

# Interactive Parallelizing Assistance Tool for OpenMP: iPat/OMP

MAKOTO ISHIHARA+ HIROKI HONDA+ TOSHITSUGU YUBA+  
MITSUHISA SATO\*

+ The Graduate School of Information Systems,  
University of Electro-Communications, JAPAN  
{ishihara, honda, yuba}@yuba.is.uec.ac.jp

\* Institute of Information Sciences and Electronics,  
University of Tsukuba, JAPAN  
msato@is.tsukuba.ac.jp

## Abstract

This paper proposes an interactive parallelizing assistance tool for OpenMP called iPat/OMP. This tool provides users with the assistance needed for OpenMP parallelization of a sequential program. All of the assistance capabilities are implemented as Emacs functions and are available in the Emacs editor environment in an interactive manner.

**Keywords:** OpenMP, Interactive parallelizing tool, Parallelizing compiler

## 1 Introduction

There are many tools[1]-[11] available for assisting users in the process of source-to-source program modification for parallelization and/or parallel-performance tuning. The main aim of these tools is to provide a framework for generating a high performance program by coupling the power of automatic parallelizing techniques with the user's application-specific knowledge. To make this coupling more effective, some of these tools are implemented with the editor interface, enabling users to get assistance interactively at a source program editor.

Recently, OpenMP has received much attention as a method for parallel programming, because the OpenMP programming model, directive-base parallelism representation in a source program, can provide capabilities for incremental parallelization and high portability. To facilitate OpenMP programming, some parallelizing translators automatically translate a sequential program into a parallelized one by means of OpenMP. However, the program generated by automatic and mechanical transformation is often very complicated, making it difficult for users to recognize the original functionality from the source program. This phenomenon prohibits the user from making needed modifications on the source program. In addition, such automatic transformation tends to parallelize to too great a degree, which merely results in performance degradation. On the other hand, some parallelizing tools for the OpenMP program have been developed. These tools are very useful not only for generating high performance programs but also for keeping source programs human-readable by deeply involving the user in the process of program parallelization.

In this paper, we propose an interactive parallelizing assistance tool for OpenMP, named iPat/OMP. The tool assists the user in transforming a sequential program into a parallelized one using OpenMP. The tool is

implemented as a set of functions on the Emacs editor. All the activities related to program parallelization, such as selecting a target portion of the program, invoking an assistance command, and modifying the program based on the assistance information shown by the tool, can be handled in the source program editor environment.

The rest of the paper is organized as follows: Section 2 presents an overview of iPat/OMP and discusses its functionality and design concept. An implementation of the tool is illustrated in Section 3. Section 4 shows parallelizing workflow with this tool. Section 5 describes related work and Section 6 concludes the paper.

## 2 Overview of iPat/OMP

### 2.1 Assistance Capabilities

iPat/OMP is designed to have four types of assistance capabilities (Figure 1):

**Parallelism analysis:** analyzing the data/control dependencies in a program section and showing whether or not the section can be parallelized.

**Directive creation:** showing the candidates of the OpenMP directives for the program section which can be parallelized.

**Program restructuring:** restructuring a program section in order to enhance parallelism and parallel execution effectiveness.

**Execution time analysis:** inserting calls for execution time measurement and showing the result of the measurement.

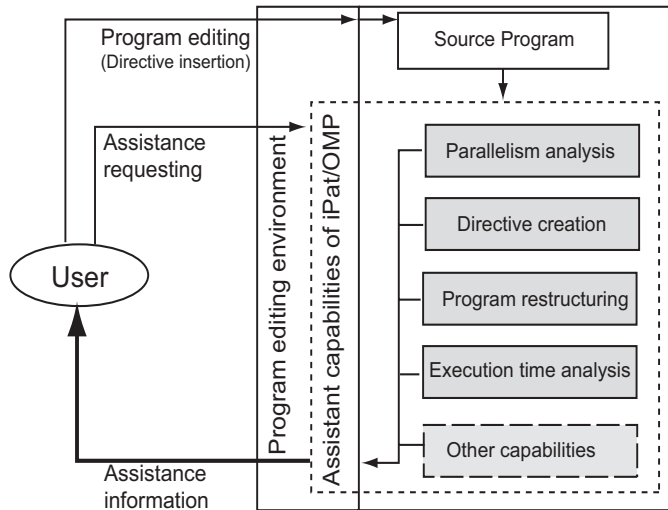


Figure 1. Overview of iPat/OMP.

