

# Parallelnavi Workbench: an OpenMP Development Environment for a Wide-Area Network

*Matthijs van Waveren and Pierre Lagier  
Fujitsu Systems Europe Ltd  
8, rue Maryse Hilsz  
31500 Toulouse, France*

*Minoru Tanaka, Akira Ohwada, and Naoki Sueyasu  
Compiler Development Department, Software Technology Development Division,  
Software Group, Fujitsu Ltd.  
140 Miyamoto  
Numazu-shi, Shizuoka 410-0396, Japan*

## Abstract

Parallelnavi Workbench is an OpenMP development environment that can be deployed on a Wide Area Network (WAN). It is based on a client/server/executor architecture. The user has a client running on his local workstation, and this client allows him to access a Workbench Server located anywhere on the Internet. He will be able to execute OpenMP programs on any executor machine. The server is the entry point to the Parallelnavi environment. The development environment gives the user access to a debugger, a tuning tool, and a profiler, as well as an OpenMP code generator, and a static analyzer. This paper describes the architecture of the development environment and the tools that are provided.

Keywords: OpenMP, development environment, client/server, WAN

## 1. Introduction

Parallelnavi Workbench is an OpenMP development environment that can be deployed on a Wide-Area Network (WAN). A WAN is a computer network that spans a relatively large geographical area. Typically, a WAN consists of two or more local-area networks (LANs). Parallelnavi Workbench is based on a client/server/executor architecture. The user has a client running on his local workstation, and this client allows him to access a Workbench Server located anywhere on the Internet. He will be able to execute OpenMP programs on any executor machine. The server is the entry point to the Parallelnavi environment. The development environment gives the user access to a debugger, a tuning tool, and a profiler, as well as an OpenMP code generator, and a static analyzer. This paper describes the architecture of the development environment and the tools that are provided.

In Section 2, we describe the client/server/executor architecture of the development environment. The environment is very flexible, insofar that clients, executors, and the server can be located on any machine on the Internet.

In Section 3, we describe the tools that are provided as part of Workbench. These include OpenMP Fortran, C, and C++ 2.0 compilers [1,2], a full and a light-weight debugger, a tuning tool, and a profiler, as well as an OpenMP code generator, and a static analyzer. Parallelization of the WORKSHARE construct in the OpenMP Fortran 2.0 compiler is attained using selective loop fusion [3]. The compilers support nested parallelism.

## 2. Parallelnavi Workbench

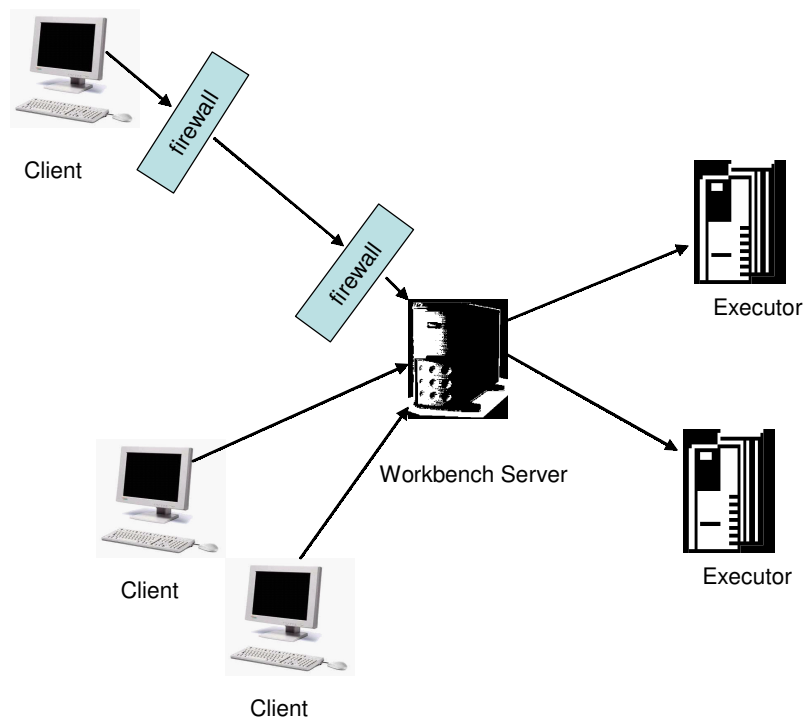
### 2.1 Network-based development environment

Parallelnavi Workbench is a development environment that is deployed on a WAN. It is based on a client/server/executor architecture. A typical installation is shown in Figure 1. The components shown are:

- Server that is used as entry point to the development environment. This server holds the repository database, and the workspaces of the clients.
- Client, that is located on the local workstation of the user. This client can connect to the Workbench server from any location.
- Executor, that is the machine where the generated executables will be executed.

