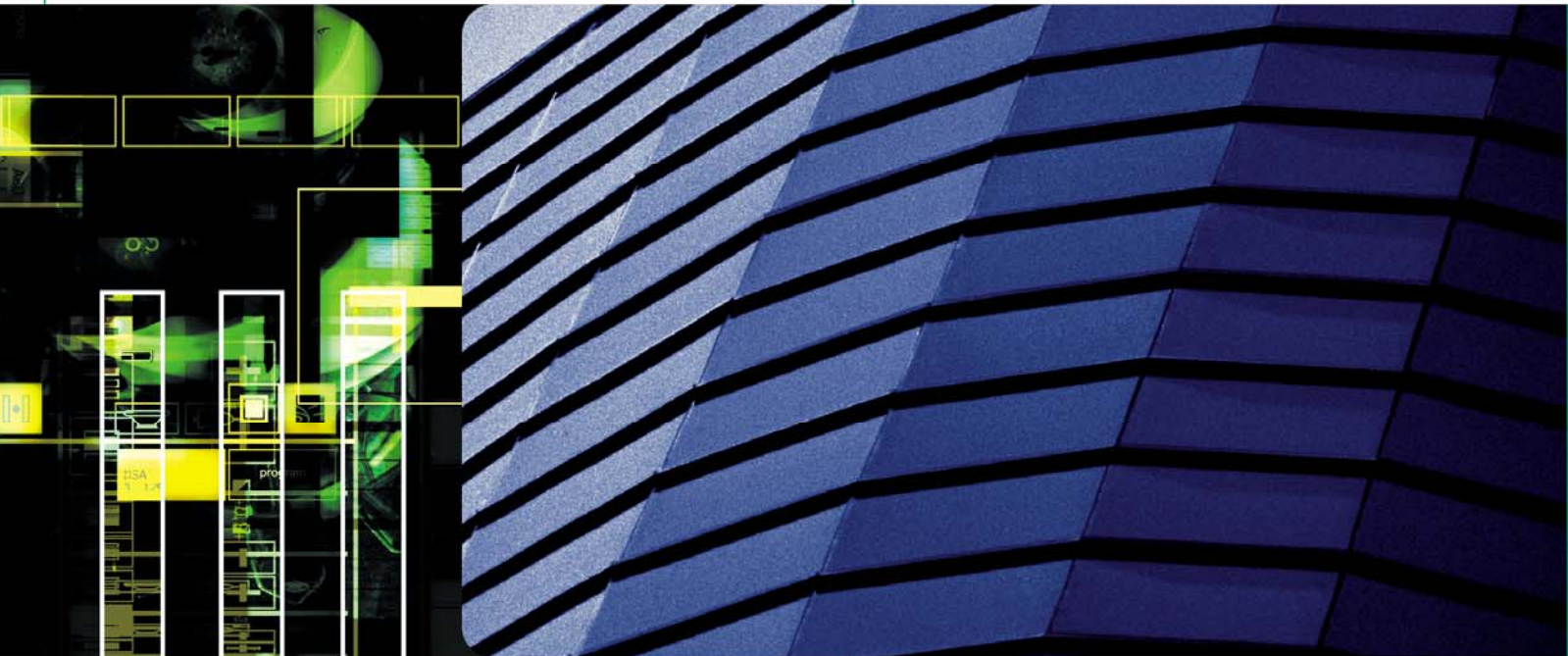




Accelerating  
Cluster Performance™

# Maximizing Application Performance Through Inter-Procedural Optimization with the PathScale EKOPath Compiler Suite



by DR. FRED CHOW  
Director of Compiler  
Engineering  
PathScale, Inc.



Developers are constantly seeking new ways to improve application performance. While architecture and algorithms are important factors, development tools can also have a significant impact on application performance. Traditional compilers perform separate compilation on each program source file, forcing them to make worst-case assumptions that result in poor application performance. Developers need more sophisticated tools that can take advantage of all program information to better apply analysis and optimization techniques and create high performance applications.

