iCetus: A Semi-automatic Parallel Programming Assistant

Parinaz Barakhshan(✉) and Rudolf Eigenmann

University of Delaware, Newark, DE 19716, USA
{parinazb,eigenman}@udel.edu

Abstract. The iCetus tool is a new interactive parallelizer, providing users with a range of capabilities for the source-to-source transformation of C programs using OpenMP directives in shared memory machines. While the tool can parallelize code fully automatically for non-experts, power users can steer the parallelization process in a menu-driven way. iCetus which is still in its early stages of development is implemented as a web application for easy access, eliminating the need for user installation and updates. The tool supports the user through all phases of the program transformation process, including program analyses, parallelization, and optimization. The first phase includes both static and dynamic analyses, pointing out loops that represent performance bottlenecks and should be improved. The parallelization phase offers diverse options to cater to different levels of user skills. By displaying compiler analyses results in an interactive manner, iCetus supports the user in pinpointing parallelization impediments and resolving them. During the optimization phase, the programmer can apply successive improvements by editing the program, evaluating the performance, and comparing it to that obtained by previous program versions. iCetus also serves as a learning tool to help users understand important program patterns and their parallelization. In this way, it also helps train the user in writing code that likely yields better performance.

Keywords: Interactive source-to-source compiler · OpenMP parallel programming model · Shared memory architecture · Code optimization · Code parallelization

1 Introduction

With the advent of multi-core architectures, the need to fully utilize the capabilities of a computer system has become a topic of great concern among application developers. Given the difficulties of mastering the skills of manually writing high-quality parallel code, many attempts have been made in the past to automate the process of converting sequential to parallel programs. Despite more than four decades of research in automatic program parallelization and although nearly all of today's computer architectures are parallel, current software engineers still make little use of automatic parallelization tools.

X. Li and S. Chandrasekaran (Eds.): LCPC 2021, LNCS 13181, pp. 18–32, 2022.
https://doi.org/10.1007/978-3-030-99372-6_2

The state-of-the-art parallelizer is a batch-oriented optimizing compiler that offers its users little guidance for and control over its operation, except for a sizeable number of command-line options.

Typically, parallelizing compilers are able to extract parallelism in about one in two science/engineering applications. While this is a success from a science viewpoint, it is unsatisfactory to the end user. It is especially aggravating for the engineer of novel applications, which may not exhibit the regular data structures that parallelization technology learned to optimize well.

What's more, even where the tools succeed in detecting parallelism, mapping this parallelism to a given architecture may introduce overheads that offset the gain of automatic optimization. The result is that users see large performance variations across programs and architectures, ranging from nearly ideal speedup to significant slowdown compared to the original program.

From a compiler point of view this problem has two major reasons:

1. Parallelization techniques are highly complex and user code may obscure parallelism. Furthermore, we demand that compilers perform their optimizations correctly on *all* programs. The latter is different from how we think about parallel programming models. For example, OpenMP permits its users to parallelize a loop even if there is a race condition. It is the user's responsibility if the execution is incorrect. The strict demand for correctness makes parallelizers conservative, bypassing many opportunities for optimization. The demand also prevents transformations that are considered *unsafe*. These are transformations that may produce a different, but user-acceptable result than the original code.
2. Every program transformation introduces overhead. Estimating this overhead is highly complex and depends on characteristics of both the program and the target architecture. Performance models usually include parameters that are only known once the program executes, making it often infeasible for the compiler to decide whether or not an applicable technique is beneficial. The dilemma is that not applying the technique forgoes the optimization opportunity; applying it, may introduce overhead that offsets the gain or, worse, degrades performance.

An additional issue motivating the present work is that teaching the skills of program parallelization lacks educational tools that illustrate concepts, program analyses & transformations, and report performance results in an intuitive way.

How can we work around these problems?

- ***Parallel Programming Models:*** Writing a program using parallel programming models, without automatic parallelization, gives full control to the software engineer. This route may be desirable for experienced programmers but is often prohibitive for domain scientists and engineers focusing on their physics, chemistry, or biology, rather than program parallelization.
- ***Auto-tuning:*** Platforms have been proposed that try many optimization variants for a given program and data sets, picking the best. Doing so can

be extremely time-consuming, due to the combinatorial complexity of trying the many program optimization variants. What's more, tailoring such a platform to a user's specific compilation and execution environment can take a prohibitive number of engineering parameters. As a result, no available parallelizer today offers a general auto-tuning platform.

