INTERPROCEDURAL ANALYSIS: A NEW TECHNOLOGY FOR SOFTWARE DEVELOPMENT

WHITE PAPER SERIES

BY ROBERT METZGER
CONVEX COMPUTER CORPORATION

A new type of application development software is making its appearance in the marketplace. This software makes use of a new technology called interprocedural analysis. It is being packaged into compilers, optimization tools, and CASE tools.

TRADITIONAL COMPILER TECHNOLOGY

Traditional procedure compilers have some significant limitations. If writers edited papers the way procedure compilers process programs, they would check each page of the paper t ughly. They would not, however, check that the last sentence on one page made a logical connection with the first sentence of the following page. Nor would they check whether there were dangling references or undefined terms on one page that no other page resolved. In fact, every time they picked up a new page, they would completely forget everything they had read on the previous page.

Procedure compilers must make worst-case assumptions about theside effects of procedure calls. This means that they must make these assumptions.

- Every global variable used in the calling procedure is used and assigned by the called procedure.
- Every variable passed by reference is used and assigned by the called procedure.
- ry memory location that might be pointed at by pointers used by the called procedure is used and assigned.

The result of making such assumptions is over-cautious optimization andapplications that don't execute as fast as they might. Procedure compilers generally assume procedure invocations are correct. Some languages (such as Ada) require the programmer to declare theproper way to call a procedure, so that procedure calls can be checked.

Other languages (such as ANSI C) provide the means to check procedure calls (function prototypes) but don't require their usage. Some of the most commonly used languages (FORTRAN, common usage C) don't even provide the means to check the correctness of procedure calls, even if the programmer wants to. Compilers for languages that don't require procedure call checking cannot find common errors such as omitting arguments, passing arguments of the wrong type, and so forth.

Procedure compilers are unable to find subtle programming errors that require analysis of an entire application. They cannot determine whether global variables used in one procedure are initialized somewhere else. Nor can they find violations of Fortran's rule that a dummy argument and COMMON variable that refer to the same storage location must not both be assigned in the same procedure.

INTERPROCEDURAL ANALYSIS TECHNOLOGY

Compilers that use Interprocedural Analysis overcome many of the limitations of procedure compilers. Such compilers compile programs the way writers really edit papers. Each page is checked carefully. In addition, they check the connections between pages, and the references to terms and ideas used on multiple pages. They arrange the sections and pages in an order which is most profitable for the reader.

Compilers that use Interprocedural Analysis don't make any assumptions about the side effects of procedure calls. Instead they develop precise information on all the variables that are used or assigned by a procedure and any procedures it may call.

Compilers that use Interprocedural Analysis don't assume that procedure invocations are correct. Instead, because they must inspect the arguments transmitted at each call site, they find all calls where the wrong number or type of arguments are passed, or where the type of a function return value doesn't correspond to its usage in context.

Interprocedural analysis answers two basic questions:

- What are the actual objects referred to by name in each procedure?
- What are the side effects of invoking each procedure?

It answers these questions by performing the following analyses:

Call analysis - which procedures are invoked by a call?

Alias analysis - which names reference the same location?

Scalar analysis - which procedures (and subordinates) use and assign which scalar variables?

Array analysis - which procedures (and subordinates) use and assign which sub-sections of which array variables?

ADVANCED ARCHITECTURES

Interprocedural analysis also opens the way to effective use of advanced system architectures. Massively parallel systems normally have high communication latency between processors. Hence, synchronization must be infrequent if parallelism is to be used efficiently. This means that parallelism must be found at a very high level.

Massively parallel systems typically have distributed memories in which accesses to some locations take much longer than others. Hence, data must be distributed to the various memories by the programmer or by the compiler. This means that the compiler must optimize the placement and transfer of data.

Interprocedural analysis, particularly array analysis, gives the compiler the information it needs to find high level parallelism and to distribute program data across multiple memories. Researchers and vendors have been working on an extension of Fortran, called HPF, that allows the user to specify data distributions. The results of the research include the following.

Empirical results show that interprocedural compilation can improve performance by several orders of magnitude for an important application. We do not expect interprocedural optimizations to be required in all cases, but for many computations it can make a significant difference.[1]

FEATURES AND BENEFITS

Software tools that have been announced or shipped provide the following features.

AUTOMATIC PROCEDURE INLINING

The goal of procedure inlining is to reduce call overhead and remove procedure boundaries when they inhibit optimizations. Procedure inlining is the process of replacing a call to a procedure with the actual body of the called procedure. User-assisted inlining has been available in some compilers for over decade.

Automatic procedure inlining finds procedures and procedure calls which, if inlined, make the greatest improvement in application performance. It does this without user assistance, although it may use hints from the user or execution profile data in determining the profitability of inlining a given call.

INTERPROCEDURAL CONSTANT PROPAGATION

The goal of constant propagation is to find procedure interface variables (both arguments and global variables) that always have the same value on entry to a given procedure. Once those variables and their values are identified, the called procedure must be modified so that the constant value is used everywhere the name of the interface variable occurs.

