IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received April 5, 2020, accepted May 19, 2020, date of publication May 22, 2020, date of current version June 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2996669

Extending OpenMP for the Optimization of Parallel Component Applications

YUNFENG PENG AND HAI LIU
School of Software Engineering, Anyang Normal University, Anyang 455000, China

Corresponding author: Yunfeng Peng (peng_ayit@163.com)

This work was supported by the Key Research and Development and Promotion Projects in Henan Province, China, under Grant 192102310212.

ABSTRACT Motivated by the promotion of development efficiency and performance, we research on the common performance optimization technology for parallel component applications. We define OpenMP pragmas to describe the parallel component instance calls in three types of complex structure code. By data flow analysis, we can find the parallelism potential of these codes. We extends the OpenMP pragma to support virtual computing resources and the deployment of complex structure codes on them. We provide three reference rules for the component instance tasks to be scheduled on certain resources to get better performance. To better utilize resources, we provide mapping of virtual computing resources to real resources. We extends the OpenMP programming model and execution model to accommodate the extended pragmas. The extended Babel compiler is responsible for data flow analysis. The extended CCAFFEINE framework is in charge of generating the task schedule policy. Experiments show that our performance optimization platform Bomp can get better scalability and performance than other methods existed.

INDEX TERMS Parallel component, performance optimization, OpenMP, parallel software engineering.

I. INTRODUCTION

The traditional method of parallel software development is code written by scientists in specific fields, which greatly affects the development efficiency of parallel software. At the same time, more and more new hardware architectures are emerging, which makes developers focus on improving the portability and the development efficiency of parallel software, so as to keep up with the changes of architecture. Scientists proposed to use software engineering to solve the problem of parallel software development. However, the current mainstream component specifications are not oriented to the field of high-performance parallel computing. Therefore, with the establishment of the CCA(common component architecture) [1] forum in 1998 as a sign, researchers began to study the parallel component technology applicable to the field of scientific computing based on the traditional serial component technology. Performance prediction or adaptive methods are commonly used to improve the performance of parallel component programs [2]. After using the common strategy of performance optimization of parallel component program, most of the execution time of parallel component program is spent on some complex structure codes which can not be optimized by common means of performance optimization. These codes become the bottleneck to further improve the performance of parallel component programs. Based on the OpenMP [3] parallel computing model, this paper extends the model. We use communication optimization technology to support the optimal scheduling of parallel component instance calls in complex structure codes. Our research supports the description and mapping of heterogeneous hardware platforms, and improves the performance of parallel component programs, which has great practical significance. The parallel component performance optimization platform developed in this paper is called Bomp. The system consists of three parts: the extended Babel compiler [4], the extended ccaffeine [5] framework and the runtime library. This system has been applied to the HPC(High Performance Computing) heterogeneous cluster in the high performance and Data Engineering Laboratory of University of science and technology Beijing(USTB), greatly improving the performance of parallel component programs running on the cluster.

II. EXTENDING OPENMP TO SUPPORT CCA PARALLEL COMPONENTS

A. OPERATION MECHANISM OF CCA PARALLEL COMPONENT

There are two operation modes of CCA parallel components, SCMD(single component multiple data) and MCMD(multiple

The associate editor coordinating the review of this manuscript and approving it for publication was Stavros Souravlas.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

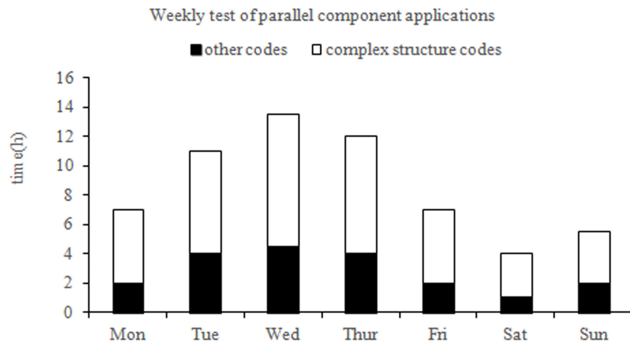

FIGURE 1. Weekly test of parallel component applications executed on the heterogeneous cluster of the high-performance and data engineering laboratory of USTB.

