 |
| Effect of I/O..... | 18 |
| Libraries and Unknown Functions..... | 18 |
| OTHER RESOURCES:..... | 19 |

BERT 77 FEATURES

BERT IS MORE THAN A **CONCURRENCY DETECTOR**

BERT IS AN **OPTIMIZING** CONVERSION TOOL

Feature: Automatic detection of concurrent parts

Advantage: Global dependency analysis - finds all concurrent parts of your program (not just loops)

Feature: Estimates parallel efficiency of concurrent parts

Advantage: Only converts efficient concurrent parts
(not all concurrent parts are efficient in parallel!)

Feature: Estimates parallel efficiency of **non-concurrent** loops

Advantage: Determine if “parallelizing” a loop will result in program speed-up
Do not waste time with inefficient loops!

Feature: Pre-run-time performance estimations allow conversion of programs, development of algorithms, and changes in processors without the edit/compiler/run/measure cycle.

Advantage: Save time converting programs. Allow parallel machine to be used for production work (not development work)

Feature: Provides a quick feasibility study

Advantage: In very short time, learn how well program can be executed in parallel.

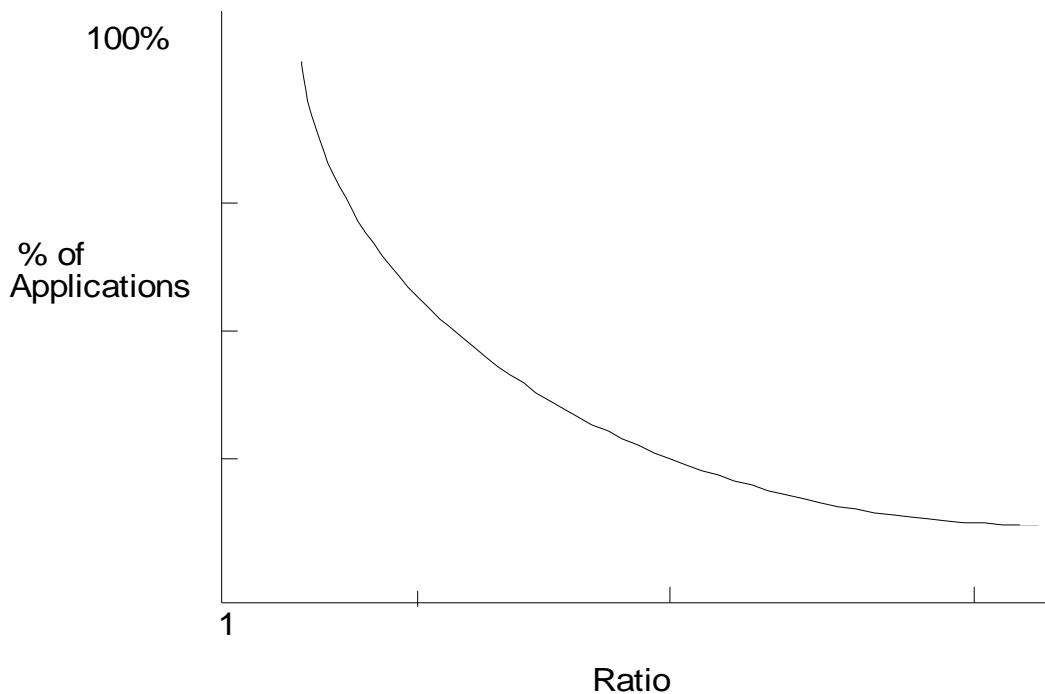
WHY IS BERT NEEDED?

Suitability of Machine to Application Program

The suitability of a given machine depends upon:

$$\textbf{suitability ratio} = \textbf{processing speed/communication speed}$$

Assume single CPU =1 then the further away from 1, the less number of applications will be suited for this machine. **Suitability Estimate** determines if a particular machine can be used for your application.



BERT DETERMINES EFFICIENCY

To know how well an application will operate on a parallel computer we need to consider:

CPU/compiler (g77, f77)

Communication/speed (PVM, MPI, Threads)

Parallelization model (Dataflow, Dynamic, Static, Shared Memory)

What is best ?

Compilers do not optimize these variables, so user must determine if application (or parts of application) are suitable for a particular parallel machine. Wrong suitability estimates can result in performance degradation!

BERT WILL OPTIMIZE THESE VARIABLES!