Each of the components can be installed on different machines, and the client-server connection can go through a firewall.

**Figure 1: System Architecture**



Parallelnavi Workbench uses the Java programming language for the client/server/executor architecture. This means that the Workbench components can be run on any platform supporting Java. The client and server components are java processes that communicate via sockets. When a computer program needs to connect to a local or wide area network such as the Internet, it uses a software component called a socket. The socket opens the network connection for the program, allowing data to be read and written over the network. It is important to note that these sockets are software, not hardware, like a wall socket. The client, the server and the executor can be on different platforms, or on the same platform.

The Parallelnavi Workbench server manages a common database called the Repository which synchronizes all client activities. Each client communicates with the server for doing day to day development tasks as editing, compiling, linking, ... on the workspaces. Each user has his own workspace directory on the server, where he stores his source code. Generated programs can be run on any remote machine. The input files can be stored either on the user's workspace or on the executor machine.

It is possible to specify directories to which a user has access on the Workbench server. This brings an extra level of security. The environment contains a file explorer, which allows the user to seamlessly transfer files between his client and his workspace and other directories to which he has access on the server.

## 2.2 Tools

The Parallelnavi Workbench programming environment provides an integrated development environment for building, editing, debugging, source browsing, and tuning application software development projects.

Parallelnavi Workbench provides two types of built-in services: basic services and tool services. Basic services are self contained services while tool services are services able to communicate with an existing external tool (a debugger, a profiler, etc) using a message protocol.

Adopting this type of architecture has the following advantages:

- Portability - very little code must be ported to run on another platform
- Efficiency - only small string messages are sent across the network between the end user and the remote systems, keeping network traffic to a minimum.
- Easy integration of existing tools such as compilers, debuggers, profilers, source code manager, etc...

Tool services

- OpenMP Fortran, C, and C++ 2.0 compilers [1-3]
- An OpenMP code generator
- A static analyser
- A debugger
- A profiler
- A version control service

Basic services

- An application builder service to develop and maintain applications
- A text editor service
- A help service
- Team development service including tools for merging source file versions

## 3. Tool Descriptions

### 3.1 Creation of OpenMP source code

The tool supports automatic parallelization of DO-loops (including nested DO-loops) and array assignments. A DO-loop may be sliced into several DO-loops. When nested loops are sliced, the system attempts to parallelize the outermost loop if it can. Therefore, the system selects the DO loop that can be sliced most cheaply and interchanges it with the outermost possible loop. DO-loops in sequence having the same DO-loop control can be merged into a single DO-loop. Loop reduction optimization is supported.

The following OpenMP directives are output by this function:

- PARALLEL DO
- FIRSTPRIVATE(*var\_name*,...)
- LASTPRIVATE(*var\_name*,...)
- PRIVATE(*var\_name*,...)
- REDUCTION(+:*var\_name*)
- REDUCTION(-:*var\_name*)
- REDUCTION(\*:*var\_name*)
- REDUCTION(MAX:*var\_name*)
- REDUCTION(MIN:*var\_name*)
- REDUCTION(.OR.:*var\_name*)
- REDUCTION(.AND.:*var\_name*)

### 3.2 Static Analyzer

The tool makes a static analysis of the user code. The parallel structure of the user code is displayed for the user, as shown in Figure 2. The static analyzer also displays information on procedure call relationships, assembler instruction occurrence counts, parallelization information on loops and lines, and the PRIVATE access range of variables.

**Figure 2: Static Analysis example**

Inc	Line	Nest	o	p	Statement
	171	1			!\$omp parallel
	172				
	173	1			!\$omp do
	174	2	p		do j=1,m
	175	3	u	p	do i=1,n
	176	3	u	p	uold(i,j) = u(i,j)
	177	3	u	p	enddo
	178	2		p	enddo
	179				
	180				* Compute stencil, residual, & update
	181				
	182	1			!\$omp do private(resid) reduction(+:error)
	183	2	p		do j = 2,m-1
	184	3	u	p	do i = 2,n-1
	185	3	u	p	* Evaluate residual
	186	3	u	p	resid = (ax*(uold(i-1,j) + uold(i+1,j))
	187				+ ay*(uold(i,j-1) + uold(i,j+1))
	188				+ b * uold(i,j) - f(i,j))/b
	189				* Update solution
	190	3	u	p	u(i,j) = uold(i,j) - omega * resid
	191				* Accumulate residual error
	192	3	u	p	error = error + resid*resid
	193	3	u	p	end do
	194	2		p	enddo
	195	1	u		!\$omp enddo nowait
	196				
	197	1			!\$omp end parallel