### 2.2 Design concept

iPat/OMP is designed to have the following characteristics: to be embedded in a commodity editor, to provide text-based interaction, to allow decision-making by the user, to enhance portability and extensibility, and to customize its assistance capabilities according to user's level of parallel programming skill.

**Embedded in a commodity editor.** All the assistance functions are designed to be available in a commodity

editor (Figure 1). The user can get assistance in a familiar environment for source-program editing.

Selecting a commodity editor as a base environment also provides a tool with high portability and expandability. With its high expandability, facilitating the addition of new assistance functionalities, the tool can be used as the platform of a parallelizing assistance tool.

**Text-based interaction.** Interactions between the tool and the user, such as issuing a command, reporting the results of a command, and making the user’s knowledge of program properties available to the tool, are performed by text-based communication.

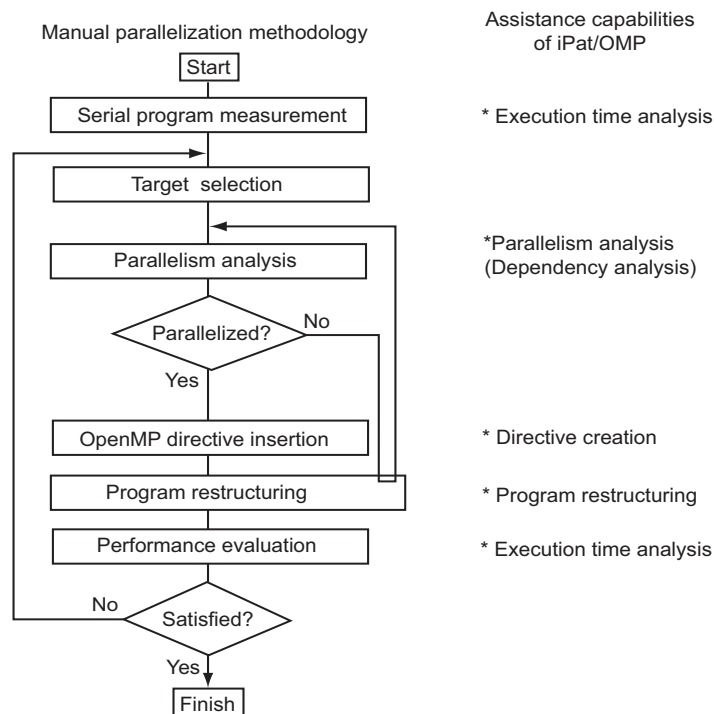
The user’s knowledge of program properties should be written in a source program as directives. The tool acquires knowledge through the directives and makes use of it for program analysis. This method is in accordance with one of the OpenMP concepts which states that required information should be written in the source code explicitly.

**Decision-making by user.** The user is assigned responsibilities for selecting a section of a program as a target of parallelization, issuing a command for an assistance function, and modifying the program manually based on the information provided by the tool. On the other hand, the tool is assigned responsibilities for carrying out the requested function and reporting the result of the function.

By assigning the responsibilities in the form of parallelizing activities as above, the system achieves the advantages of both keeping the program human-readable and having the ability to select appropriate target sections.

**Keeping the program human-readable:** Restricting the tool to modify a program allows a program to be in human-readable form.

**Appropriate target selection:** Manual target selection by the user prohibits unnecessary sections from being parallelized.



**Figure 2. User-Tool role assignment and parallelization methodology.**

**Portability and extendibility.** In order to enhance the portability of iPat/OMP, its assistance capabilities are designed to be implemented without using libraries that run only in a particular OS environment.

Furthermore, in order to enhance the extendibility of iPat/OMP, it is designed with a framework that allows for the easy addition of new analysis methods.