HOW TO USE BERT

PLEASE NOTE: BERT 77 is powerful new parallel optimization technology for FORTRAN 77. This guide is intended to help you investigate the important features of BERT 77. It is not a complete user manual and many features of BERT are not presented in this document.

STEP 1: Consult the "README.bert" file in
/opt/bert_lite/doc.

STEP 2: Make sure your program compiles with `g77` and runs correctly on *one CPU* before using BERT.

STEP 3: To investigate BERT, follow the instructions on the next several pages.

STEP 4: Read the "*IMPORTANT POINTS*" section.

How to Start the XBERT User Interface

1. Set the following (may be placed in .cshrc file):

bash:

```
export BERT=/opt/bert_lite/bin
export PATH=$PATH:$BERT
```

csh:

```
setenv BERT /opt/bert_lite/bin
set path = ($path $BERT)
```

2. Copy ~bert/XBert to your home directory (only on first use of BERT)
3. Start the BERT daemon by entering:

```
$BERT/bertd
```

(Only if bertd is not running)

4. Move to a working directory that contains your FORTRAN files (or use opt/bert_lite/examples). Enter:

```
xbert
```

5. The XBERT main screen should now be visible.

XBERT Main Screen

XBert [/home/deadline/bert/bin/examples/mandel.xbp]

File View Make Project Options Help

Profile: PPR0200-linux, g77, MPI mpich/p4 Select Profile...

Method: Automatic FLOW Normal All par Number of processors: 8

Estimated sequential time: 4.84 seconds Estimated speed-up: 3.93 (74.58%)

Estimated parallel time: 1.23 seconds Potential speed-up: 3.93 (74.58%)

Program's files list

| File Name | Lines | Bytes |
|---|-------|-------|
| /home/deadline/bert/bin/examples/mandel.f | 332 | 7836 |

Add... Edit Delete Sources: Original Help

READY Database date Tue Dec 2 12:24:23 1997

Done File: Module: Line Pass

Important Items:

Profile: BERT makes a parallelization for each type of parallel computer. Use "Select profile to choose a specific parallel computer.

Method: BERT can use two different "parallelizations" models. The best model is determined automatically. User can select specific model if needed.

Number of Processors: User can enter a specific number of processors. Note: this can be done in second box (right side) or under BERT options. Changing the number of processors also makes current estimates invalid.

Estimated sequential time: BERT estimates the sequential time for your application. (the user may need to supply further information if this estimate is incorrect)

Estimated parallel time: BERT supplies estimated parallel time for the specific parallel computer. BERT will only perform safe parallelizations - those that have all dependencies resolved automatically.

Estimated speed-up: BERT measures speed-up as (sequential time/parallel time). Percent speed-up is ((sequential time-parallel time)/sequential time)*100

Potential speedup: This is maximum speed users program will run if all data dependencies are removed. These are data dependencies that were not removed automatically by BERT.

Program files list: A list of program files used for the analysis.

Status windows: Provides an indication what BERT is doing.

XBERT Main Menu Bar

| | | | | | |
|-------------|-------------|-------------|----------------|----------------|-------------|
| FILE | VIEW | MAKE | PROJECT | OPTIONS | HELP |
|-------------|-------------|-------------|----------------|----------------|-------------|

Brief Menu Bar Summary:

FILE

- OPEN FILE - open a file for editing
- QUIT (ctrl q) - quit XBERT

VIEW

- FILES (F7) - show file view (used for adding files)
- SUMMARY (shift F7) - show text summary window (after analysis)
- MODULES (ctrl F7) - show modules browser (after analysis)
- LOG WINDOW (F5) - show message log window
- PROJECT TREE (F6) - open project tree window

MAKE

- INFORMATION (F9) -do parallel analysis without writing new sources
- PRAGMAS (Shift F9) * - rewrite sources with BERT pragmas inserted
- PARALLEL SOURCES (Ctrl F9)* - generate parallel sources

PROJECT**

- OPEN - open an existing project
- IMPORT - import BERT text mode version project file
- SAVE - save project
- SAVE AS - save project as ...
- REMOVE - remove a project
- FILES - short cut to add many files to a project
- BERT OPTIONS - a comprehensive list of BERT options**

OPTIONS

- LOG AUTO RISE - log is displayed automatically if messages
- AUTO LOAD LAST PROJECT - automatically load last project file
- SAVE SORT&FILTERS AS DEFAULT - save view filter and sort information
- SHOW PROGRESS INFORMATION (F4) - display progress information

HELP

- CONTEXT (F1) - show context help
- CONTENTS (Shift F1) - show help contents (not complete)
- INDEX (Ctrl F1) - index of help (not complete)
- ABOUT - license information

* Not available on the demonstration version.