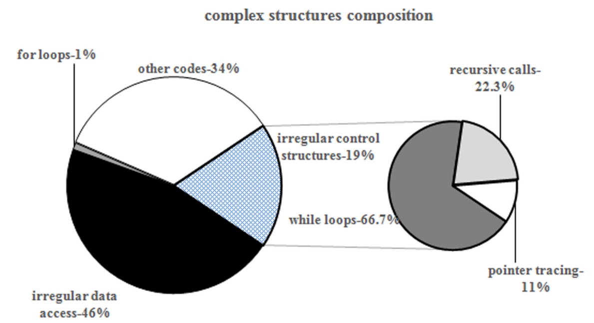

FIGURE 2. Composition of complex structures, These complex structure codes mainly include three types, namely the for loops; irregular control structures; calls to parallel component instances that access irregular data structures. Among them, the irregular control structures mainly includes three kinds of irregular control flow, namely the unbounded while loop, recursive call and pointer tracing.

components multiple data) [5]. When a parallel component instance is generated, the first step is to create a C++ object to represent the component instance. Then the framework engine uses the Component Register function to register the component instance with the framework, which is equivalent to generating the service object representing the component instance at the framework end. The component registers its own provides and uses ports with the framework through the addProvidesPort() and registerUsesPort() methods in the object. RMI (remote message invocation) [6] mechanism is used for component calls in different processes. At present, ccaffeine [5] framework has added support for calling MPI communication library, which can be directly used in program code. Adding pthread multithreading code to component implementation code can realize multithreading of component internal code. We can also add OpenMP pragmas to component implementation code to support multithreading optimization by linking the runtime library of OpenMP at compiling time. The existing means to optimize the running of CCAFFEINE components are mainly to dynamically change the number of processes running MPI code in the component, or the number of threads running OpenMP code, or to place the components on the appropriate resources through performance predictions [7].

B. COMPONENT INSTANCE INVOCATION WITHIN COMPLEX STRUCTURE CODE

In reality a call to a parallel component instance is often contained in the implementation code of another parallel component instance. The existing optimization methods improve the adaptability of parallel components by identifying the parallel execution pattern of the internal codes. This identification destroys the black-box nature of parallel components and increases the burden on programmers. The development of parallel component technology will be greatly promoted by mining the parallelism of parallel component program independent of the specific architecture and improving the performance of parallel component program. Figure 1 shows the performance record of parallel component programs executed on the heterogeneous cluster of the high-performance and data engineering laboratory of USTB in a week. These programs include applications such as reservoir numerical simulation and material molecular dynamics simulation. It can be seen from this record that in one week, the parallel component programs spent about 80% of their time on the execution of complex structure codes. How to improve the performance of this part of codes becomes the bottleneck for the parallel component programs.

Through data analysis of the test in Figure 1, the composition of the complex structure codes that affects the performance of the parallel component programs is shown in Figure 2. These complex structure codes mainly include three types, namely the for loops; irregular control structures; calls to parallel component instances that access irregular data structures. Among them, the irregular control structures mainly includes three kinds of irregular control flow, namely the unbounded while loop, recursive call and pointer tracing. Given a for loop, the inner part contains the invocations of the instances of parallel component A. Through data flow analysis, it can be concluded that there are n-m loop iterations of the loop, and each iteration has h times to invoke the instances of A. If the h∗ (n-m) calls do not have any dependencies including data dependency and control dependency, the input parameters of each component instance can be obtained through data flow analysis, and the input parameters of each instance are roughly the same size. The maximum parallelism of the for loop is h∗(n-m). The smallest unit of parallel execution is an A component instance. We can then record the results of the data flow analysis can be recorded in an XML file. For irregular control structures and parallel component instance invocations accessing irregular data structures, the results of data flow analysis can be recorded in an XML file in a similar manner.