A full-featured constant propagation algorithm should recognize constants in a variety of contexts:

- literal constants passed as arguments
- variables passed as arguments that always have the same value on entry to some procedure
- global variables that have been statically initialized
- global variables that have been assigned prior to the call always have the same value on entry to the given procedure

Constant propagation by itself doesn't yield significant performance improvements. Its value is found in the improvement of other optimizations that result from evaluating the resulting constants.

PROCEDURE CLONING

The goal of procedure cloning is to increase the opportunities for constant propagation. It does this by making complete copies of selected procedures and changing some of the calls to invoke the new procedure. The calls that are selected for modification are those where an argument can be treated as constant if the substitution is made. The effectiveness of procedure cloning depends on the approach used in determining which interface variables can be caused to become propagated constants, and what effect propagation will have on the cloned procedure.

INTERPROCEDURAL POINTER TRACKING

The goal of pointer tracking is to determine which pointers refer to which memory locations. Obviously, it it useful and necessary only for those languages (like C) that provide pointers that can point to arbitrary locations in memory. It is not necessary for languages that provide pointers that establish the base address of a single array, usually referred to as based storage, or pointer-based variables.

Pointer tracking greatly increases the effectiveness of other optimizations by reducing the number of pessimistic assumptions that a compiler must make.

PARALLELIZATION ASSISTANCE

Several vendors have offered compilers that perform automatic parallelization of loops since the mid-1980's. These compilers were only of limited usefulness, however, because they did not automatically perform parallelization of loops that contained procedure calls. The first step in overcoming this limitation is to perform Interprocedural analysis, and then provide the results to the programmer in a form that can be read and used to manually edit the program to run concurrently.

AUTOMATIC STORAGE OPTIMIZATION

On high performance computers, the best performance can only be obtained by carefully managing the memory hierarchy. This hierarchy includes registers, instruction and data caches, processor and system memories, and interleaved memory banks. Storage optimization does not save space. It rearranges the text and data portions of a program to save time.

There are as many types of storage optimization as there are elements of a memory hierarchy. One example of this type of optimization is array dimension extension. To get the best performance on systems that have interleaved memories, each memory bank should be actively serving a stream of requests. When any of the dimensions of an array, except the one that varies most slowly (left-most in Fortran), is a multiple of the number of memory banks, all of the accesses to that array may be going to a single memory bank. A compiler that performs interprocedural analysis can determine that it is safe to extend such dimensions. When this is done, the accesses are spread across all the banks, and performance is dramatically improved.

INTERPROCEDURAL ERROR CHECKING

Programmers are able to develop applications more quickly using program development tools

that use interprocedural analysis. This occurs because more bugs are found automatically, and because less manual effort is required to obtain necessary performance.

Applications developed with software that uses interprocedural analysis execute more quickly. These compilers produce applications that execute more quickly because they tailor procedures to the way they are actually used.

Applications developed with software that uses interprocedural analysis is more reliable. Some interprocedural errors can be found by using persnickety languages like Ada. Others can be found by using tools like the *lint* tool that UNIX™ systems offer. But many such errors can only be found by trial-and-error debugging. Because finding these bugs requires looking at multiple procedures, it is more time-consuming than other types of debugging.

EVALUATING TOOLS

The July 1990 issue of *Computer Language* (page 12) had an interesting advertisement. It said the following:

"Compiler Ads are Confusing. They all claim that their products are the fastest and most powerful. Buzz words like optimized, integrated, and modular are everywhere never meaning quite the same thing."

It is true that it can be quite difficult to choose a programming tool, particularly when it offers a new technology. Tools that employ interprocedural analysis will make this process more difficult, particularly because they tend to render completely invalid the standard synthetic benchmarks used to evaluate compilers today!

Listed here are several criteria that should be used in evaluating program development tools that employ interprocedural analysis.

1. Does the tool perform automatic optimization or does it assist you in manual editing of your program source files?

If the tool only performs user-assisted optimization, it is less likely to be effective in reducing development or execution time. Whenever human intervention is required, the potential for error increases. Just as Interprocedural Analysis makes it possible to find subtle errors, interactive tools that use interprocedural analysis make it possible to introduce subtle errors.

Most users of scientific and engineering systems see the computer as the means to get their real job done, not as an end in itself. They often have neither the time nor the expertise to perform optimizations manually.

Choose an automatic tool over one that requires manual intervention.

2. Does the tool perform language independent optimization or is it strictly limited to supporting a single language (FORTRAN)?

If the tool is restricted to a single language, it is less likely to be effective. This is because interprocedural analysis must look at the entire application, and most real applications end up needing to have a few procedures coded in a different language, often C or assembly language. If the tool cannot handle multiple languages, it will make worst-case assumptions about the procedures it cannot analyze, which will severely impair the quality of the information it produces.

In addition, the growing trend to code scientific and engineering applications in C rather than FORTRAN means that a tool limited to FORTRAN will exclude the newest and most important applications.

Choose a language-independent tool over a monolingual one.

3. Does the tool work on any type of terminal or is it strictly limited to operating on bit-mapped graphics workstations?