### 3.3 Debugger Function

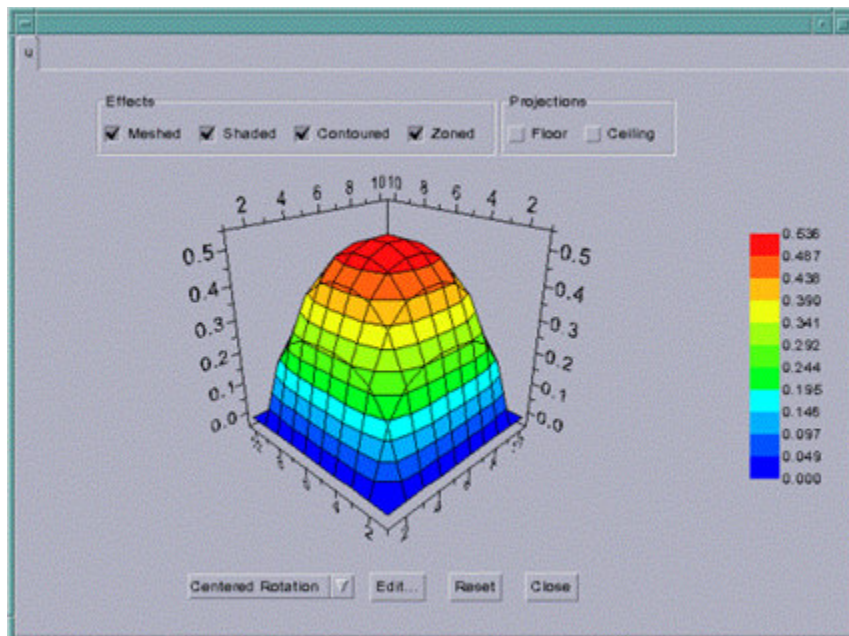
The tool contains both a full debugger and lightweight debugger. The lightweight debugger can be applied to production versions of applications, which have been compiled with full optimization. The full debugger works on applications compiled with the '-g' option, thus without compiler

optimizations. The lightweight debugger will not be able to give the full information given by the full debugger, since line information is lost.

The functionalities of the full debugger are:

- Setting breakpoints, barriers, and watchpoints
- Evaluating and displaying expressions and variables in a graphical format
- Using call stacks
- Debugging of parallel multithreaded applications
- Data Visualization of arrays (See Figure 3).

**Figure 3: Data Visualizer**



### 3.4 Tuning Function

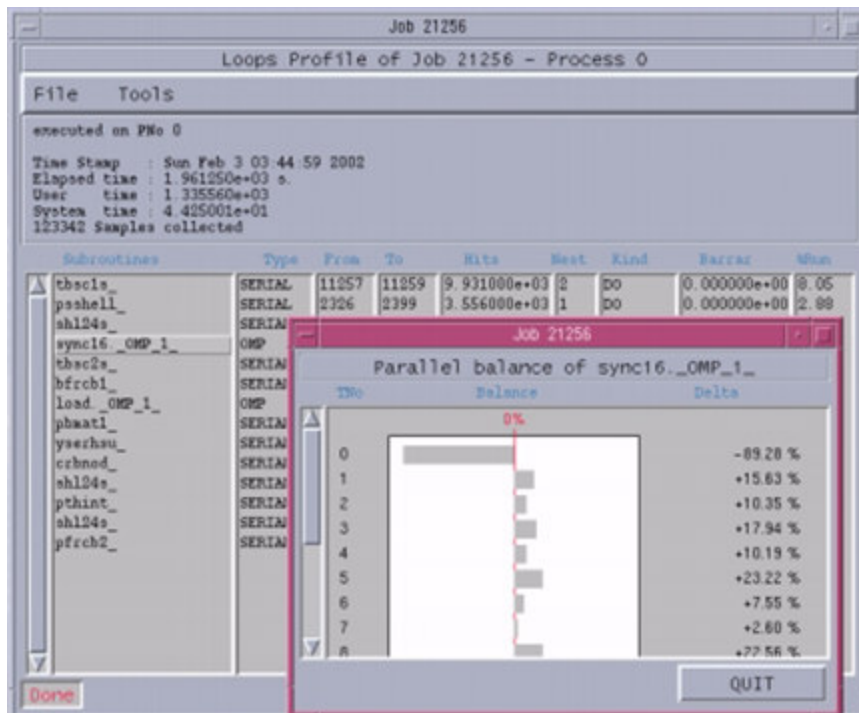
The first step of tuning of a parallel code is to optimize the sequential code. Some of the code sections might not be able to fully utilize the optimization function of the compiler. The sections that block optimization are pointed out by the static analyzer (see Section 3.2). This allows the user to focus on the blocking sections of the code.