- ***Hardware Support:*** Hardware solutions can significantly reduce parallelization overhead and enable certain unsafe optimizations. For example, architectures have explored support for instruction-level launch of parallel loops (substantially reducing the loop fork-join cost - a major parallelization overhead), loop-level synchronization (enabling low-overhead parallel execution of loops with dependences), and speculative parallelization (overcoming some of compilers' conservative assumptions). While these techniques are known, engineering trade offs so far have prevented them from becoming part of modern computer architectures.
- ***Interactive Parallelization:*** The approach pursued in this paper is to equip a parallelizing compiler with the ability to interact with the users, involving the user into the decisions that compilers struggle with. The idea is to consider user feedback in program parallelization. The objectives include (1) providing the user with information about how the compiler analyzes, transforms, and parallelizes the program, and (2) creating an interface for controlling program parallelization, based on this feedback. Doing so combines user knowledge and compiler capabilities. This information will also help the programmer to write code that is more amenable to automatic parallelization as well as help the student understand the involved techniques and their interactions.

While there are several early projects exploring interactive parallel optimization, which will be discussed in Sect. 5, to the best of our knowledge, no interactive tool exists that harnesses the power of today's most successful automatic parallelizers. This project builds on the Cetus parallelizer, which has shown to be the most effective, making its capabilities available for interactive use. The paper presents an initial design of iCetus and then discusses and evaluates features requested by an early user community.

The rest of the paper is organized as follows. Section 2 explains automatic parallelization, the opportunity of interactive parallelization, the features of iCetus, and the limitations of the current version of iCetus. Section 3 describes the iCetus implementation. Section 4 evaluates existing as well as proposed iCetus features. Section 5 discusses related work and Sect. 6 presents conclusions.

2 Rationale for the iCetus Interactive Parallelizer and Tool Features

This section provides a brief overview of the capabilities of automatic parallelization (Sect. 2.1) and then describes how the provision of these capabilities in an interactive manner can address the issues described in the introduction (Sect. 2.2). Section 2.3 presents the features of iCetus through an example.

2.1 Automatic Parallelization in Cetus

The iCetus tool is based on the Cetus parallelizing compiler infrastructure [2]. Cetus performs source-to-source translation, converting C source code into equivalent C code, annotated with OpenMP parallel directives.

To do so, Cetus applies a number of compilation passes that we classify into program analysis, parallel loop transformations, and performance optimization techniques. This classification is not strict, serving just the presentation of this paper. Program analysis passes include range analysis, alias analysis, points-to analysis, private variable analysis, reduction variable analysis, induction variable analysis, and data dependence analysis. Parallel loop transformations use the analysis information to determine which loops can safely be executed in parallel, annotate these loops as such (using Cetus-internal pragmas), and transform induction and reduction expressions into their parallel forms, as needed. Performance optimizations deal with the efficient mapping of the identified parallel loops to the target architecture. The involved techniques include loop interchange, tiling, and profitability analysis.

The above description is simplified for the presentation of this paper. Additional passes bring the code into a normalized form for easier analysis and transformation. Also, some passes may be split, such as the actual parallel reduction expressions being inserted only after profitability analysis has determined that the parallel execution of a given loop is beneficial.

Cetus generates a report documenting the passes it has applied and providing details on the operation and findings of the passes. Users can select the verbosity of this report via command line options. The highest verbosity level can generate an extensive optimization report.

2.2 The Opportunity of Interactive Parallelization

Recall from Sect. 1 the key problems of batch-oriented compilation, which are (1) conservative optimizations due to the requirement for absolute correctness, and (2) insufficient knowledge of the compiler for making informed decisions about which optimizations to beneficially apply to which program sections. Section 1 has also expressed the need for intuitive educational instruments. Here, we describe the opportunity for a tool that presents the capabilities of Sect. 2.1 interactively, addressing these challenges.