**BERT operates using the "project file concept. A list of all FORTRAN modules and BERT options are kept in a "project file":

How to Change Parallel Computer System Profile

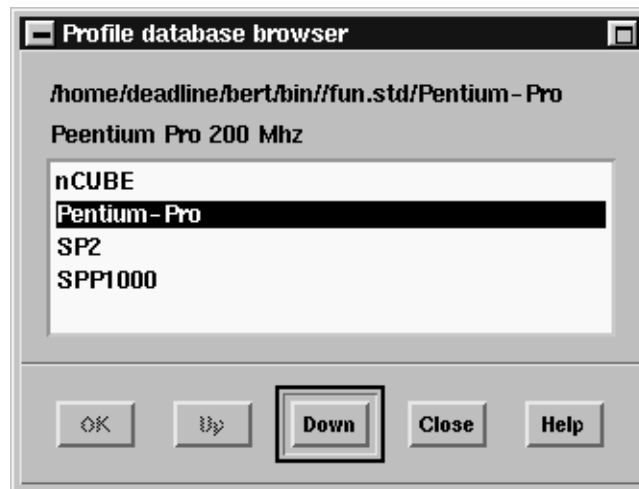
The demonstration version has four computer profiles. The profiles are classified (where applicable) by the machine, FORTRAN compiler, and message passing interface. The following profiles are included:

- nCUBE/nf77

- Pentium-Pro/g77/MPI/mpich/p4
- SP2/xlf/POEMPI/ip
- SPP1000/fc/PVM

For instance, Pentium-Pro/g77/MPI/mpich/p4 indicates a Pentium Pro processor, with g77, MPI, mpich implementation, the p4 device.

A profile can be selected by pressing the “Select Profile...” button. The selection box will look like the following:

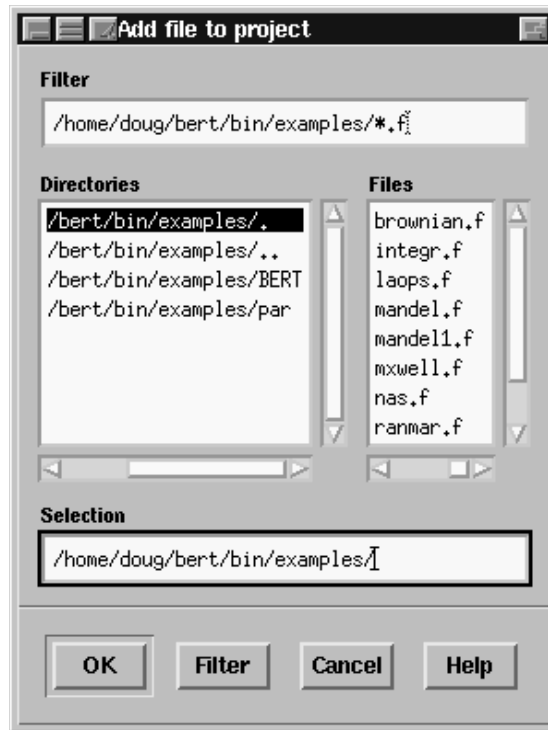


Use the “Down” and “Up” buttons to select a profile. You must select a complete profile. Continue until you can not move down, then select OK.

NOTE: Changing the profile will make the current results invalid (as will changing the number of processors). You will need to “MAKE/INFORMATION” to use the new profile.