The OpenMP extension pragmas are defined as follows:

#pragma bomp component A (in si, inout li) (a)

#pragma bomp component R h^* for (n-m)
A {(s1, l1), (s2, l2)…(sn-m, ln-m)},
…{(h1, j1), (h2, j2)…(ht, jt)} (b)

#pragma bomp component R h *while(unkown)
A {(s1, l1), (s2, l2),…(st, lt) }, …{(h1, j1),…(ht, jt)} (c)

#pragma bomp component R h^* recursive (unkown)
A {(s1, l1), ...(st, lt) }, ... {(h1, j1), ...(ht, jt)} (d)
#pragma bomp component R h^*recursive (unkown)
A {(s1, l1), ...(st, lt) }, ... {(h1, j1), ...(ht, jt)} pre (e)
#pragma bomp component R h^*recursive (unkown)
A {(s1, l1), ...(st, lt) }, ... {(h1, j1), ...(ht, jt)} suf (f)

Pragma (a) is used to identify a parallel component instance call. Pragma (b) is used to identify a for loop containing parallel component instance calls. Pragma (c) is used to identify an unbounded while loop containing parallel component instance calls. Pragma (d)-(f) is used to identify the recursive component that contains its own component instance call. Pragma(d) represents the recursive component instance call without dependency. Pre in pragma (e) represents the pre-dependency between the recursive component instance calls. Here pre-dependency means the execution of an instance call must wait for the execution of its generated instance call to complete. Suf in pragma (f) represents the suf-dependency between the recursive component instance calls. Here suf-dependency means the generation of the task of the invoked component instance, that is, the acquisition of the input parameters of the invoked component instance, must depend on the completion of the calculation of the invoking component instance. In addition, there are corresponding extended OpenMP pragmas for pointer tracing and parallel component instance calls accessing irregular data structures. The format and parameters are similar to (b) - (f).

III. SUPPORT TO HETEROGENEOUS PLATFORM AND SCHEDULING

Because parallel software often has high requirements for performance and execution platform, regardless of the difference of running platform will bring performance loss, and even affect the correctness of software in some cases.

This paper extends OpenMP to describe heterogeneous resource platform. Our extended pragmas support the description of virtual computing resources. The extended mechanisms support the mapping of virtual resources to real resources, and performance optimization of parallel components on heterogeneous platforms through appropriate component deployment strategies.

A. DEPLOYMENT ON VIRTUAL RESOURCES

To better describe the platform on which the component runs and facilitate the scheduled execution of the component, this paper extends the OpenMP pragma to support virtual computing resources. Each virtual computing resource identified by the OpenMP extension pragma has a unique name. Pragma (g) identifies a virtual isomorphic multicore server.

pragma bomp server T procs(m), cores(n) (g)

For the parallel component instance call in the complex structure codes proposed in this paper, we provides extended OpenMP pragmas to support the scheduling and specific scheduling strategy of the parallel component to virtual computing resource by referring to the two task allocation methods of master-worker and task pool.

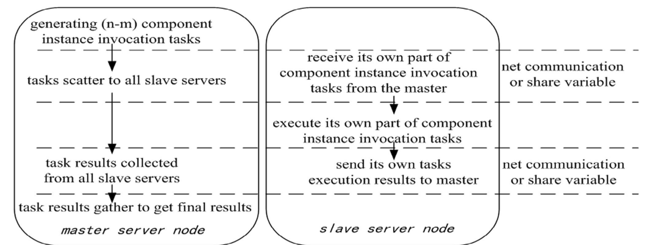

FIGURE 3. Task distribution in for loop. A component instance is invoked as a task. Master server node scatters the tasks to slaves, and it gather results from slaves.

#pragma bomp schedule R W on T (h)

Pragma (h) indicates that the structures R and W containing parallel component invocations are scheduled to run on the virtual computing resource T.

The specific scheduling policy mainly includes the following three reference rules:

1) If T is a virtual isomorphic multicore server and the number of available computing cores is r, different scheduling policies can be generated for different code structures. If R is a structure defined by pragmas (b), (c), or (d), then the value of the corresponding parameter can be obtained at runtime. A component instance is invoked as a task, and the overall task allocation is shown in figure 3.