## **The PathScale EKOPath Compiler Suite — Optimizing Application Performance**

The PathScale EKOPath Compiler Suite is a family of compilers for the AMD64 processor family. Taking advantage of the high performance and 64-bit features of AMD64 processors, the PathScale EKOPath Compilers provide industry-leading application performance for AMD Opteron-based systems. Utilizing advanced processor features, including complex addressing modes, large register sets, more efficient parameter passing, and SSE2 support, the PathScale EKOPath Compiler Suite generates code that simply performs better on these systems.

The PathScale compilers provide 100% binary compatibility, with the ability to mix and match the linking of GNU and PathScale compiled libraries and objects. The front ends are source compatible with the GNU compiler suite for C/C++, while the FORTRAN 95 compiler provides support for the most common Cray/SGI extensions. In addition, the PathScale EKOPath Compiler Suite is available in installable Linux RPM format, and is verified on SuSE, Red Hat, and Fedora Linux distributions.

Key features of the PathScale EKOPath Compiler Suite include:

- C, C++, and Fortran 77/90/95 compilers
- Industry-leading optimizations
- Complete support for 64-bit compilation
- Complete support for 32-bit compilation on x86-based systems
- Code generation for AMD64 ABI and AMD Opteron
- Compatibility with GNU/gcc tool chain and debuggers
- Tested with popular third-party debuggers
- Binary and source code compatibility

## Inter-Procedural Analysis — More Information, Better Decisions

Software applications are typically comprised of multiple source files that are compiled separately and linked together to create an executable program. This traditional approach, known as *separate compilation*, limits most commercial compilers. Incomplete program information is available during compilation, forcing compilers to make worst-case assumptions about programs that access external data or call external functions. The PathScale Compilers solve this problem by performing *whole program optimization*, enabling them to make intelligent decisions on where and how to perform various optimizations, resulting in higher performing applications. By collecting information over the entire program, the PathScale compilers can make better decisions regarding the applicability and safety of many optimization techniques. The result — a wider variety of optimization techniques used to greater effectiveness.

Whole program optimization is performed in the PathScale EKOPath compilers through a technique called *inter-procedural analysis*. Inter-procedural analysis links all source files together early in the compilation process — before most optimization and code generation is performed. Using intermediate representation files, the PathScale compilers perform inter-procedural analysis on the entire program, invoke the back-end of the compilers to optimize and generate object code, and finally invoke the standard linker to produce the final executable (Figure 1).

### Analysis and Optimization

Inter-procedural analysis occurs in two steps: analysis and optimization. In the analysis stage, information is collected for the whole program. Application source code is analyzed on several levels, and the information gathered is used during the optimization stage to make intelligent decisions on how best to transform the application source code and achieve the best possible performance.

The first step in analyzing application source code is understanding how different functions relate to one another. To help this effort, the PathScale compilers first construct a *program call graph* — a representation of all functions and their caller/callee relationship. Using the call graph, the compiler determines whether each program variable is modified or referenced inside a function call. Alias information is generated for every variable whose address is taken, enabling pointer accesses to be optimized more aggressively, minimizing their impact on application performance. All variable information gathered during the analysis phase is stored, enabling the back-end of the compiler to perform additional optimization steps.

