УДК 004.75 DOi 10.30837/ bi.2021.1(96).02

Kyrychenko I.V.[1], Kolesnyk V.V.[2], Shmelov O.B.[3]

[1]PhD, Senior Lecturer at the Department of Software Engineering, Kharkov National University of Radio Electronics, iryna.kyrychenko@nure.ua, ORCID iD: 0000-0002-7686-6439

[2]Graduate students of the Department of Software Engineering, Kharkov National University of Radio Electronics, valeriia.kolesnyk@nure.ua, ORCID iD: 0000-0001-7811-716X

[3]Graduate students of the Department of Software Engineering, Kharkov National University of Radio Electronics, oleh.shmelov@nure.ua, ORCID iD: 0000-0002-7489-6386

THE USAGE AND IMPLEMENTATION OF PARALLELISM IN GO PROGRAMMING LANGUAGE BASED ON THE MPI INTERFACE AS A MESSAGE EXCHANGE METHOD

The development of the methods for optimizing computer processes by the means of Go programming language. The resources for MPI computations were analyzed from the side of Go programming language. Proposed attempts to fabricate the ties the Go form devices hit their restriction of adaptability quick. Among the advantages of using Go programming language for implementation MPI algorithms, could be saud that it eliminates the need for the developer to manage memory and resources used by software manually, own binaries, fast and efficient compilation. Athough Golang uses several resources to create parallel computations, MPI algorithms implemented by Golang methods and techniques do not fully integrate exchange and computation. Were compared two Jacobi methods for solving partial differential equation. The results showed that Go cannot coordinate the execution of C, although Go scales a part more pleasant when using non-blocking communication when comparing the blocking C usage with the blocking Go execution and and comparing the non-blocking implementations with each other. Go programming language is used for developing massive systems that can speed up software code several times by properly converting sequential algorithms to competing ones, nevertheless MPI developers are not recommended to use it due to its complexity for implementation. As a result, there is currently almost no MPI implemented by Golang methods and techniques that would fully integrate exchange and computation.

MESSAGE PASSING INTERFACE, PARALLEL PROGRAMMING, GO PROGRAMMING LANGUAGE, JACOBI METHOD

І.В. Кириченко, В.В. Колесник, О.Б. Шмельов. Використання і реалізація паралелізму в мові програмування Go на основі інтерфейсу MPI як методу обміну повідомленнями. Розглянуто розробку методів оптимізації ком'ютерних процесів за допомогою засобів мови програмування Go. Проаналізовано ресурси обчислення Інтерфейсу передачі повідомлень, що містить мова програмування Go. Пропонується метод фабрикування зв'язків для аналізу швидкості враження обмеження пристосованості. Серед переваг використання мови програмування Go для реалізації алгоритмів MPI є факт усування необхідності розробнику керувати пам'яттю та ресурсами, що використовуються програмним забезпеченням вручну, власними двійковими файлами, швидкою та ефективною компіляцією. Хоча Golang використовує кілька ресурсів для створення паралельних обчислень, алгоритми MPI, реалізовані методами та техніками Golang, не повністю інтегрують обмін та обчислення. Було порівняно два методи Jacobi для розв'язку рівняння з частковими похідними. Результати показали, що Go не може координувати виконання C, а також, що Go масштабує частину при використанні неблокуючого зв'язку та порівнянні неблокуючих реалізацій між собою. Мова програмування Golang є конкуруючим інструментом розробки послідовні алгоритми на конкуруючі, проте розробникам MPI не рекомендується використовувати його через його складність для реалізації. Як результат, в даний час майже не існує MPI, реалізованого методами та техніками Golang, які б повністю інтегрували обмін та обчислення.