How to Add FORTRAN Files

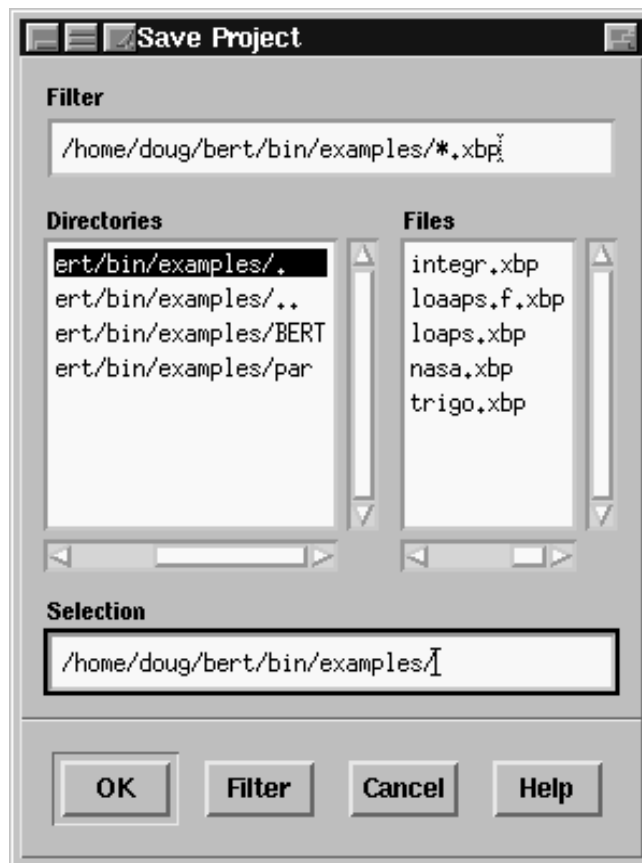
1. Set the View to Files (press F7). The Files View is default.
2. Click on the ADD box near the bottom of the window.
3. Select the FORTRAN File to Add using the window below:



4. Add as many files as needed. In this example, we will add one file: `"trigo.f"`. Click OK when done.

How to Save a Project

1. Under the PROJECT Menu select SAVE.
2. Enter the name of the project with the extension “.xbp”.
3. Click OK when done.

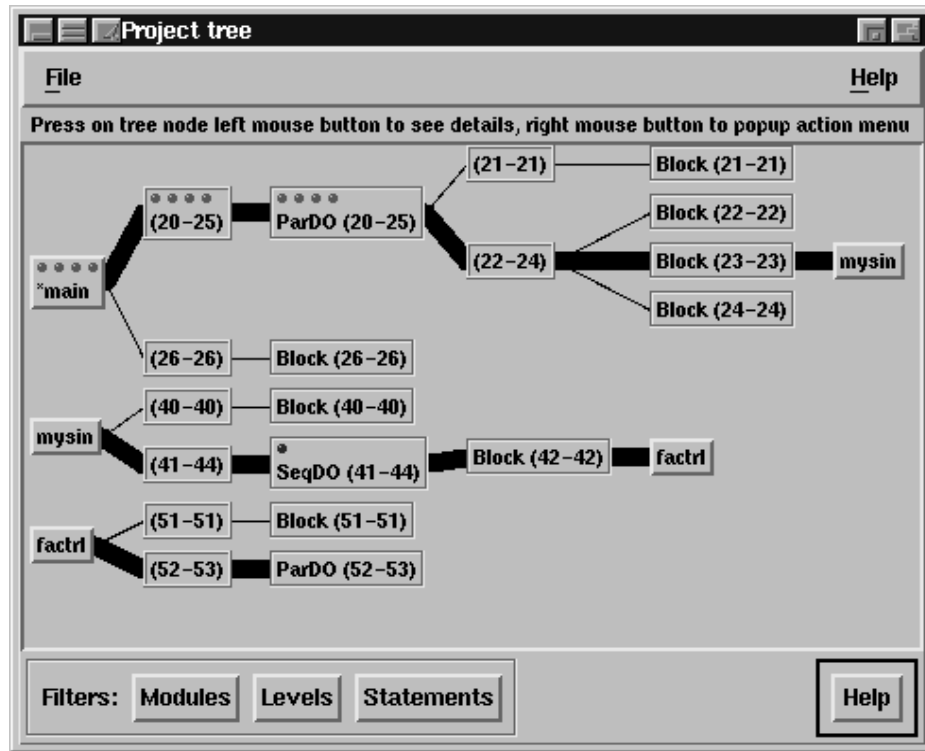


How To Analyze a Program

1. Select MAKE/INFORMATION.
2. BERT will now process the file(s).
3. New information will now appear in the Main window:

How to Examine Results

1. Select VIEW/PROJECT TREE.
2. A tree view of your program will be shown:



How to Interpret the Tree Graph

Line thickness is amount of computation “flow” in application (based on BERT estimates).