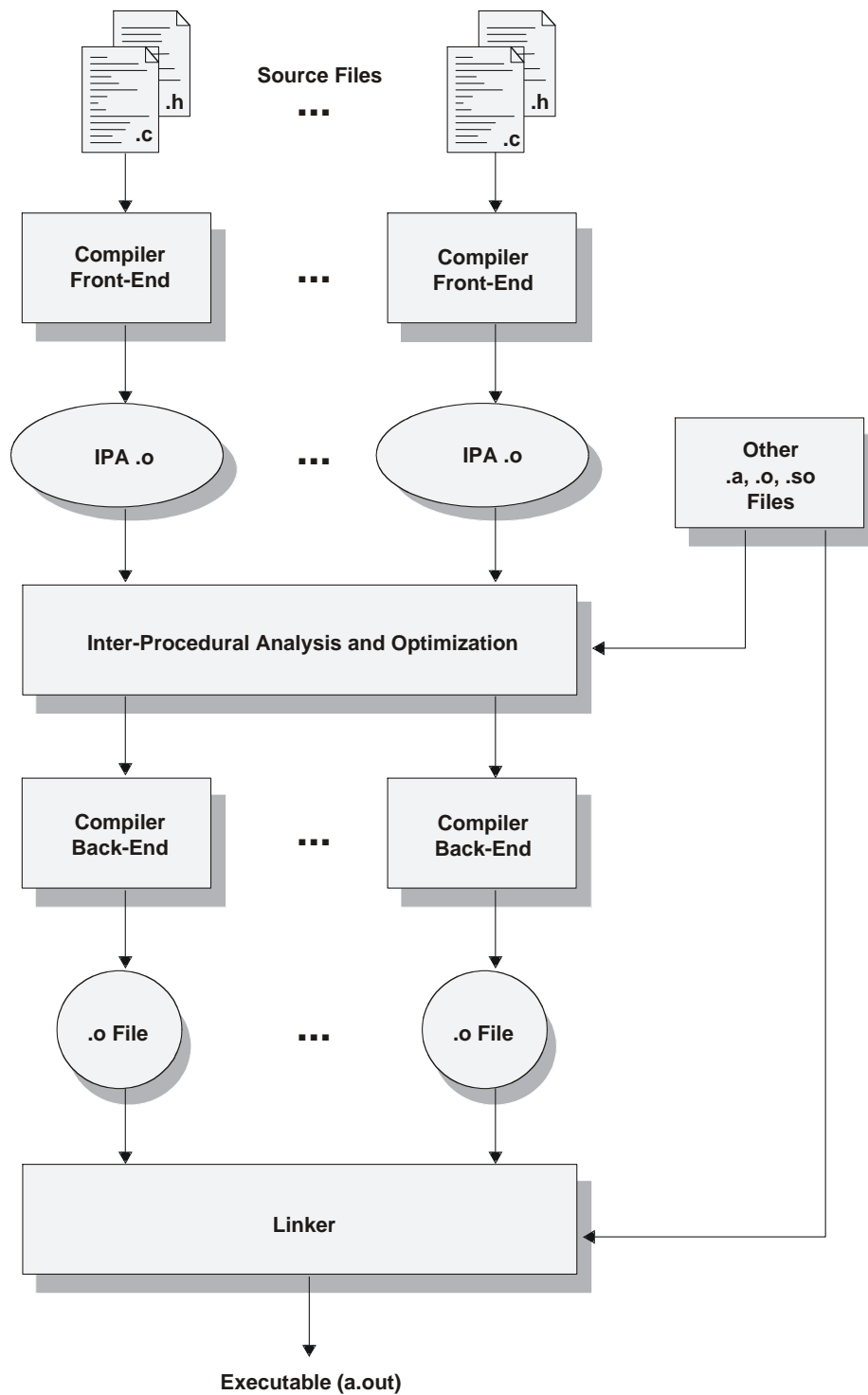


Figure 1. The PathScale compilers use a compilation model built on inter-procedural analysis and optimization to create high performing applications.

Once the application source is analyzed, several types of optimizations are performed automatically by the PathScale compilers:

- *Inlining.* Perhaps the most important inter-procedural analysis performed, inlining replaces calls to a function with the body of the function. Function call overhead is eliminated, and the back-end phases of the compiler are able to work on larger sections of code, potentially enabling the compiler to take advantage of other optimizations that would have been impossible when less code was available. For example, inlining may result in the formation of a loop nest that enables aggressive loop transformations. Inlining is performed with great care in the PathScale compilers in an effort to ensure performance degradation does not result. Large function and program sizes can cause higher instruction cache misses, run out of registers, use memory too frequently, or slow down later stages of the compilation process. Because the PathScale compilers take a whole program compilation approach, they are able to inline any function into any other function, even if they are not located in the same source file. As a result, the PathScale compilers are better able to perform inlining more aggressively than traditional compilers.
- *Constant propagation and function cloning.* Many function calls pass constants, including variable addresses, as parameters. Replacing a formal parameter with a known constant value creates opportunities for optimization. For example, portions of a function often become unreachable and can be deleted as dead code. Constants are exploited in the PathScale compilers in two ways. First, if all calls to a function use a given constant, constant propagation is performed, modifying the function without increasing program size. Second, if a function is called with several constant parameters, function cloning is performed to create alternate functions that use customized parameters.
- *Dead variable elimination.* Dead variable elimination removes all global variables that are never used, as well as the code that updates them, thereby speeding execution.
- *Dead function elimination.* Dead functions — functions that are never used — can result from inlining and cloning techniques, or from continual program modification during the development process. Dead function elimination removes such functions, saving valuable space, and reducing memory and cache consumption.
- *Common padding.* Common padding improves array alignments in FORTRAN common blocks. Unlike traditional compilers that are unable to coordinate changes to the layout of user variables in a common block, the PathScale compilers can take advantage of all subroutines being available. By implementing common padding, array alignments can be improved, enabling arrays to be vectorized and accessed more efficiently and potentially reducing cache conflicts, improving performance. A similar technique can rearrange C and C++ structures.
- *Common block splitting.* Common block splitting divides a FORTRAN common block into smaller pieces, reducing data cache conflicts during program execution.
- *Procedure re-ordering.* Procedure reordering organizes functions based on their call relationship, potentially reducing instruction cache thrashing during program execution.