**Target user.** iPat/OMP is designed such that a user is able to customize its assistance capabilities according to his or her level of parallel programming skill, enabling it to serve as an effective tool for beginners as well as experts. For beginners, iPat/OMP indicates which sections of code can be parallelized. It also lists the OpenMP directives which can be used in each section of parallelizable code. For experts, iPat/OMP provides more detailed information, including data dependencies which prohibit parallelization.

## 2.3 Parallelization methodology

iPat/OMP is designed to be based on the parallelization methodology, basically same as one of URSA MINOR / INTERPOL [5], described below (Figure 2).

Step1: Target selection: The user selects a program section which needs to be parallelized. In order for the user to identify a target section, such as a time-consuming loop, iPat/OMP provides an execution time analysis function.

Step2: Dependency analysis: The user leaves the dependency analysis to iPat/OMP.

- If the program is already parallelizable, iPat/OMP shows the required OpenMP directives.
- If the program is not already parallelizable, iPat/OMP reports the obstacles hindering parallelization.

If needed, the user inserts knowledge about the program's properties, which is then used for dependency analysis in the form of directives.

Step3: Program modification: The user modifies the program based on dependency reports generated by iPat/OMP.

- If the program is already parallelizable, the user inserts the required OpenMP directives.
- If the program is not already parallelizable, the user restructures the program.

Step4: Program restructuring: In order to remove the obstacles to parallelization or enhance parallelism, the user may manually restructure the program or ask iPat/OMP to show the recommended restructured code.

Step5: Performance evaluation: iPat/OMP provides an execution time analysis function.

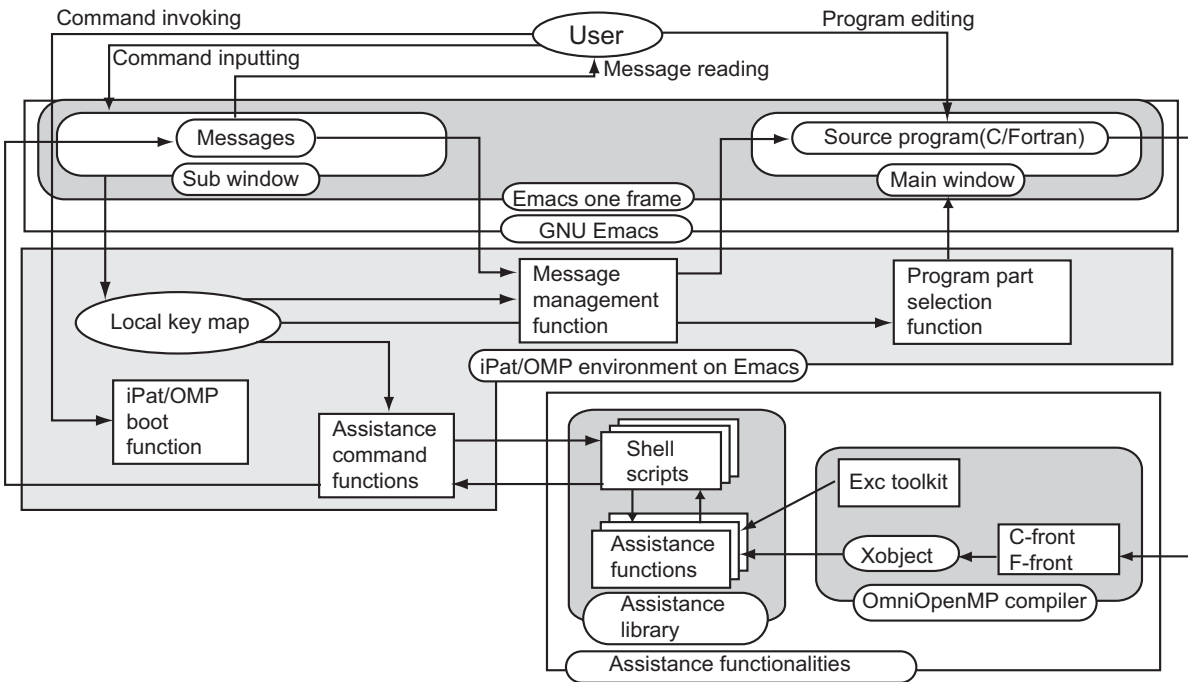
## 3 iPat/OMP embedded in Emacs

We chose GNU Emacs, one of the widely used free-software editors, so that iPat/OMP may have high portability and high extendibility, allowing tool developers to easily add new parallelizing functionalities using EmacsLisp.

iPat/OMP is divided into two subcomponents: one to control the iPat/OMP environment on Emacs and the other to provide a set of assistance functionalities. Figure 3 shows the structure of iPat/OMP.