Many organizations have huge amounts of capital invested in video terminals and workstations. If the tool doesn't work on video

terminals, a large fraction of potential users may be shut out. Organizations are less likely to investigate the benefits of a new technology like interprocedural analysis if they are forced to make huge capital outlays to acquire a particular type of workstation on which a tool runs.

Choose an I/O device-independent tool over one that requires a particular terminal or workstation.

4. Does the tool benefit all types of applications or does it only benefit applications that can be made to run in parallel?

Interprocedural analysis certainly can help in restructuring applications to use high-level parallelism on a MIMD architecture. There are many applications, however, that have no parallelism at all, or that only have low-level parallelism exploitable on a pipelined vector processor or SIMD architecture. Since interprocedural analysis can help in optimizing all three types of applications, it would be foolish to use a tool that only applies interprocedural information on one type of application.

Choose a tool that is useful for all system architectures over one that is helpful only on parallel systems and applications.

5. Does the tool coexist with current development tools you use or does it requires you to learn their editor, debugger, profiler, and source control system?

The choice of an editor is a very emotional issue with most programmers. Organizations trying to introduce tools that use interprocedural analysis will encounter resistance if such tools require programmers to change the way they do their work. Even if they are willing to change, the loss in productivity while they are learning a different way of doing things they already know how to do should not be regarded lightly. If the tool also requires them to learn a new debugger, profiler, and source control system, the productivity loss will be even more severe.

Choose a tool that lets you use your current development tools over one that forces you to learn a new set of tools.

6. Does the tool use execution profile information?

There are a number of optimizations that can be improved by using execution profile information. If a tool doesn't use this information when available, it is leaving performance improvements on the table.

Choose a tool that uses both compile-time and run-time information over one that uses only compile-time information.

7. Does the tool have a user interface that is easy to learn?

There are many ways to build in ease-of-use. Here are some questions to help you evaluate a software tool: Can you learn all you need to know in an hour? Is it similar to an existing tool that you already know how to use? If it is interactive, does it use menus, a command language, or both? Does it provide the means to customize its behavior to your needs?

Choose a tool that you find easy to work with.

8. Is it an integrated tool, or does it rely on other software systems?

Much of the data generated during interprocedural analysis cannot be easily converted to a human-readable form. If you are working with a tool that does source-to-source transformations, some of the information it deduces will be lost to subsequent tools, or will have to be regenerated.

This will result either in less effective optimization, or in slower tools.

Choose an integrated tool over a source-to-source converter.

LOOKING INTO THE FUTURE

Interprocedural analysis opens up a whole new world of optimizations andprogram development tools. This section explores some of the possibilities for the future.

Automatic parallelization of loops containing calls is the next step beyond parallelization assistance. It will take advantage of precise array analysis to determine which loops can be run in parallel, and perform code transformations needed to make those loops run concurrently.

The results of array analysis can also be used to determine how to distribute arrays across the memories of distributed memory parallel processors. This optimization is critical to minimize communication on such processors.

The next step beyond constant propagation is predicate propagation. A predicate is a statement that has a TRUE or FALSE value. Constant propagation is a special case of predicate propagation, where the predicate is 'variable always equals constant' for some variable and some constant.

Researchers have been looking into interprocedural analysis as a debugging aid for several years. One idea that seems useful and practical is the notion of 'slicing'. Programmers normally look at their applications horizontally — on a procedure-by-procedure basis. It is clear that it would be quite useful if tools were available that displayed an application vertically — showing all the code related to a given data structure, or all the code that contributes to the value assigned to a variable, and so forth.

Another logical idea for applying interprocedural analysis is a tool that produces reverse engineering documentation. Programmers have used tools for decades that generate information such as cross-reference tables and call graphs. Interprocedural analysis offers the opportunity to create similar tools with more detailed and sophisticated information.

Interprocedural analysis should also open the way for a whole new set of tools that identify sections of code that possibly or even certainly contain logic errors. Such tools will range from interprocedural measures of design and code complexity to those that use the results of predicate propagation to find code that operates under mutually conflicting assumptions.

As products that use interprocedural analysis become more common in themarketplace, programmers can look forward to a whole new set of software tools and techniques that promise even more productivity.

[1]"Interprocedural Compilation of Fortran D for MIMD Distributed Memory Machines", M. Hall, S. Hiranandani, K. Kennedy, C. Tseng, Proceedings of Supercomputing '92, pp.522-534.

For more information or for the location of your local Convex sales office, contact:

U.S.A. CORPORATE HEADQUARTERS

Convex Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, Texas 75083-3851
Phone: 214/497-4000

EUROPEAN HEADQUARTERS

Convex Computer, Ltd.
Randalls Research Park
Randalls Way
Leatherhead
Surrey, U.K. KT22 7TS
Phone: 44-372-386696

FAR EAST

Convex Computer PTE, Ltd.
1 Scotts Road
#25-06 Shaw Centre
Singapore 0922
Phone: 65-733-4355

AUSTRALIA

Convex Computer PTY, Ltd.
Level 20, Como Office Tower
Chapel Street
South Yarra, Victoria 3141
Phone: 61-3-823-6216

080-002250-000