## Delivering High Performance Applications

The PathScale EKOPath Compiler Suite is proving to be the highest performing set of 64-bit compilers for AMD-based Opteron systems running the Linux operating system. Indeed, the PathScale compilers produced the highest performance results for both integer and floating-point SPEC® CPU2000 speed benchmarks for any AMD64-based Linux system (Table 1).

Benchmark	Result
SPECfp® 2000	2267
SPECfp_base2000	2086
SPECint® 2000	1929
SPECint_base2000	1745

*Table 1 — The PathScale EKOPath Compilers provide outstanding performance on standardized benchmarks<sup>1</sup>*

### SPEC CPU2000 Benchmark

The SPEC CPU2000 benchmark is an industry-standard test suite aimed at measuring compute-intensive performance for real-world applications. A CPU-intensive set of tests, the SPEC CPU2000 benchmark measures processor, memory, and compiler performance. The SPEC CPU2000 benchmark suite consists of two groups of tests: CINT2000 and CFP2000. The CINT2000 tests include applications that stress integer performance, such as compression software, circuit routing and placement, word processing, compilers, and more. The CFP2000 tests include applications that stress floating-point performance, including finite element analysis, 3-D graphics, solving partial differential equations, image recognition, and other scientific applications. Each set of tests is run with a single set of uniform compiler options specified (referred to as “base” result) as well as with individualized optimization flags (referred to as “peak” result).

To demonstrate the profound effect inter-procedural analysis can have on application performance, PathScale ran the CPU2000 benchmark suite compiled using Release 1.2 of the PathScale EKOPath Compiler on a 2.2 Ghz Opteron-based system configured with 64 KB on-chip primary cache, 1 MB on-chip secondary cache, and 1 GB of main memory.

---

<sup>1</sup> The top SPEC CPU2000 speed results at [www.spec.org](http://www.spec.org) on AMD64-based Linux systems as of Nov. 9, 2005 were obtained using PathScale EKOPath compilers on two systems: a) HP Proliant DL385 with 2.8 GHz AMD Opteron™ CPUs achieved SPECfp2000 = 2267 b) AMD submissions using 2.8 GHz AMD Athlon™ 64 FX-57 CPUs on a ASUS A8N-SLI motherboard, achieved SPECfp\_base2000 = 2086; SPECint2000 = 1929 and SPECint\_base2000 = 1745.

Figure 2 illustrates the effects of inter-procedural analysis on the “base” tests of the CPU2000 benchmark. In these tests, all compilations use the same four optimization options, providing a test suited to users who are not interested in finding the peak flags for a given application. The PathScale inter-procedural analysis system improves the performance of 17 out of 26 benchmarks, with up to 26.6% improvement in some cases, and an average (geometric mean) improvement of 6 percent. Use of additional inter-procedural tuning flags can enable further improvement in all these programs.