ParDo is “do loop” parallelized by BERT.

SeqDo is “do loop” not parallelized by BERT for two possible reasons:

- 1) loop has inhibitors (loop carried dependencies).
- 2) loop is not efficient when executed in parallel).

Number of “dots” represent the efficiency of the loop when executed in parallel.

Blue dots are for sequential loops.

Red dots are for parallel loops.

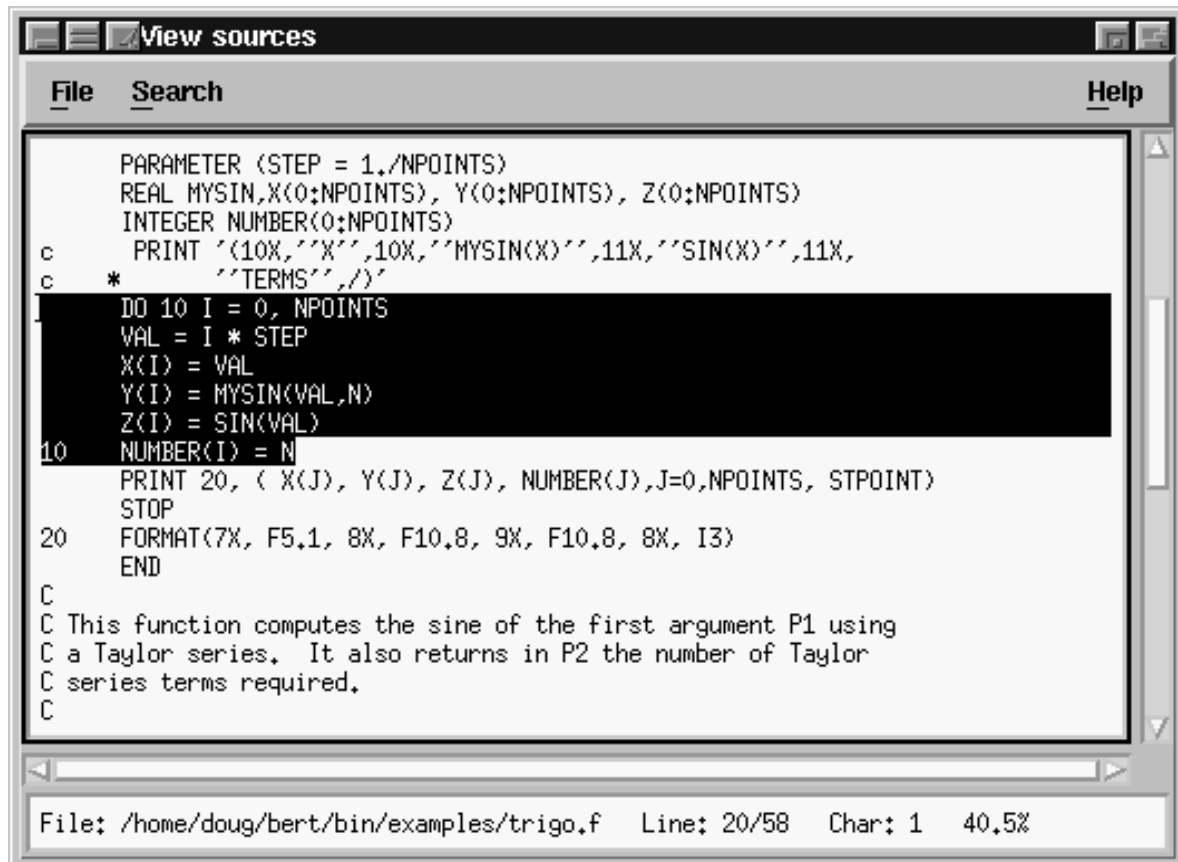
Line numbers corresponding to tree nodes are in parenthesis. “ () ”.

Left Mouse Button is used to see percent of program computation/parallel speed-up for loop/ and number of parallel parts. (weight/gain/parts).

Right mouse button is used to View/Edit/Browse a block of code.