ІНТЕРФЕЙС ПЕРЕДАЧІ ПОВІДОМЛЕНЬ, ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ, МОВА ПРОГРАМУВАННЯ GO, МЕТОД JAKOBI

И.В. Кириченко, В.В. Колесник, О.Б. Шмелев. Использование и реализация параллелизма в языке программирования Go на основе интерфейса MPI как метода обмена сообщениями. Рассмотрена разработка методов оптимизации комъютерних процессов с помощью средств языка программирования Go. Проанализированы ресурсы вычисления интерфейса передачи сообщений, которые содержит язык программирования Go. Предлагается метод фабрикации связей для анализа скорости поражения ограничения приспособленности. Среди преимуществ использования языка программирования Go для реализации алгоритмов MPI может быть выделен факт устранения необходимости разработчику управлять памятью и ресурсами, которые используются программным обеспечением вручную, собственными двоичными файлами, быстрой и эффективной компиляцией. Хотя Golang использует несколько ресурсов для создания параллельных вычислений, алгоритмы MPI, реализованные методы и техники Golang, полностью не интегрируют обмен и вычисления. Сравнены два метода Jacobi для решения уравнения с частными производными. Результаты показали, что Go не может координировать выполнение C, а также, что Go масштабирует часть при использовании неблокирующий связи при и сравнения неблокирующий реализаций между собой. Язык программирования Golang является конкурирующим инструментом разработки последовательные алгоритмы на конкурирующие, однако разработчикам MPI не рекомендуется использовать его за его сложности для реализации. Как результат, в настоящее время почти не существует MPI, реализованного методами и техниками Golang, которые полностью интегрировали обмен и вычисления.

ИНТЕРФЕЙС ПЕРЕДАЧИ СООБЩЕНИЙ, ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ, ЯЗЫК ПРОГРАММИРОВАНИЯ GO, МЕТОД JAKOBI

Introduction

With the development of personal computer processor development technologies and the increasing demand for fast calculations, the need to optimize computer computing processes is growing. To reduce the cost of software support and unit processing, software customers rightly require developers to process as much data as possible in the shortest amount of time. Given the vast amount of stakeholder data available to stakeholders, which is subject to, for example, analysis, effective sequential algorithms alone are not enough to speed up processing.

Many algorithms are based on the sequential execution of certain actions on a list of entities.

Because the actions performed on each element of the processed sequence are not likely to change regardless of the iteration number, such algorithms can be accelerated several times using multi-core processor resources, parallel programming methods and tools, and the Go (or Golang) programming language.

Go programming language, which is quite low-level and has many advantages, such as: static typing, garbage collector - a tool that eliminates the need for the developer to manage memory and resources used by software manually, own binaries, fast and efficient compilation, easy to work with multithreading, the developer can manage and interact, for example, with threads and use all the benefits of MPI.

1. Resources for parallel computations

The definition of the parallelism assumes that an application splits its its tasks into smaller sub-tasks, processing, for example, on multiple processors at the same time. It is well-known fact, that a language that supports multiple processes at the same time is ideal for creating global scalable programs.

Go uses several resources to create parallel computations. One of such specifications is Message Passing Interface (MPI). When working with Go, you need to clearly define the concept of competitiveness. Competitiveness is the ability of various elements of the program, algorithms or tasks to run erratically, or in part without affecting the initial result. This significantly affects the efficiency of the software and the speed of performing computational calculations.

The main tools for working with competition in the Golang programming language are Goroutnies - easily executable methods, or functions that are performed regardless of the calling method or function. The efficiency of Goroutines is largely due to their behavior, which is to plan the processing of a given number of system threads, which, in theory, allows you to process any number of Goroutines. It is worth noting that one of the advantages of using Gorutin in the Go is the minimization of the developer's effort to write code.

Also, one of the basic concepts for working with competitiveness in Go are channels - a way of communication between individual gorits. Channels are a conditional «backbone» for communication and synchronization of performed bursts. With the help of channels, there is no need to create intermediate conditional variables that would control and check each of the executed Goroutines for the completion of a certain stage of the method or its execution. Channels, like any container, must be precreated before use as follows: make (chan int). The given code creates a channel for transmitting integers. To process «events» created by transmitting data to channels from bursts, you can use the select operator, which blocks the execution of bursts until the operation of at least one of the described situations.

2. Golang as MPI computation and its results

Golang is a static typed language that uses a compiler. The program needs to explicitly declare types of variables, so even trivial errors are easily detected. Although an interpreter is available for Go, there is not much need for one, as the compilation speed is high enough to ensure interaction while development.