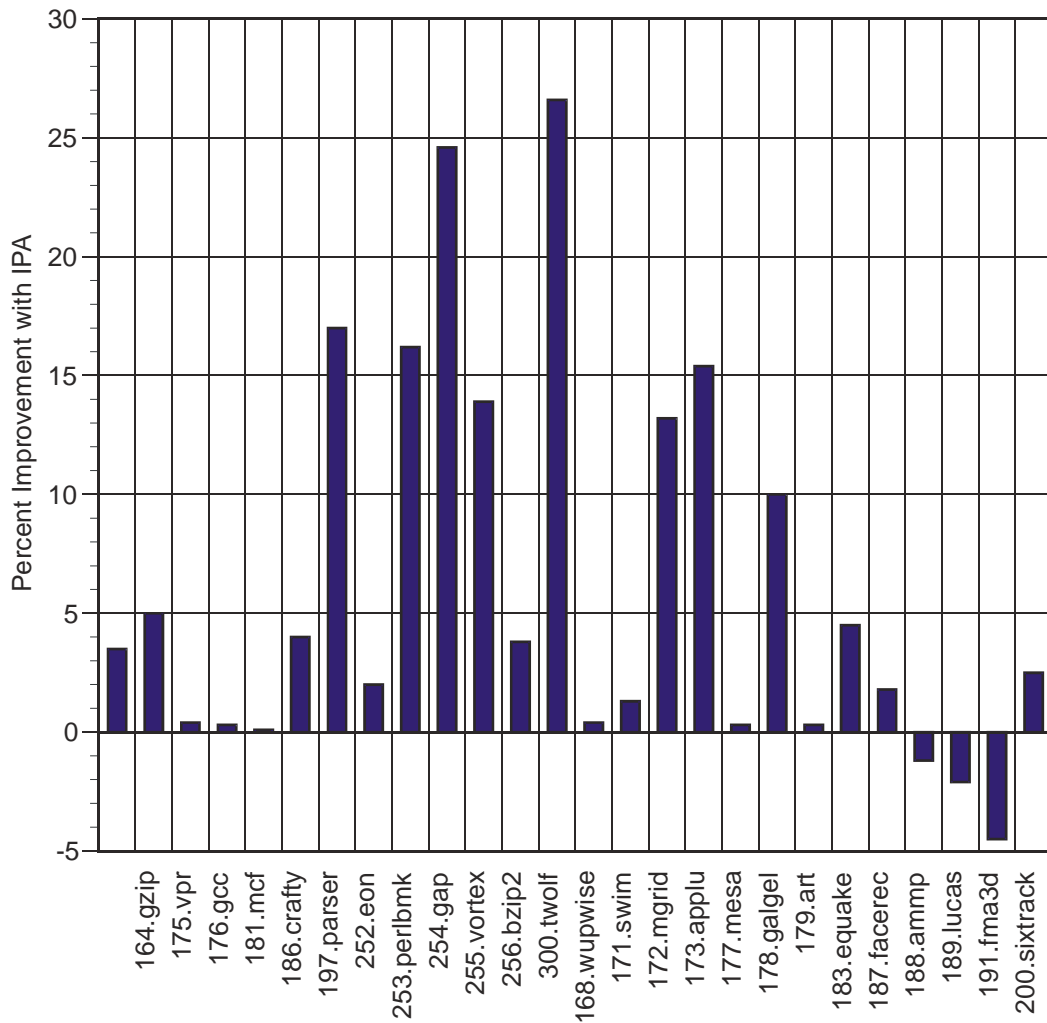


Figure 2 — The inter-procedural analysis features of the PathScale EKOPath Compilers provide outstanding improvement on CPU2000 “base” benchmarks

## Getting Maximum Performance with the PathScale EKOPath Compilers

Dedicated to making application performance the highest priority, the PathScale compilers make using inter-procedural analysis easy. Developers only need to compile and link with the `-ipa` option to begin taking advantage of the performance benefits of inter-procedural analysis and optimization. When the `-ipa` option is specified, the PathScale compilers compile program code and link their intermediate form, perform inter-procedural analysis and optimization, generate object code, and invoke the standard linker to produce the executable — all in a single, easy step.

### Tuning for Maximum Performance

While the innovative, cross-file optimization heuristics employed by the PathScale EKOPath compilers are designed to result in the best possible performance, developers may wish to tune applications for peak performance. To aid this effort, the PathScale EKOPath compilers provide several options that can further improve application performance (Table 1).