Correctness and Conservative Assumptions: Two key compiler capabilities in identifying parallelism are data dependence and private variable analysis. If a compiler cannot prove that data accesses are dependence free or variables are private, it conservatively assumes that they are not. Similar holds for other techniques, such as alias analysis, reduction parallelization, and induction variable recognition. What's more, certain loops may be correct in their parallel form, even if dependences provably exist. There may be a race condition that will lead to results that are different from the original sequential program, and different parallel executions may yield different results; but all these results may

be algorithmically correct. An example is a search algorithm that finds a different one of multiple elements, all of which match the search criterion. Compilers must always create sequentially consistent results and thus cannot perform such transformations.

The opportunity for an interactive tool is to present the results of these analyses and then let the user decide what is acceptable. In this way, a data dependence that the compiler cannot disprove or a variable that the compiler cannot privatize can be tagged as such by the user. This is especially useful in the fairly common case of a loop where only a few hard-to-detect data dependence or private variable patterns remain that can be recognized by the user. Cetus' optimization report will be of help in this situation. By selectively showing the remaining dependences of a loop and allowing the user to drill down into the analysis details, an interactive tool can thus help parallelize key loop patterns that batch-oriented compilers are unable to.

Overheads and Profitability: A major reason that an automatically parallelized loop may execute more slowly than the original is that the loop is too small so that the cost of invoking and terminating the parallel activity dominates. Recall that not only is modelling the performance of a loop, transformed with potentially many techniques, highly complex, in most cases the model also includes parameters that depend on data read from a program input file and are thus unknown at compile time. The model could be evaluated at run-time, but such execution itself can introduce excessive overhead. We have observed that even using the seemingly low-overhead OpenMP conditional parallel loop construct (run in parallel if a certain condition holds) can yield low profitability. Transformations that add substantial code to the program, such as reduction parallelization and loop tiling, are especially prone to low profitability.

The opportunity for an interactive tool lies in informing the user about loops where profitability is borderline or needs run-time information. The tool can also disclose high-overhead transformations that have been applied, allowing the user to be the judge on profitability. While advanced users may have information that is not available to the compiler for such judgment, the task can still be arduous.

Another tool opportunity is to offer run-time measurements gained through program execution. The values of critical variables may be evaluated (e.g., the number of iterations of a loop), the execution time of a loop may be measured, or the performance of a serial and parallel code version may be compared. An advanced such scenario would be to "auto-tune" a code section or the entire program. That is, the interactive tool would execute many optimization variants and determine the best.

Educational Instrument: Teaching parallel programming techniques, their correctness, and their automation are highly complex. There are many involved concepts, program analyses that need to be understood, and transformations that need to be grasped. Tools that can illustrate these subjects, show the many aspects of program analyses and transformation with representative examples, and allow the student to play with what-if scenarios, can improve the learning experience tremendously.