2) If R is a recursive invocation of the component defined in pragma (e) and (f), then a dependency table needs to be maintained in addition to a parameter list after all the tasks are generated. If T is a virtual SMP cluster composed of r virtual SMP servers, the communication between the computing cores on different SMP is conducted through the high-performance network communication protocol armci [8] to input data of the task and return results. We use SMP server multi-threading mechanism, overlapping component instance call execution, communication, disk I/O and other operations, to improve the performance of the program execution.

3) If execution code of the invoked parallel component instance is implemented by SPU [9] vector operations on CELL BE processors and T is a virtual PS3 server, the complex structure codes containing the parallel component instance invocation can also be optimized for execution on T. The optimizing schedule process is similar to that of isomorphic multicore servers.

B. MAPPING OF VIRTUAL COMPUTING RESOURCES TO REAL RESOURCES

In order to better utilize resources and properly deploy parallel component, this paper defines two types of mapping from virtual computing resources to real computing resources. Congeneric mapping is mapping a virtual server to an actual server that has the same configuration as it. This is the preferred mapping when resources are sufficient. Uncongeneric mapping includes mapping virtual multicore

servers to real SMP clusters and mapping virtual SMP clusters to real multicore servers or server clusters. For the virtual multicore server mapping to the actual SMP cluster, there is also a source-to-source translation of the C + extended OpenMP pragmas to the C + armci high-performance network communication library. The translation process is divided into two parts. For the calculation codes which are not component invocations, the method is similar to the ordinary OpenMP to MPI source-to-source translation [10]. First, through the translation from OpenMP to MPI, the MPI version of the program is obtained, and then in the component program armci initialization and end statements are added. MPI statements are replaced by the corresponding armci functions. Armci provides high performance network communication by packaging and sending decentralized, fine-grained communication, reducing the number of interrupts caused by messages, and prefetching data. Armci and MPI can be used together.

IV. EXTENDING THE OpenMP COMPILER AND RUNTIME

A. EXTENDING THE OpenMP PROGRAMMING MODEL

The traditional OpenMP programming model mainly identifies the parallelizable parts(for loops) with the pragmas and generates the corresponding multithreaded parallel program through the compiler. In this paper, the programming model is extended. When compiling parallel components, the extended Babel compiler conducts data flow analysis on the source code of parallel components, and automatically adds extended OpenMP pragmas to identify the parallel component instances within the complex structure codes. The extended Babel compiler records the results of data flow analysis in XML files. The components are compiled into.la library files suitable for CCAFFEINE framework. Before the component application runs, the user defines the component to be used and the connection mode in the rc file [5]. For simple component, an instance of the specified component can be generated with the instantiate command in the rc file. However, for component invocations within complex structure codes, users should define the virtual resources to be used and the deployment of structure codes to resources by extended OpenMP pragmas. When the CCAFFEINE framework reads a rc file, it parses the contents of the file using the parser function. We added parsing of extended OpenMP pragmas to the parser. These pragmas describe virtual resources and the deployment of components. By searching the XML file of the specific component, the parser obtains the codes of the complex structures contained in that component, and the invocation forms of the component instances within the structures. In combination with the information of the resources, a policy generator generates the specific component deployment strategy. The component initiator then deploys the component according to the deployment strategy and starts the program to run.

B. EXTENDING THE OpenMP EXECUTION MODEL

The traditional OpenMP execution model is carried out in the fork-join form. In this paper, the model is extended to determine the communication mode between different component invocation tasks before program execution. According to the different deployment platforms, it can be divided into two types: Shared memory and armci network communication.