#### 3.4.1 Profiler

Using the CPU time and elapsed time measurements of the profiler, the user can find out the bottlenecks of his software, and focus his optimization work on these sections of the code. The profiler provides detailed information on the cost of procedures, loops and lines of code. For parallel

programs, the profiler provides information on work distribution, synchronization wait times between threads, and parallel program overhead. An example of a loop profile is shown in Figure 4. The user can then focus on improving the work distribution, and decreasing the parallel overhead.

**Figure 4: Loop profile**



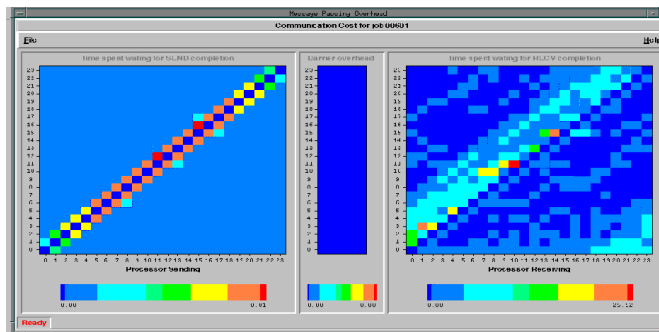
The profiler provides detailed hardware monitor information, so that the user can improve the software to fully utilize the high-speed hardware mechanisms. The hardware monitor information provides information on cache misses, cache contention, and translation lookaside buffer (TLB) misses, so that the user can optimize cache and TLB usage. See Figure 5 for an example of cache miss ratio.

Figure 5: Example of cache miss ration



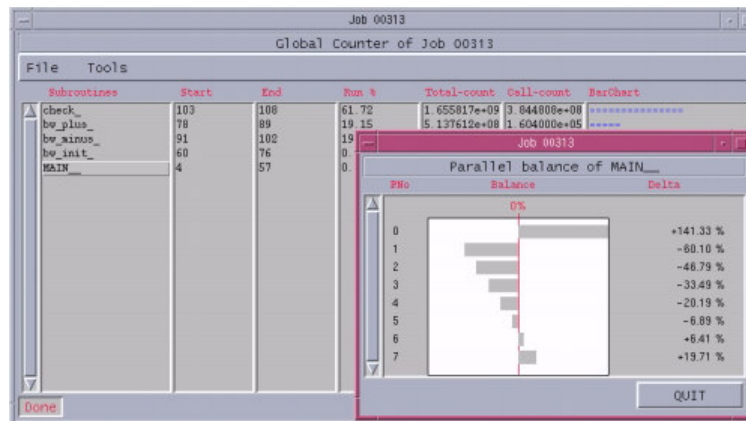
The profile provides information on inter-process communication data transfers (see Figure 6). This communication cost information can be used to reduce the amount of inter-process data transfers.

Figure 6: Communication cost example



### 3.4.2 Counter

**Figure 7: Example of counter at procedure level**



The counter provides execution count information of procedure, loops and lines of a Fortran program. The information is shown in a manner that allows comparison of processes or threads. See Figure 7 for an example at procedure level. This information can be used to identify candidate procedures for inlining, e.g. procedures with a small number of lines of code and a high execution count. It can also be used to improve the program structure (e.g. loop structure, if statements) to decrease the execution count. Optimization work can be focused on loops with a high execution count.

## 4. Conclusions

The combination of client/server technology and an OpenMP development environment leads to a powerful tool, which allows the harnessing of parallel computing resources from anywhere on the Internet. The lightweight debugger supports debugging of production codes, which have been compiled with full optimization. The profiler provides detailed hardware monitor information, which allows the user to fully use the high-speed hardware mechanisms. The OpenMP compilers support nested parallelism, and the OpenMP Fortran 2.0 compiler parallelizes the WORKSHARE construct using selective loop fusion.

## References

- [1] OpenMP Architecture Review Board. *OpenMP Fortran Application Program Interface Version 2.0*, November 2000. (<http://www.openmp.org/specs/mp-documents/fspec20.pdf>)
- [2] OpenMP Architecture Review Board. *OpenMP C/C++ Application Program Interface Version 2.0*, March 2002. (<http://www.openmp.org/specs/mp-documents/cspec20.pdf>)
- [3] H. Iwashita, M. Kaneko, M. Aoki, K. Hotta, and M. van Waveren: *On the Implementation of OpenMP 2.0 Extensions in the Fujitsu PRIMEPOWER compiler*, Proceedings of ISHPC 2003, Lecture Notes in Computer Science, vol. 2858, pp. 523-528, 2003.