2.3 iCetus Features

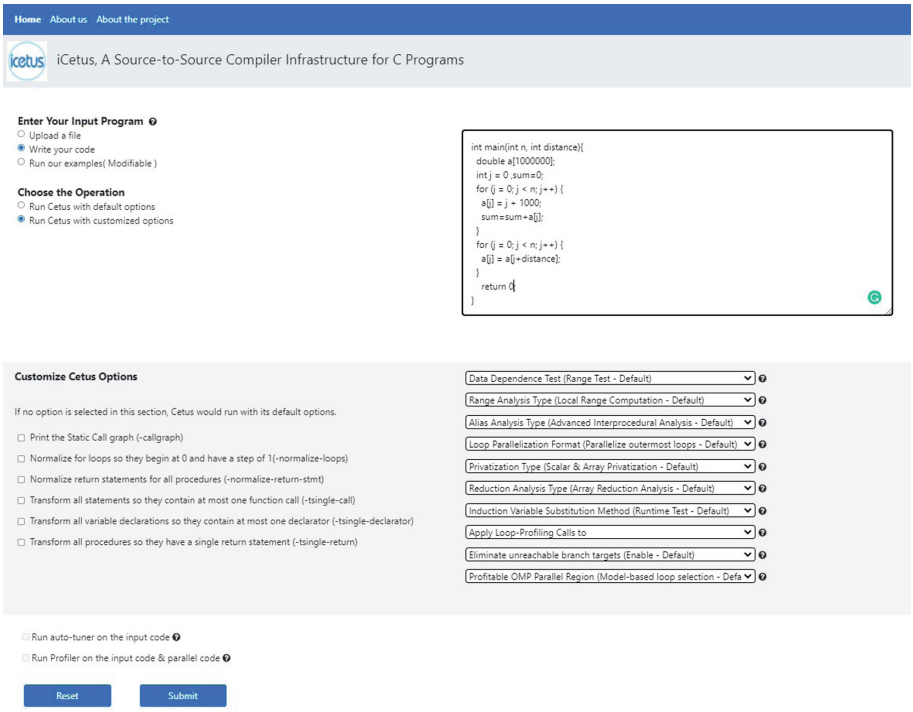

Fig. 1. Front view of iCetus, available at http://icetus.ece.udel.edu/cetusWeb/. The availability as a web tool obviates the need for download, install and version updates.

Building on the Cetus source-to-source restructurer, the tool displays the parallelized version of a given program in the form of OpenMP-annotated source code. The tool allows the user to observe the applied transformations and can serve as a starting point for further, manual optimizations.

iCetus is developed with the purpose of extending the capabilities of the Cetus compiler. Our intention is not to present just a user-friendly interface to the Cetus compiler, but to convert an automatic compiler to an interactive one. The followings are key features of the current iCetus prototype:

- iCetus is developed as a web application, in order to make it easier for the user to interact with it. Such an implementation introduces lots of benefits like cross-platform availability, portability, no need for installation, automatic updates, and being light on client-side computer resources since all processing would be done on server-side resources.
- Making the parallelization process easily customizable in a menu-driven and interactive way.

– Making the optimization process less error-prone by guiding the programmer's attention to the regions that hinder parallelization.
– Providing an interactive menu-driven display of program analyses and transformations while enabling the user to act on that information and make required modifications to the input code.
– Providing Run-time measurements gained through program execution, such as profiling information as well as the speedup and the efficiency of the code.

Figure 1, on page 6 shows the front view of iCetus. The user has typed a sample input program (alternatively a file can be uploaded or selected from among examples that illustrate key concepts) and has chosen to customize a number of compilation options.

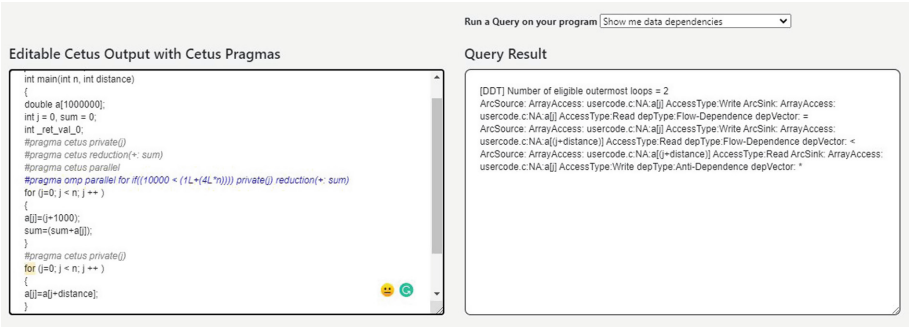

Fig. 2. Parallel code & display of data dependency information (Color figure online)

Figure 2 illustrates the menu-driven display of program analysis results. In the given program that is displayed on the left side, the second loop is not parallelized and is marked yellow. Color coding is applied to the output for showing the loop that is not parallelized to the user. That's why the second "for" loop is highlighted yellow. Given that information, the user has chosen to look at the existing data dependences from the drop-down menu. This menu is designed to let the user easily query the result of different analyses performed by the compiler on the given program. This feature not only helps the user identify the impediments of parallelization but also displays the performance gain from applying parallelization. Based on the query passed by the user, the report on the right side of the screen updates. In this case, a *flow-dependency* between a[j] and a[j+distance] with dependence Vector of "<" in the second loop is displayed. In this example, the compiler does not know about the value of variables "n" and "distance", and it reports on the dependency that might exist in between a[j] and a[j+distance], in which the "j" variable increases from "0" to "n" in steps of 1.