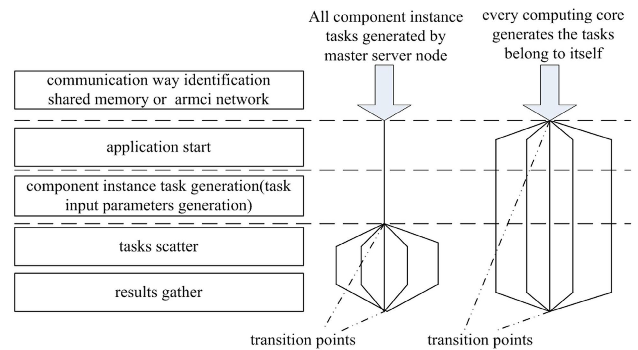

FIGURE 4. Extended OpenMP execution model. The execution mode transition point of the parallel component program is defined as the turning point of serial execution to parallel or from parallel execution to serial. For different modes of communication, the operations performed at the transition point are different.

As shown in Figure 4, the execution mode transition point of the parallel component program is defined as the turning point of serial execution to parallel or from parallel execution to serial. For different modes of communication, the operations performed at the transition point are different. For shared-memory communication, a fork or join operation is performed at the transition point, changing the number of threads in the program. For armci network communication, MPI_Comm _Spawn and MPI_Comm_Free operations are performed at the transition point to change the number of MPI processes in the program. The position of transition point also varies with the generation method of component instance task in parallel component program.

C. EXTENDING THE OpenMP COMPILER AND RUNTIME LIBRARY

In this paper, the extended OpenMP programming model and execution model are implemented through the parallel parallel component performance optimization platform Bomp. After writing the source code of the parallel component, the parallel component writer first compiles the source code of the parallel component with the extended Babel compiler. The extended Babel compiler is responsible for data flow analysis of the source code of the parallel component and automatically adds the extended OpenMP pragmas defined in this article to identify the calls of the parallel component instance within the complex structure codes defined in this article. In addition, the results of the data flow analysis are recorded in an XML file. The extended Babel compiler generates the.la dynamic linked library of the components. In this way, the parallel component writer provides the user with.la libraries representing the parallel components and XML files that containing the results of data flow analysis. When a user wants to start an application, the first step is to write a rc file to define the connections between the parallel components

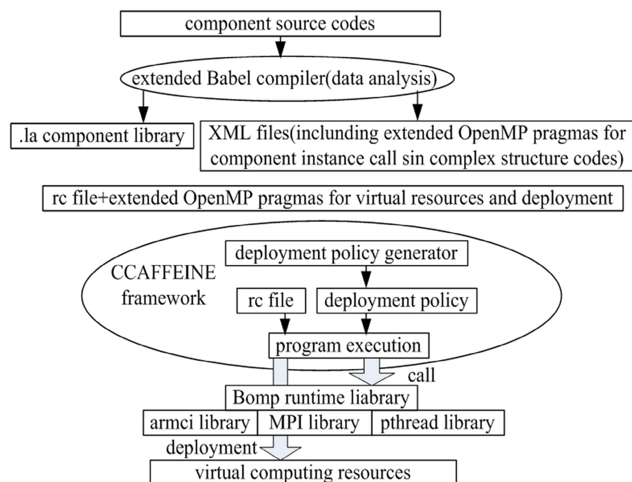

FIGURE 5. **Bomp platform for parallel component. The extended Babel compiler is responsible for data flow analysis of the source code of the parallel component. The extended CCAFFEINE Framework analyzes the extended rc file. It can get the information of virtual resources and the deployment of component instances to virtual resources from the extended OpenMP pragmas.**

in the whole parallel component program. At this point, users can add the extended OpenMP pragmas defined in this article to the rc file to identify the platform's virtual computing resources and the deployment of component instances to virtual resources. The user then runs the entire parallel component program by running the extended CCAFFEINE framework with a rc file.

The extended CCAFFEINE framework analyzes the extended rc file. It can get the information of virtual resources and the deployment of component instances to virtual resources from the extended OpenMP pragmas. The CCAFFEINE framework, based on the results of data flow analysis in the XML file and the definition of the component in the.la component library, generates the optimal scheduling policy of the component instances on the virtual resources, and then executes the policy to run the entire parallel component program. The Bomp runtime library defines a set of functions for process and thread management, component instance task generation, partitioning, scheduling, and communication. Depending on the virtual deployment resources, these functions are translated into calls to the armci library, the MPI runtime library [11], and the pthread multithreaded library [12]. Figure 5 illustrates the operation of the parallel component performance optimization platform.