The Go programming language was developed itself when the multi-core processors existed. It is the main reason for builting-in the overlapping into the Golang. Go has goroutine instead of having the threads. The goroutines consume approximately 2 Kb of memory. That is why goroutines could be activated anytime. Among the advantages of goroutines could be highlated:

– goroutins have segmented extensible stacks that consumes using more memory when necessary.

– goroutines run faster than streams, it happens because of going with built-in primitives to safely share data.

– in case of simultaneous using of data structures, it is not necessary to use mutex locking.

– 1 goroutine can operate freely on multiple streams. Goroutins multiplexin a small number of the streams of operating systems.

The MPI computations in Golang should be written as a special algorithm. Obviously, it needs creation of the function that definitely should be straightforward and wrapper. The algorithm of writing such a function itself includes following:

1. Identification of the inputs and ouputs for Golang supports the return values with multiplication. The organization of the function signatures makes the usage of the multiplication more comprehensible. This feature requires separation the inputs from the ouputs. There are some exceptions as memory regions pointers that are used as buffers for arbitrary data.

2. Dispense with superfluous input/yield parameters: A few MPI capacities anticipate clusters of a particular sort (ordinarily numbers or MPI_-Datatype clusters). In C those capacities anticipate an information pointer as

well as the array size. In Golang we can make utilize cuts instep. Since the length/measure of a cut is a portion of its sort, an additional parameter for the estimate is unnecessary. In this manner, it is in some cases conceivable to decrease the parameter list in a sensible way to maintain a strategic distance from extra duplicate overhead (call by esteem) and keep the marks clear by diminishing their complexity.

3. Declare local variables for the output parameters:

MPI functions that are realized in C just return an error code and expect further output so-called output parameters. Obviously, we make use of the multiple return value feature of Golang. This requires that the wrapper holds a local variable for each output parameter that can be passed in the function.

4. Call the C work inside line sort transformation of the input parameters: before an input parameter can be passed into the C work it needs to be converted to the comparing C sort. Rather than investing extra nearby factors which would require memory and runtime, we straightforwardly change over the input parameters inside the contention list of the C function.

5. Convert the sorts of the neighborhood factors and return their worth to return the qualities, the covering got through the yield boundaries of the C capacity, the nearby factors which hold the information must be changed over to Golang types [1].

Go additionally has a particular form framework. When attempting to fabricate the ties the Go form devices hit their restriction of adaptability quick. There were a few issues when attempting to construct the ties for various executions on a solitary framework. The arrangement was to send a custom form framework written in Python. Go has to know the area of the common article documents of MPI to effectively construct the ties.

The Go form framework can use pkg-config. In any case, on many register groups the common article documents are situated in some sort of custom envelope and can't be found by pkg-config. It is feasible to pass the way of the common item documents to Go by sending out it as a climate variable. The Python assemble script first attempts to decide the way of the common item documents by utilizing pkg-config and afterward adding custom ways that are given to it by the order line. Another issue is that not all MPI executions are totally viable with the standard like they present diverse blunder codes. Since Go doesn't uphold macros the source code should be exceptionally produced.

When introducing the ties, it is important to pass the execution to the form script, so it realizes how to create the source code. Since a process group can have more than one MPI execution introduced, as OpenMPI and MPICH2. It should be feasible to introduce the ties more than once. The form content can produce various libraries of the ties in the event that there is more than one execution present. Taking a gander at the exhibition of the MPI covering and the Go language utilizing a custom benchmark and the different compilers used to make it. The benchmark utilizes the Jacobi strategy to settle incomplete differential conditions.

It is a line-by-line port of the C program bite the dust Go program is contrasted and the parallelism is just documented through the MPI covering. Each MPI cycle just comprises of a solitary goroutine. The benchmark is worked with various compilers and linkers. For example, the default GNU BFD linker, the gold linker created by Google for quicker connecting, and others. After the test, the outcomes showed that Go can't coordinate with the presentation of C however that should have been normal. One fascinating perception is that Go scales significantly more pleasant when utilizing non-obstructing correspondence when contrasting the hindering C execution and the impeding Go execution and contrasting the non-obstructing executions and one another.