### 3.1 iPat/OMP environment on Emacs

On Emacs, the iPat/OMP environment is realized as one of the Emacs modes. The properties of the mode are specified by four groups of Emacs functions: the boot function, the assistance command functions, the message management function, and the target selection function.



**Figure 3. Structure of iPat/OMP.**

**Boot function:** The boot function is used to invoke the iPat/OMP environment. In the iPat/OMP environment, an Emacs frame is split into two windows: one is the main window for editing a source program, and the other is the sub-window both for displaying results of assistances and for receiving the user's commands.

**Assistance command functions:** Each assistance function is linked to an assistance command, which is also bound to a local key command, in the iPat/OMP mode. When an assistance command is executed by the user, a shell script for the command will be invoked.

**Message management function:** The result messages from an assistance function are displayed in the sub-window. In the case that the result message contains information related to a program section, the message management function visually points out the section of the source program at the main window. In the case that the result message contains OpenMP directives (directives for Fortran programs), the message management function displays the source program with the directive in the main window if requested to do so by the user.

**Target selection function:** This function enables the user to specify the target program section on which the assistance command is applied. The target can be specified in two ways, described below.

1. The mark function of Emacs.
2. The narrowing function of Emacs.

### 3.2 Assistance functionalities

**Omni OpenMP as the underlying compiler.** iPat/OMP includes the Omni OpenMP compiler[12][13] as an underlying compiler. The compiler has two front-ends, the C-front for C and the F-front for Fortran, each of which translates a source program into an intermediate code described by the Xobject language. In addition, the C-front can translate an Xobject code into a C program.

```

A[0]=0.0;
for(I1=1;I1<1000;I1++)
{
14-> [ ] A[I1]=(double)I1;
15-> [ ] tmp=(A[I1-1]+A[I1])/2.0;
      B[I1-1]=tmp;
}

for(I1=0;I1<1000;I1++)
{
  A[I1]=(double)I1;
}

for(I1=1;I1<100;I1++)

```

```

--:-- Avg.c (C CVS:1.1 Abbrev)--L14--22%-----
Avg.c:14:15: array 'A' : flow or anti dependence.
Avg.c:14:15: array 'A' : flow or anti dependence.
Avg.c:19:#pragma omp parallel for shared(A) lastprivate(I1)
--:%% *messages* (iPat/OMP mode)--L2--Top-----

```

**Figure 4. iPat/OMP environment in Emacs**

Xobject is a human-readable intermediate language having abstract-syntax-tree specification. The Omni OpenMP compiler is equipped with the Exc-toolkit, written in Java, which provides a variety of functions for transformation and analysis of Xobject codes.

**Assistance functionality library.** All of the assistance functionalities are packed into a library. The body of each assistance function is a Java program, which makes use of the libraries in the Exc-toolkit to handle Xobject codes. The program is wrapped by a shell script for the interface with Emacs. Each shell script, which is designed to put its result onto the standard output, can also be executed from a command line, as well as from within the iPat/OMP environment.

As an example of the assistance functions, two functions that realize the fundamental capabilities for assistance in loop parallelization are described below.

**Loop parallelism analysis function:** This function determines whether or not a target loop is parallelizable. This is done by employing a “Power Test” [14] for inter-iterations data dependency analysis on the loop.

**OpenMP directive advisory function:** This function shows the OpenMP directive (a directive for Fortran programs) “*pragma omp for*” with variable attributes, such as “*private*” or “*shared*” clauses, appropriate for a parallelizable loop.