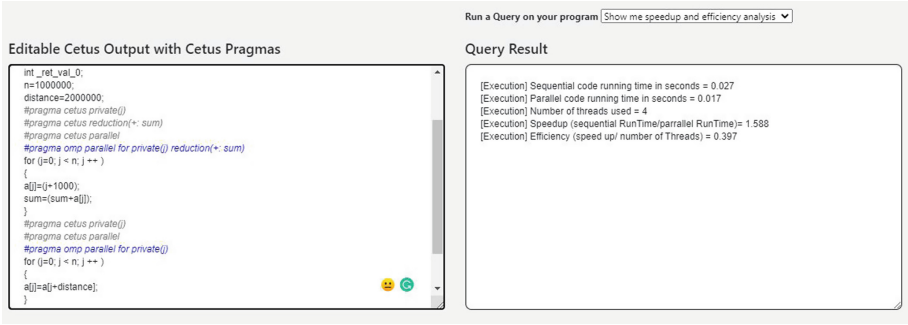

Fig. 3. Determining performance and efficiency

By providing a greater value to variable *"distance" comparing to variable* "n" the user manages to resolve the loop-carried flow dependency. Figure 3 shows the speedup and efficiency gained by the transformations after parallelizing both loops. The resources for this program execution are part of the web server, executing in a sandbox environment for security reasons.

The tool allows the user to edit and re-compile the resulting code. In this case, the data dependence is removed, turning the second loop into a parallel region as well. Recomputing the speedup shows the effect of this program improvement immediately.

2.4 Limitations of the Current Version of iCetus

Recall that iCetus is still in the early stages of its development. Some of the current limitations are given below; they will be resolved in future versions.

- The current version only accepts a source code from the user. It does not accept any data input file.
- The given program should be self-contained, meaning it must include all header files that contain developer definitions. The header files that come with the compiler are recognized by the tool, however.
- Computational resources for program executions to obtain profile runs and other dynamic measurements are limited to a small machine.
- The focus of the current version is on exploring the functionality needed by an interactive compiler. Adding the many "bells and whistles" needed for an easy-to-learn tool will come later.

3 iCetus System Overview

The iCetus tool is implemented as a dynamic web application, generating the pages/data in real time, as per the user's request. The response will trigger from the server end and reach the client, causing the desired action. Figure 4, on page 9, illustrates this process.

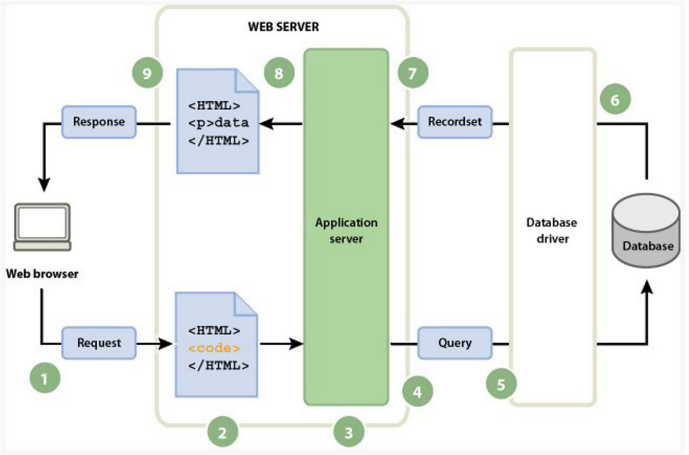

Fig. 4. Processing dynamic web pages

1. Web browser requests dynamic page.
2. Web server finds page and passes it to application server.
3. Application server scans page for instructions.
4. Application server sends query to database driver.
5. Driver executes the query against the database.
6. Record set is returned to driver.
7. Driver passes record set to application server.
8. Application server inserts data in page, and then passes the page to the web server.
9. Web server sends finished page to requesting browser.

As illustrated in Fig. 4, the current design includes a database that saves user inquiries. This information will be used for the purpose of evaluating the project.

Since the application server cannot communicate directly with the database, due to its proprietary format, an intermediary driver acts as an interpreter between the application server and the database. After the driver establishes communication, the query is executed against the database, creating a record set – a set of data extracted from one or more tables. This record set is returned to the application server to complete the page. The final result is in pure HTML

format, which the application server passes back to the web server. The page is then sent to the requesting browser.

Technologies and programming languages used in developing these web pages are: JSP 2.2, Apache Tomcat version 9.0.41, JSTL 1.2, Servlet API 3, Mysql connector 8.0, OpenMp 3, Java 11.0.2, GCC 9.2.0, JavaScript 1.0, HTML 5.0, CSS 2.0.