How to See Parallel Program Code

Move the mouse to box “ParDo (20-25)” on the tree graph. Click the Right Mouse Button and select VIEW. The parallel loop will be highlighted.



```

PARAMETER (STEP = 1./NPOINTS)
REAL MYSIN,X(0:NPOINTS), Y(0:NPOINTS), Z(0:NPOINTS)
INTEGER NUMBER(0:NPOINTS)
c  PRINT '(10X,"X",10X,"MYSIN(X)",11X,"SIN(X)",11X,
c  *      "TERMS",/)'
10 DO 10 I = 0, NPOINTS
   VAL = I * STEP
   X(I) = VAL
   Y(I) = MYSIN(VAL,N)
   Z(I) = SIN(VAL)
10  NUMBER(I) = N
   PRINT 20, ( X(J), Y(J), Z(J), NUMBER(J),J=0,NPOINTS, STPOINT)
   STOP
20  FORMAT(7X, F5.1, 8X, F10.8, 9X, F10.8, 8X, I3)
END

C
C This function computes the sine of the first argument P1 using
C a Taylor series. It also returns in P2 the number of Taylor
C series terms required.
C

```

File: /home/doug/bert/bin/examples/trigo.f Line: 20/58 Char: 1 40.5%

How to See Loop Inhibitors

Move the mouse to box “ParDo (20-25)” in the tree graph. Click the Right Mouse Button and select BROWSE. The browse window will be shown. Loop inhibitors are given in Inhibitors Box. (In this case there are none.)

Parallel estimations are given for the loop:

Estimated Sequential Time - estimated time for loop if sequential.

Estimated Parallel Time - Estimated parallel time for loop.

Speed-up - (Sequential time/parallel time).

Contribution to Overall Speed-up. How much this loops contributes to overall speed-up of program.

Browse statement

Statement Help

Start: 20 end: 25 Concurrent: YES

Repetitions: 1 Efficiency if Parallel: EXTRA

Scheduled as: parallel loop

Has pragmas for: none

Split by: BERT Executed on: Worker

Estimated Sequential Time: 1.9e+02 Percent of Total: 100,00%

Estimated Parallel Time: 27 Speed up: 85,65% Contribution to Overall Speed-up: 100,00% **10 vars**

Inhibitors:

Levels:

| Lines | Parts | Seq. time | Weight | Gain | Type |
|-------|-------|-----------|--------|-------|------|
| 22-24 | 1 | 1.9e+02 | 99,99% | 0,00% | I |
| 21-21 | 0 | 0 | 0,00% | 0,00% | |

Edit Browse View Filter... Sort by: Sequential time Help

Called modules:

| Name | File | Seq. time | Weight | Gain | Type |
|------|------|-----------|--------|------|------|
| | | | | | |

Edit Browse View Filter... Sort by: Sequential time Help

Substatements:

| Statement | Seq. time | Weight | Gain | Prize Type | Pragma Place | Parts |
|-----------|-----------|--------|------|------------|--------------|-------|
| | | | | | | |

Edit Browse View Filter... Sort by: Sequential time Help

How to Change Number of Processors

Select Menu PROJECT/BERT OPTIONS and select MISCELLANEOUS.

Enter number of processors.