At present, iPat/OMP aids in the creation of user-defined parallel regions by carrying out loop parallelism analysis on for-loops in sequential programs and creating “*#pragma omp for*” and its associated clauses in parallelizable loops.

Figure 4 shows a sample of the iPat/OMP environment on Emacs. The upper part of this Emacs frame is the main window, used as a source editor, and the lower part is a sub-window, used for displaying result information.

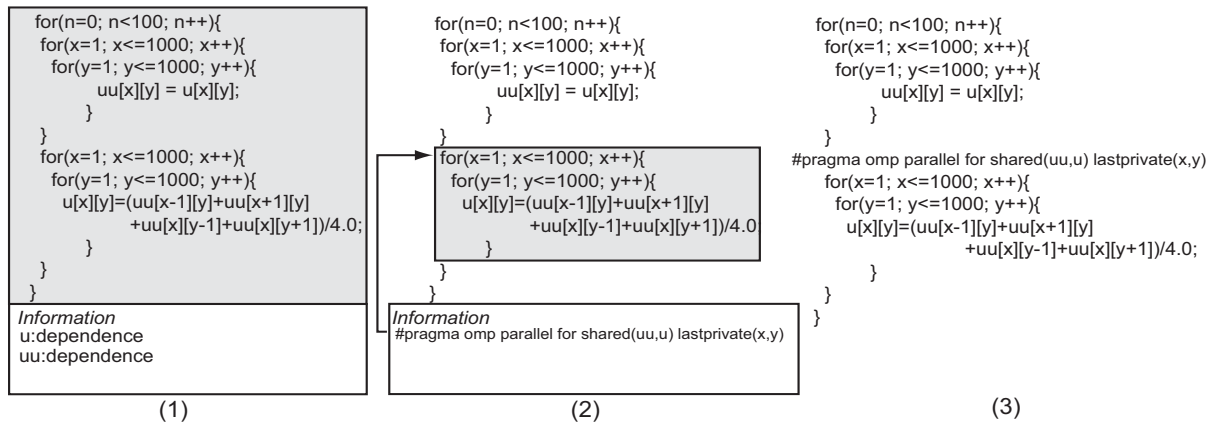


Figure 5. Parallelization workflow using the current implementation of iPat/OMP.

## 4 Parallelizing workflow with iPat/OMP

The following describes the workflow of an iPat/OMP session, the iPat/OMP environment on Emacs, and the current implementation of the assistance functionality library.

- User
  1. When the user requests assistance: The user selects a specific part of the program and invokes parallelism analysis in the sub window of iPat/OMP.
  2. After the user requests assistance, the user carries out two tasks:
    - A) The user executes the message management function and refers to the part pf the program indicated by the assistance information.
    - B) Based on the assistance information in the sub-window, the user decides either to modify the program or to insert OpenMP directives in order to carry out parallelization.
- iPat/OMP environment on Emacs
  1. When assistance starts: The assistance command function invokes a shell script which executes parallelism analysis. And this environment displays information from the standard output of the shell script in the sub window.
  2. After assistance starts: The message management function analyzes the message in the sub window according to the user's command, and this function links information from the messages in the sub-window and the program in the main window.
- Assistance functionality library.
  1. When assistance starts: This library invokes the assistance function after it translates the source program in the main window into an Xobject by using C-front.
  2. The assistance function analyzes the Xobject, and sends the analysis results to the standard output.
    - If the loop is already parallelizable: iPat/OMP presents OpenMP directives to the user via its directive creation capability.
    - If the loop is not parallelizable: iPat/OMP displays the obstacles to parallelism that result from interdependency.

The following describes the parallelization method based on Figure 5.

- (1) The user selects the outer loop and invokes parallelism analysis in iPat/OMP. In this case, iPat/OMP informs the user of data dependencies which inhibit parallelization.
- (2) The user selects the inner loop and invokes parallelism analysis in iPat/OMP. In this case, iPat/OMP displays OpenMP directives with clauses if iPat/OMP determines that the program is parallelizable based on the result of parallelism analysis.
- (3) The user inserts the OpenMP directives presented by iPat/OMP into the selected section of the program.