For software design, we have used an MVC (Model-View-Controller) design pattern to separate application concerns. In this method, *Model* represents objects carrying data, *View* represents the visualization of the data, and the *Controller* acts on both model and view by controlling the data flow into model objects and updating the view whenever data changes.

4 Evaluation

To evaluate the preliminary results of the project we presented the tool to more than 20 users with different skill levels with regard to parallelization techniques and familiarity with the OpenMP parallel programming language. Our goal is to make a tool that can serve users with different skill levels that's why the feedback of all participants matters to us.

The respondents to the survey include users with diverse skill levels. 38.1% of participants are categorized as for beginners with regard to knowing parallelization techniques. 47.6% of them are categorized as intermediate having some knowledge with regard to parallelization techniques, and 14.3% of the participants are categorized as advanced being able to parallelize the code manually.

Of our participants, 66.7% of them were not familiar with OpenMP parallel programming model, while 33.3% had a good understanding of it.

We also inquired our participants about their level of familiarity with the Cetus compiler. 61.9% of users did not know the Cetus compiler but 38.1% of the participants have already tried it at least once.

We presented the list of current iCetus features and also features that we consider implementing in the next version of the tool. We asked the users to rate these features on a scale from 1 to 5, where 1 means the feature is unimportant and 5 means the feature is judged very important. We also asked for a list of features the users wish to see in such an interactive tool.

Section 4.1 shows the resulting importance of current features of the iCetus tool, Sect. 4.2 evaluates the importance of features proposed by us to be considered for the next version of the tool, and Sect. 4.3 describes the features requested by users for the next release of the project.

4.1 Importance and Usefulness of Existing iCetus Features

Figure 5, on page 11, shows the results collected on the existing features of iCetus. The user scores for all questions are above 4, indicating importance and usefulness of all implemented features.

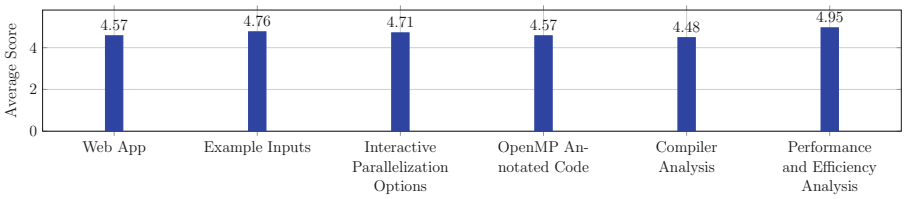

Fig. 5. User feedback on existing features

- **Web Application:** This question asked about the usefulness of iCetus being available as a web application. Having the tool implemented as a web application eliminates the need for download, install, and updates, and would be light weight on the client-side considering the fact that all the processing is done on the server-side. The high score of 4.57 indicates strong agreement with these advantages.
- **Example Inputs:** iCetus offers many example input programs that the user can choose from, illustrating key concepts of parallel programming, and transformations, as well as the tool functionalities. Users gave this feature the high score of 4.76.
- **Interactive Parallelization Options:** Users can choose parallelization options in a menu-driven way. This feature enables skilled users to take detailed control of the applied analyses and transformation techniques, while providing reasonable defaults for beginners. This question obtained a 4.71 score.
- **OpenMP Annotated Code:** Building on the Cetus source-to-source restructurer, iCetus shows the result of its transformations in the form of OpenMP-annotated source code. Users scored this feature 4.57. They also offered the following comments to explain the relevance of this capability: OpenMP-annotated source code makes it easy to understand the transformations applied to a code. The portability of OpenMP provides for a good abstraction of possible underlying machines, eliminating the need for understanding many architectural details. Similarly, reasonable performance portability is appreciated. Last but not least, the users valued the incremental parallelization process supported by this feature.
- **Compiler Analysis:** This key feature enables users to understand the applied compiler passes and inspect specific categories of program analysis results. In this way, users can query the compiler's reasoning, drilling down into questions why certain program optimizations could or could not be applied, and determining possible manual program changes to increase performance. The score for this feature was also 4.48.
- **Performance & Efficiency Analysis:** With the highest score of 4.95, users judged the availability of run-time information, such as performance and efficiency as most important. This result is consistent with the fact that the lack of run-time information can be viewed as the Achilles heel of static, batch-oriented automatic parallelization. It also points to an opportunity for

improving parallelization environments further by including additional types of dynamic program information.