In order to make better use of the resources of heterogeneous cluster platform and reasonably conduct the deployment and scheduling of parallel components, this paper implements a resource manager to uniformly manage the software and hardware resources of heterogeneous platform. Our resource manager takes a similar approach to Concerto [13] in modeling resources by asking the local agent for the availability of individual servers.

V. EXPERIMENTS AND RESULTS

With the improvement of model accuracy and the increase of computation load, parallel computing becomes the

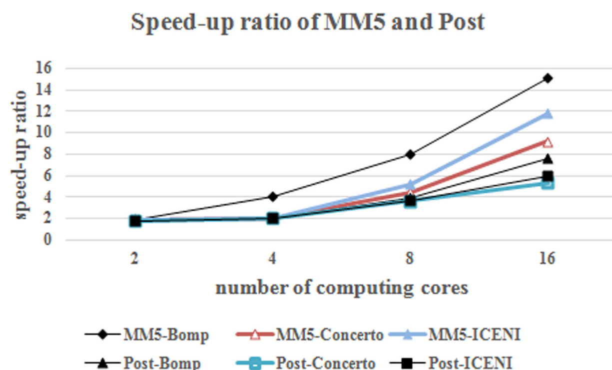

FIGURE 6. **Speed-up ratio of MM5 and Post. The optimal speed-up ratio of Bomp system on MM5 components is almost linear. The acceleration ratio on MM5 is better than on many other parts as it varies with parallel threads. Post component gets a lower acceleration ratio.**

first choice for numerical meteorological prediction. The implementation of numerical prediction software by means of component is beneficial to the software reuse. MM5(fifth generation mesoscale model) [14] is a mesoscale weather prediction model jointly developed by the University of Pennsylvania and National Center for Atmospheric Research(NCAR). In this paper, the typical functions in this pattern are extracted, and a component-based test case is made by using C language and component tools of Bomp performance optimization platform. The test mainly includes two components, MM5 and Post. For comparison, a MM5 testing system with Concerto's adaptive function and a MM5 system with ICENI's performance prediction and adaptive function [15] are also implemented manually. The experiment was deployed on a heterogeneous platform consisting of three clusters, including a multicore server cluster, an SMP cluster, and a PS3 cluster. High-speed Ethernet connections are used between the servers. The input data is the height, temperature and other weather information of Pennsylvania every 2 hours in two days. The output is the precipitation in the next 48 hours. Meanwhile, in the performance test section, this paper tests the performance of an original MM5 system without any optimization added as a comparison.

A. SCALABILITY TEST

Figure 6 shows a comparison of the speed-up ratios of different application versions deployed on a 16-core multicore server. The Bomp version has a resource detection mechanism similar to Concerto, but avoids the performance prediction overhead of going through the performance prediction phase of ICENI. In addition, the optimization of complex structure codes is added. As the core component of the program, MM5 component contains the most complex structure codes with parallel instance invocation, and it invokes the instances of these optimized components more frequently than other parts of the application. As a result, the optimal speed-up ratio of Bomp system on MM5 components is almost linear. For Concerto version, the calculation amount of MM5 part is the largest, and the calculation-communication

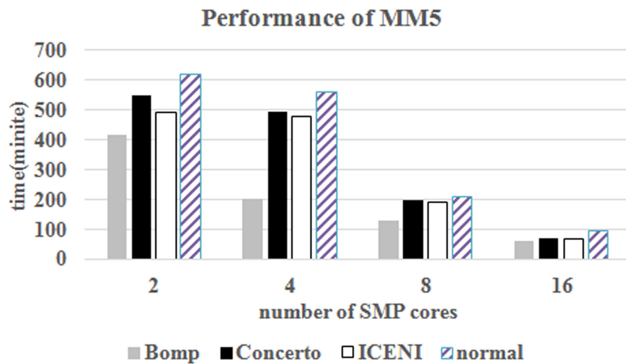

FIGURE 7. Performance of MM5. Performance of Bomp version of MM5 component is better than that of other versions. ICENI version is better than the Concerto version. Concerto version performs better than the original version.