Option	Description	Notes
-INLINE:aggressive=ON	Increase inlining aggressiveness	Enables more non-leaf and out-of-loop calls to be inlined
-INLINE:all	Perform all possible inlining	Use only if the program is small
-INLINE:callee_limit=n	Functions whose size exceeds specified limit should not be automatically inlined	Default is 500
-INLINE:list	List inlining actions on-the-fly	Identifies functions inlined
-INLINE:must=name[,name,...]	Force inlining of specified functions	
-INLINE:never= name[,name,...]	Suppress inlining of specified functions	
-INLINE:none	Turn off automatic inlining	Language inlining is still performed
-IPA:forcedepth=n	Inline all routines at or below the specified depth in the call tree	Performed regardless of space limitations
-IPA:inline=off	Suppress invocation of the inliner	Default is ON
-IPA:maxdepth=n	Inline all routines at or below the specified depth in the call tree	Performed subject to space limitations
-IPA:min_hotness=n	Specify that a call site's invocation count must exceed a specified number before it can be inlined (applicable during feedback compilation only)	Call site's invocation count must exceed n before it can be inlined by IPA
-IPA:multi_clone=n	Specify the maximum number of clones that can be created from a single function	
-IPA:node_bloat=n	Specify the maximum percentage growth in the number of procedures relative to the original program during cloning	
-IPA:plimit=n	Suppress inlining into a function once a specified size (in blocks) is reached	Default is 2500
-IPA:callee_limit=n	Indicate that functions exceeding a specified size limit are never to be automatically inlined	Default is 500
-IPA:small_pu=n	Indicate that functions smaller than a specified number of blocks are not subject to the -IPA:plimit restriction	Default is 30
-IPA:space=n	Inline until a specified percent factor increase in code size is reached	Default is 100; Higher values can be specified for small programs
-IPA:specfile=filename	Specify a file containing additional inlining options	
-OPT:0limit=n	Specify the maximum allowed size, in blocks, for a function after inlining	Default is 6000 with -O2, 9000 with -O3; A value of zero imposes no limit

*Table 1 — The PathScale EKOPath Compilers provide a number of options to help developers tune applications for even higher performance*

A

Additional options unrelated to inlining and cloning may prove useful when tuning applications and studying the effects of optimizations. (Table 2).

Option	Description
-IPA:alias=OFF	Disable alias and mod-ref analyses
-IPA:addressing=OFF	Disable address-taken analysis
-IPA:cgi=OFF	Disable constant propagation for global variables
-IPA:common_pad_size= n	Specify the pad size for common block padding
-IPA:cprop=OFF	Disable constant propagation for parameters
-IPA:dfe=OFF	Disable dead function elimination
-IPA:dve=OFF	Disable dead variable elimination
-IPA:linear	Enable linearization of array references
-IPA:pu_reorder	Enable or disable procedure re-ordering optimizations
-IPA:field_reorder	Enable field reordering optimization to minimize data cache misses
-IPA:split=OFF	Disable common block splitting

Table 2 — The PathScale EKOPath Compilers provide additional tuning options

## PathScale — Committed to High Performance

At PathScale, we take performance seriously, and are committed to providing the tools needed to accelerate the development and deployment of high performance applications. With more than 25 years of professional experience, the PathScale team is dedicated to finding new and exciting ways to use technology advancements to greater advantage. With 64-bit C, C++ and Fortran 77/90/95 compilers that perform industry-leading optimizations, the PathScale EKOPath Compiler Suite lets developers take advantage of processor-specific features and accelerate application performance for AMD Opteron-based systems. The result — increasing application functionality and improved performance that does not sacrifice time-to-market. Indeed, with PathScale, developers can rest assured that they have access to the best development tools possible.

### For More Information

Additional information on PathScale, Inc. and its products can be obtained by visiting <http://www.pathscale.com> on the World Wide Web.



PathScale Inc.  
2071 Stierlin Ct. Suite 200  
Mountain View CA  
94043 USA

tel: 650.934.8100  
fax: 650.428.4282

© 2005. All rights reserved. PathScale, InfiniPath and the PathScale logo are trademarks of PathScale, Inc. All other trademarks are the property of their respective owners. [www.pathscale.com](http://www.pathscale.com)