4.2 Importance and Usefulness of our Proposed iCetus Features

We asked for user feedback on the features we proposed to be added to the next version of the tool. Figure 6 reports the obtained scores.

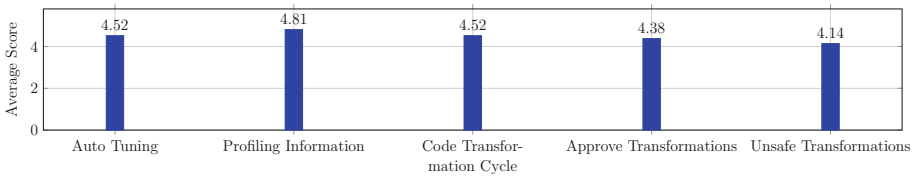

Fig. 6. User feedback on our proposed features

- **Auto-tuning:** Having an *auto-tuning* capability that determines the best combination of compiler options, obtained a score of 4.52. Some users wanted the tool to find the combination that leads to the best performance, but wanted some control over the techniques being tuned. Having such control is important, as auto-tuning can be a highly time-consuming process. Another reason given was that auto-tuning can help users learn and understand code parallelization, how it applies in different use cases, and what performance can be expected.
- **Profiling Information:** Providing loop-by-loop profiling information in the serial code and parallel code, as well as loop speedups and efficiencies, are important aids in the optimization process, indicated by the score of 4.81. The feature helps users focus attention on relevant code sections and understand performance bottlenecks.
- **Code Transformation Cycle:** Being able to modify the input code and submit it for another round of compilation is essential in an interactive optimization scenario. Applying such modifications in the presence of the available analyses information goes substantially beyond the features offered by a standard program editor. The user score for this feature was 4.52.
- **Approve Transformations:** Giving the user the ability to approve or reject transformations suggested by the parallelizer provides fine control over the code optimization process, especially for judging the profitability of a transformation. The score for this feature was 4.38.
- **Unsafe Transformations:** With a score of 4.14 users judged the importance of a capability to choose from potentially applicable transformations, even if they may be unsafe. Some users requested that this option be only available to advanced skill levels, as program correctness is no longer guaranteed.

While all scores of proposed features are above 4, they are slightly lower than those of the implemented capabilities. It can be attributed to the fact that it is easier to understand and judge existing versus projected functionality. The scores are expected to be higher, once the proposed features are implemented.

4.3 Requested Features for iCetus

One of the questions in the user interviews asked for additional suggested features. Below is the result, including the percentage of users who requested those features. The priority of implementing each feature will be based on the score. Table 1 lists these suggestions.

Table 1. Requested features by users

Row	Requested features	Priority
1	Graphical representations	33%
2	Downloading optimization reports	28%
3	Uploading multiple files	19%
4	Display differences between the input and the parallelized code	19%

- **Graphical Representations:** 33% of users requested combining text reports on the result of compiler analyses with graphical reports wherever possible.
- **Downloading Optimization Reports:** Providing the possibility of downloading the parallel code as well as the report of the compiler analyses was requested by 28% of the users.
- **Uploading Multiple Files:** 19% of users requested adding the feature to upload as many files as needed to the web server at once.
- **Display Differences Between the Input & the Parallelized Code:** 19% of users requested that the differences between the given input and the parallelized code be displayed. Such a capability would help the developer further understand the specifics of the applied code transformations.

5 Related Work

Various tools have been built in the past which aim to parallelize the sequential code. ParTool [5], which is built over the ROSE compiler infrastructure [7], inserts OpenMP pragmas in serial code. It performs data dependence analysis provided by ROSE to ascertain whether a loop nest is safe to parallelize. If not, the dependences that prevent parallelization are displayed. This feedback helps understand the dependences hindering parallelism and can be used to make suitable modifications to the source code to eliminate these dependences.

The Parascope parallelization environment [1] provides an editor that supports multiple views and navigation between views. It displays the results of the various analyses and transformations carried out by the parallelizer and binds them with the various representations used. It supports applications written in Fortran. Users have found the data dependence information to be too low-level, and they need guidance with program transforms.