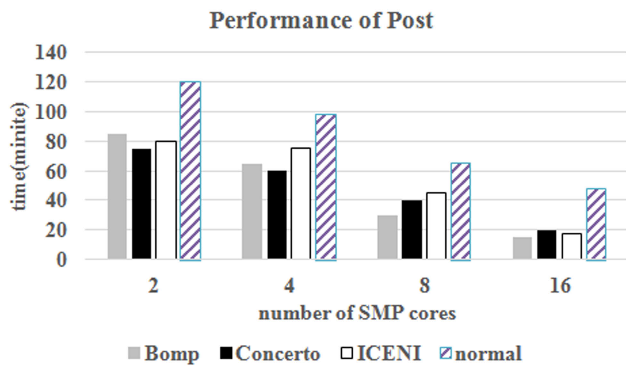

FIGURE 8. Performance of Post. Bomp version does not have a significant advantage over other versions. Bomp version does not perform as well as the Concerto version even at 2 and 4 nodes. As the number of nodes increases, the gap between versions narrows. At 16 nodes, the performance of the Bomp version is optimal.

ratio is also the largest. The acceleration ratio on MM5 is also better than on many other parts as it varies with parallel threads. In comparison, the ICENI version has a slightly lower speed-up ratio due to fewer implementation options for MM5 and little difference between implementations. Post component contains fewer optimizable structures and gets a lower acceleration ratio, but the Bomp version is still the best.

From the above analysis, it can be seen that since the Bomp performance optimization platform has adopted a lot of optimization for component instance invocations in for loops and while loops, and added the unique optimization for pointer tracing, recursive invocations and component instance invocations accessing irregular data structure, its scalability is better than the traditional methods of changing the parallelism of components, such as Concerto and ASSIST [16].

B. PERFORMANCE TEST

For performance test, we deployed four versions of the component on a SMP cluster, using 2, 4, 8, 16 SMP servers as the deployment nodes. The result is shown in Figure 7 and Figure 8. As can be seen from Figure 7, the performance of Bomp version of MM5 component is better than that of other versions due to more structures that can be optimized by Bomp. While MM5 has more implementations to choose from, after selecting the appropriate implementation through performance prediction, the ICENI version is better than the Concerto version, which relies solely on a fixed implementation. The Concerto version still performs better than the original because it dynamically changes the parallelism of components.

In Figure 8, since Post component contain fewer structures that can be optimized with Bomp, the Bomp version does not have a significant advantage over other versions. Due to the overhead of component instance task generation and result gather, the Bomp version does not perform as well as the Concerto version even at 2 and 4 nodes. As the number of nodes increases, the gap between versions narrows. At 16 nodes, the performance of the Bomp version is still optimal. It can be seen that the method presented in this paper can achieve better performance in the general high performance computing environment.

VI. CONCLUSION

By analyzing the execution performance of common parallel component programs, optimizational scheduling strategies are proposed in this paper. These strategies can be used for parallel component instances in three types of complex structures on heterogeneous platforms. We propose to extend the OpenMP pragmas to support the description of parallel component instance calls in these complex structured codes. Our extended pragmas can also describe computing resources of a heterogeneous platform and the deployment of parallel components on computing resources. We support the generation and execution of specific optimized scheduling policies by extending the Babel compiler and the CCAFFEINE running framework. Experiments show that the proposed parallel component program optimization method has better scalability and performance than the existing methods, and there is no additional burden on users. Our future work will include more performance experiments on different kinds of parallel applications. These experiments will give us more information about the optimization opportunities for the code structures in these applications.

REFERENCES

[1] CCA Forum. *The Common Component Architecture Forum*. Accessed: Apr. 3, 2020. [Online]. Available: http://www.cca-forum.org

[2] D. Bán, R. Ferenc, I. Siket, Á. Kiss, and T. Gyimóthy "Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware," *J. Supercomput.*, vol. 75, no. 1, pp. 4001–4025, Feb. 2018.