## 5 Related Work

The ParaScope Editor[3], a successor to the PTOOL interactive browser[4], is a parallelizing assistance tool for transforming a sequential Fortran program into an explicit parallelized one. This tool is one of the pioneers in combining program analysis/transformation facilities with a source editor.

The URSA MINOR / INTERPOL tool set[5] provides a comprehensive parallel programming environment for OpenMP, which realizes a parallel programming methodology using OpenMP.

ParaWise (formerly known as CAPTools)[6] is a parallelizing assistance tool which can handle message passing programs, OpenMP programs, and hybrid programs.

CAPO[7] is OpenMP programming support tool based on the CAPTools. The parallelization process, including dependency analysis, code restructuring, and OpenMP directive insertion, are interactively conducted on a GUI called the directives browser.

PTOPP[8], a tool for performance analysis, makes use of the Emacs editor environment for its capabilities. The design concept of the user/tool interface of iPat/OMP is very similar to that of PTOPP.

## 6 Conclusions

This paper presented our on-going project regarding iPat/OMP, an interactive parallelizing assistance tool for OpenMP.

The parallel assistance capabilities of iPat/OMP are embedded within GNU Emacs, a widely-used commodity editor. Users can use assistance capabilities via the source program editing environment with familiar key operations.

The implementation of iPat/OMP using EmacsLisp, Java, and shell script enhances its portability and expandability. By implementing iPat/OMP by means of a commodity editor environment, we hope that iPat/OMP may be ported onto various platforms, serving as a handy tool for OpenMP parallelization and as a platform of tool development.

We are currently working on the implementation of functions for feeding a user's knowledge to the tool, restructuring human-readable code, directive optimization, and probing for execution time measurement.

## Acknowledgment

This work is supported by the Japan Grants-in-Aid for Scientific Research (Scientific Research, (A), (1), 14208026). We would like to thank the research group members for their helpful suggestions and support.

## References

- [1] J. Brown, A. Geist, C. Pancake, and D. Rover: Software tools for developing parallel applications Part1: code development and debugging. Proc. of 8<sup>th</sup> SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
- [2] J. Brown, A. Geist, C. Pancake, and D. Rover: Software tools for developing parallel applications Part2: interactive control and performance tuning. Proc. of 8<sup>th</sup> SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
- [3] V. Balasundaram, K. Kennedy, U. Kremer, K. McKnley, and J. Subhlok: The ParaScope editor: an interactive parallel programming tool. Proc. International Conference on Supercomputing, pp. 540-550, November 1989.
- [4] J. R. Allen, D. Baumgartner, K. Kennedy, and A. Porterfield: PTOOL: A semi-automatic parallel programming assistant. Proc. of the 1986 International Conference on Parallel Processing, pp. 164-170, August 1986.
- [5] I. Park, M. J. Voss, S. W. Kim, and R. Eigenmann: Parallel programming environment for OpenMP. Scientific Programming, Vol. 9, No. 2/3, pp. 143-161, 2001.
- [6] Parallel Software Products, ParaWise. <http://www.parallelsp.com/>
- [7] NASA, CAPO(Computer-Aided Parallelizer and Optimizer). <http://www.nas.nasa.gov/Tools/CAPO/>
- [8] R. Eigenmann, and P. McClaughry: Practical tools for optimizing parallel programs. The 1993 SCS Multiconference, March 1993.
- [9] Applied Parallel Research, Forge Explorer. <http://www.apri.com/>
- [10] KAI, KAP/PRO toolset. <http://developer.intel.com/software/products/trans/kai/>
- [11] Pallas, VAMPIR: Visualization and Analysis of MPI Resources. <http://www.pallas.com/e/products/vampir/index.htm>
- [12] M. Sato, S. Satoh, K. Kusano, and Y. Tanaka: Design of OpenMP compiler for an SMP cluster. Proc. of EWOMP '99, pp. 32-39, October 1999.
- [13] RWCP: Omni OpenMP Compiler Project, Omni OpenMP Compiler. <http://phase.etl.go.jp/Omni/>
- [14] M. Wolfe, and C.-W. Tseng: The Power test for data dependence. IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 5, pp. 591-601, September 1992.