HTGviz is an interactive parallelization environment. It is implemented on top of the Parafrase-2 parallelizing compiler [6]. It supports several views to the user such as, Task Graph View, Serial Code View, Directive View to insert OpenMP tags, Parallel Code View. The interaction between the user and the compiler is carried out through the use of the Hierarchical Task Graph (HTG) program representation where task parallelism is represented by precedence relations (arcs) among task nodes. There is no support for measuring the parallelization benefits, or for displaying potential parallelism, at a regional level [3].

The SUIF Explorer [4] builds on the functionality of the SUIF compiler [8] and offers assistance for both automated and manual parallel programs creation. The SUIF Explorer offers support for user visualization and provides features such as a Parallelization Guru that offers tips for parallelization, user involvement in parallel slice creation, Execution Analyzers targeting loops and dependences, Visualizers such as graph browsers and source display, and Assertion Checkers to help users debug the parallel program.

iCetus distinguishes itself from these previous efforts mainly in three ways.

- Building on one of the most advanced parallelizers, the tool allows the user to inspect in detail the result of different compiler analyses, such as data dependence analysis, variable range analysis, private variable analysis, in an easy to understand format.
- The tool provides the user with dynamic program information, such as the speedup gained from a transformation, enabling the user to judge when further optimizations may be beneficial or have diminishing return.
- The tool supports the user in all phases of the program optimization process, including profiling, parallelizing, and optimizing.

6 Conclusion

State-of-the-art parallelizing compilers are batch-oriented tools, limited to static program analyses and transformation. This paper presented the early results of a project to develop a tool that overcomes this limitation. iCetus is an effort to involve the user in the code transformation process, supporting several program development phases. A profiler helps the programmer *analyze* the code by identifying execution bottlenecks of the program. The programmer then *parallelizes* the code by starting with the most time consuming code sections while focusing on maintaining the correct results of the parallel program. *Optimizing* the code for improving observed speed-up from parallelization is the final phase. The next release of the tool will incorporate more features in support of interactivity as well as features such as a loop-level profiler, auto-tuner, and a capability to highlight differences between source and transformed code.

References

1. Balasundaram, V., Kennedy, K., Kremer, U., McKinley, K., Subhlok, J.: The parascope editor: an interactive parallel programming tool. In: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing, Supercomputing 1989, pp. 540–550 (1989). https://doi.org/10.1145/76263.76323
2. Dave, C., Bae, H., Min, S.J., Lee, S., Eigenmann, R., Midkiff, S.: Cetus: a source-to-source compiler infrastructure for multicores. Computer **42**(12), 36–42 (2009). https://doi.org/10.1109/MC.2009.385
3. Giordano, M., Furnari, M.M.: HTGviz: a graphic tool for the synthesis of automatic and user-driven program parallelization in the compilation process. In: Polychronopoulos, C., Fukuda, K.J.A., Tomita, S. (eds.) ISHPC 1999. LNCS, vol. 1615, pp. 312–319. Springer, Heidelberg (1999). https://doi.org/10.1007/BFb0094932
4. Liao, S.W., Diwan, A., Bosch, R.P., Ghuloum, A., Lam, M.S.: SUIF Explorer: an interactive and interprocedural parallelizer. ACM SIGPLAN Not. **34**(8), 37–48 (1999). https://doi.org/10.1145/329366.301108
5. Mishra, V., Aggarwal, S.K.: ParTool: a feedback-directed parallelizer. In: Temam, O., Yew, P.-C., Zang, B. (eds.) APPT 2011. LNCS, vol. 6965, pp. 157–171. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24151-2_12
6. Polychronopoulos, C.D., Girkar, M.B., Haghighat, M.R., Lee, C.L., Leung, B., Schouten, D.: PARAFRASE-2: an environment for parallelizing, partitioning, synchronizing, and scheduling programs on multiprocessors. Int. J. High Speed Comput. **01**(01), 45–72 (1989). https://doi.org/10.1142/S0129053389000044
7. Quinlan, D., Liao, C.: The ROSE source-to-source compiler infrastructure. In: Cetus Users and Compiler Infrastructure Workshop, in conjunction with PACT 2011, p. 1. Citeseer (2011)
8. Wilson, R.P., et al.: The SUIF compiler system: a parallelizing and optimizing research compiler. ACM SIGPLAN Not. (1994)