[3] OpenMP. *OpenMP Architecture Review Board*. Accessed: Apr. 3, 2020. [Online]. Available: http://openmp.org/

[4] Lawrence Livermore National Laboratory (LLNL). *Babel*. Accessed: Apr. 3, 2020. [Online]. Available: https://computing.llnl.gov/projects/babel-high-performance-language-interoperability/#page=home

[5] Y. Peng, L. Yao, C. Zhao, and C. Hu, "Overview of technologies for parallel component," *Comput. Sci.*, vol. 38, no. 2, pp. 18–27, Feb. 2011.

[6] Oracle. *Remote Method Invocation Home*. Accessed: Apr. 3, 2020. [Online]. Available: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html
[7] Y. Peng, "Design and implementation of medical information system based on parallel component technology," *Practical Electron.*, vol. 27, no. 22, pp. 43–44, Nov. 2018.
[8] The Pacific Northwest National Laboratory. *ARMCI-Aggregate Remote Memory Copy Interface*, Accessed: Apr. 3, 2020. [Online]. Available: http://www.emsl.pnl.gov/docs/parsoft/armci/
[9] Y. Chai, W. Shen, Z. Zhang, and Z. Tang, "Design and implementation of cell BE high performance computing experimental platform," *Res. Explor. Lab.*, vol. 30, no. 5, pp. 68–71, May 2011.
[10] Sourceforge. *OpenMP Package for ORC*, Accessed: Apr. 3, 2020. [Online]. Available: https://sourceforge.net/projects/orc-openmp/
[11] J. Klinkenberg, P. Samfass, M. Bader, C. Terboven, and M. S. Müller, "CHAMELEON: Reactive load balancing for hybrid MPI+ OpenMP task-parallel applications," *J. Parallel Distrib. Comput.*, vol. 138, pp. 55–64, Apr. 2020.
[12] Lawrence Livermore National Laboratory(LLNL). *POSIX Threads Programming*. Accessed: Apr. 3, 2020. [Online]. Available: https://computing.llnl.gov/tutorials/pthreads/
[13] P. Temple, M. Acher, J. M. Jézéquel, L. Noel-Baron, and J. Galindo, "Learning-based performance specialization of configurable systems," Dept. IRISA, Univ. Rennes, Rennes, France, Tech. Rep. hal-01467299v1, 2017.
[14] MM5. *UCAR-Understanding Atmosphere, Earth, and Sun Home*. Accessed: Apr. 3, 2020. [Online]. Available: http://www.mmm.ucar.edu/mm5/
[15] ICENI Project. *Imperial College London, London e-Science Centre*. Accessed: Apr. 3, 2020. [Online]. Available: https://wp.doc.ic.ac.uk/lesc/projects/iceni/
[16] M. Vanneschi. *Marco Vanneschi's Projects*. Accessed: Apr. 3, 2020. [Online]. Available: http://www.di.unipi.it/~vannesch/projects/FIRBGRID.IT/

YUNFENG PENG was born in Anyang, Henan, China, in 1982. He received the B.S. degree in computer science and technology and the Ph.D. degree in computer architecture from the University of Science and Technology Beijing, Beijing, China, in 2005 and 2012, respectively.

From 2012 to 2017, he was a Lecturer with the School of Computer Science and Information Engineering, Anyang Institute of Technology. Since 2018, he has been a Lecturer with the School of Software Engineering, Anyang Normal University, Anyang. He is the author of four books and more than 20 articles. His research interests include parallel computing, software engineering, and component technology.

HAI LIU was born in Anyang, Henan, China, in 1990. He received the B.E. degree in network engineering from Henan Normal University, Henan, China, in 2014, and the master's degree in computer technology from the Yunnan Key Laboratory of Computer Technology Applications, Kunming University of Science and Technology, Yunnan, China, in 2017.

From 2017 to 2018, he was a Lecturer with the School of Computer Science and Information Engineering, Anyang Institute of Technology. Since 2019, he has been a Lecturer with the School of Software Engineering, Anyang Normal University, Anyang. He is the author of more than five articles. His research interests include wireless sensor networks, software engineering, and big data framework.