How To Use the BERT FLOW CHART

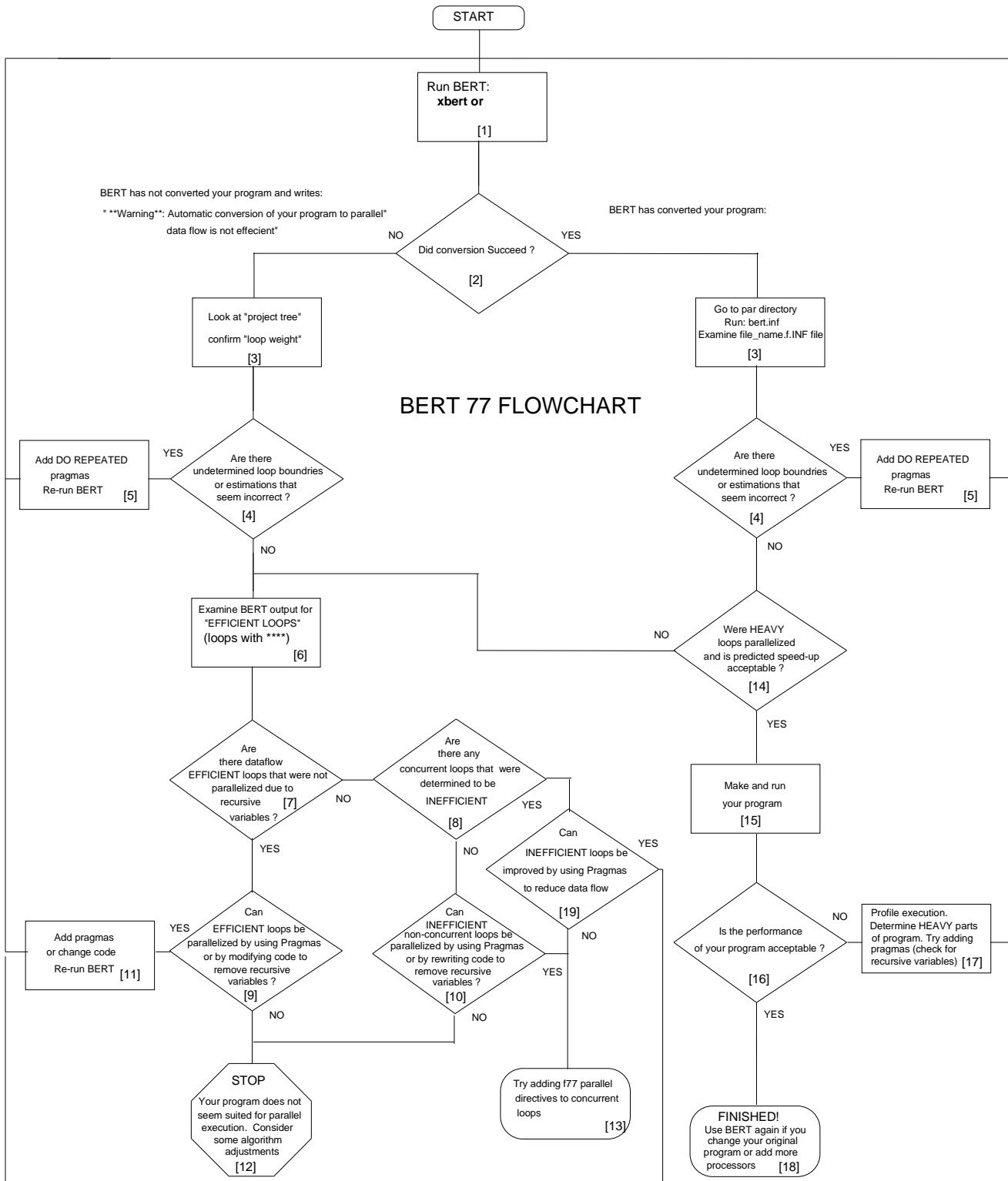
A FLOW CHART is provided to assist the decision process when using BERT. Consult the BERT Users Guide for notes on how to use the FLOW CHART.

In general, you should first examine if BERT has made a good estimation of sequential execution time. If it sequential times seem reasonable then BERT will make appropriate parallelization decisions. If estimated sequential execution time is **greater or less then expected** (+/- an order of magnitude or more), then you will need to give BERT some “hints” about loop repetitions. See the BERT Users Guide. Once the sequential estimation is close to the actual time, look for sequential do loops (SeqDo) with 3 or 4 blue dots. These are sequential loops that will be highly efficient if inhibitors can be removed. Check inhibitors for these loops. If they can be ignored (use BERT ParDo pragma) otherwise try re-writing the code.

REMEMBER: BERT will not convert loops that are not efficient, so it may be possible that your program has concurrent loops that BERT does not make parallel.

BERT 77 FEATURES GUIDE

BERT 77 FLOWCHART



IMPORTANT POINTS

Why are some programs not converted?

Sometimes the *BERT* analysis may produce disappointing results. For example, some programs with known concurrent portions do not parallelize using *BERT*. Aside from recursive variables, there is only one other reason why *BERT* will not create a parallel program; *BERT* thinks the conversion will be inefficient.

The inefficiency is due to how well a particular parallel computer is suited to your FORTRAN program. *BERT* currently has two parallelization modes (or models) that it uses to attempt the best parallelization. This method ensures that *BERT* will find the best parallelization for your program, but in some cases the program is not suited (it is inefficient) for your machine.

In other cases, *BERT* sees no need for parallelization. Always check the expected sequential execution time estimated by *BERT*. If this time is low (on the order of seconds) then it is probably not economical to convert the program to parallel. If you know the program will take longer than that which *BERT* predicted, then *BERT* needs more information; such as DO REPEATED pragmas or unknown function call times.

Why is the estimate of execution time incorrect ?

There can be several reasons,

1. *BERT* does not consider optimizations performed by the compiler.
2. *BERT* does not have enough information to determine how the program will run.

As mentioned, an important result is that if *BERT* determines a low execution time, then it will not parallelize the program. On the other hand if *BERT* estimates a high execution time, then some more information must be provided to *BERT* through the use of DO REPEATED pragmas. There may be certain loop boundaries that are read from data files or determined at runtime. In these cases, it is best to give *BERT* a "hint" using the DO REPEATED pragma. For instance:

```
DO I = 1, N
C$BERT DO REPEATED(5000)
      X = X + ..
ENDDO
```

This pragma tells *BERT* that the loop will be repeated 5000 times. Note: an approximate estimate is needed and pragma is placed after DO statement..

What are recursive variables ?

Recursive variables are data elements that may limit parallelism. For example, if a loop uses a variable that depends upon previous cycles of the loop, then this loop is not concurrent and can not be parallelized. Distributing this loop across multiple processors is inefficient because each cycle of the loop must wait for the previous cycle to finish before it can begin.

BERT recognizes recursive variables in your program and will not parallelize a loop or block that contains these variables. It is possible that some variables *BERT* "thinks" are recursive are actually not recursive. In this case, *BERT*'s decision can be overruled with a PARDO pragma. See Chapter 3 in the Reference Manual for information on "false recursion".

Effect of I/O

BERT does not account for I/O in its analysis. I/O is a difficult parameter to measure because it can vary from system to system and from day to day use. In general, if the program does a lot of I/O between parallel segments, the effect of parallelizing the code may be minimal because most of the program's time is I/O.

Also *BERT* cannot parallelize loops that have I/O statements in them. If the I/O is simple informational data, consider re-writing the loops to hold off on printing the data or eliminate it altogether.

Libraries and Unknown Functions

Libraries and unknown functions are difficult for *BERT* to handle. If function (or subroutine) is not in the fun.std file, then *BERT* has no knowledge of how much time the function will require or whether the function has data dependencies. This does not stop the *BERT* analysis however, it assumes the time is zero and makes the worst case assumption that the function can have any dependency within the program and continues the analysis. *BERT* will produce a warning message if it detects an unknown function call.

There are three problems with unknown functions:

- 1) time estimation
- 2) dependency analysis
- 3) linking at compiler time

Solutions to these will be covered below.

To solve the problem of time estimation, a pragma may be added. To account for the time of the unknown function, a TIME field has been added to the DATA pragma. This allows you to tell *BERT* the time required to call this function. The form of the TIME field is as follows:

TIME={time_value [scale]}.

where:

time_value is integer constant and scale is s (seconds), ms (microseconds) or ns (nanoseconds) The default is s (seconds).

For example:

```
c$BERT data time={12345 ms} IN = ...
```

indicates to *BERT* the execution time of an instruction following pragma is 0.012345 seconds. The time is for one pass of the instruction (or function call), not total in program. The user must determine this execution time.

To solve the problem of dependency analysis, there are two possible solutions. This problem arises when *BERT* must assume the unknown function can call any function in the program or may contain STOP and file instructions. Therefore, a loop that contains an unknown function call may have a large unresolvable set of dependencies.

The first method is to use the -call command-line option for *BERT* (or set the option in XBERT). This option indicates to *BERT* that all unknown modules have no side effects and will only use and modify their arguments. This option is applied to ALL unknown modules in program. Use of the call option is recommended when unknown modules or routines from some library are used.

The second method is to create a dummy function to be used for the *BERT* analysis. If there are no real dependencies in the unknown function call, then the dummy function can be quite simple (just a RETURN statement).

After the dummy function has been added, re-run *BERT* and examine the results for the loop in question. If the loop is scheduled as parallel (or at least is concurrent), then remove the dummy function and add a PARDO pragma with the appropriate TIME value in the DATA field. Run *BERT* again and this time the loop will be converted to parallel. Check the efficiency estimations to make sure that parallelizing the loop will actually speed things up. If the loop is not concurrent, then you may wish to examine the recursive variables. Keep the dummy function in for this step.

OTHER RESOURCES:

BERT MANUALS

EXAMPLES in ~/bert/examples

QUESTIONS: berthelp@ploginc.com