Traditional bunch-based frameworks (like supercomputers) utilize equal execution between processors utilizing MPI. MPI is a correspondence interface between measures that execute in working framework examples on various processors; it doesn't uphold other cycle tasks like planning [2].

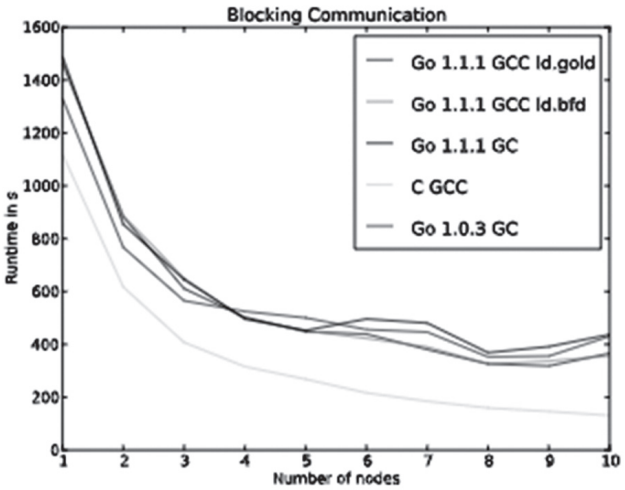

Fig.1. Using the Jacobi method for solving partial differential equation

The point is that Go 1.0.3 is quite faster than 1.1.1. The reason for this could be that Go 1.1.1 repeats slower over cuts than 1.0.3. When taking a gander at the non-obstructing correspondence benchmark clearly the GCC compiler portion preferred improvements over the GC compiler of Go.

The results appear that Go can not coordinate with the execution of C but that was to be anticipated. One curiously perception is that Go scales a part more pleasant when using non-blocking communication when comparing the blocking C usage with the blocking Go execution and and comparing the non-blocking implementations with each other [2].

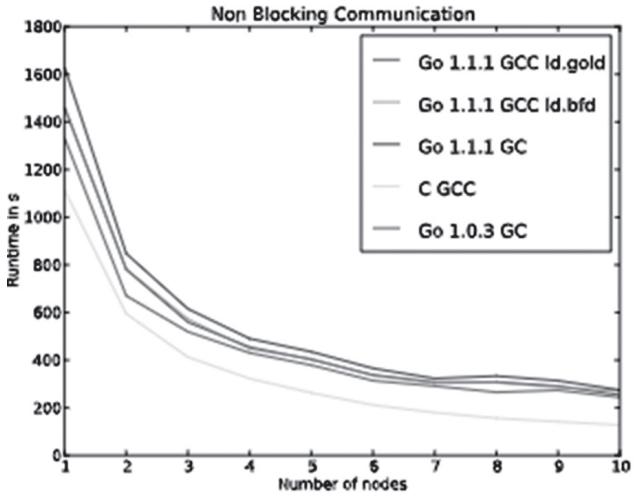

Fig. 2. Using the Jacobi method for solving partial differential equation

Besides all above said nothing was said about communicators, groups, field of communications [3]. A group is a set of branches. One branch may be a member of several groups. MPI_Group type and set of functions working with variables and constants of this type. The constant is actually two of them: MPI_GROUP_EMPTY can be returned if the group with the requested characteristics can be created, but does not yet contain any branch; MPI_GROUP_NULL returns when the requested characteristics are contradictory. According to the MPI concept, once a group is created, it cannot be supplemented or truncated - only a new group can be created under the required set of branches on the basis of the existing one. The field of communication («communication domain») is something abstract: there is no type of data at the programmer's disposal that describes directly the fields of communication, and there are no functions to manage them. Areas of communication are automatically created and destroyed along with communicators. Subscribers of one area of communication are all tasks of either one or two groups. The communicator or, or the communicator area descriptive, is the apex of a three-layer pie (groups, areas of communication, descriptions of communication areas) into which «baked» tasks: it is with communicators that a programmer deals, causing data transmission functions, and most of the support functions.

Why do we need different groups, different areas of communication and different descriptions? In essence, they serve the same purpose as message identifiers - helping the receiver branch and the receiver branch to identify each other more reliably, as well as the content of the message. Branches within a parallel application can be combined into subcollectors to solve intermediate tasks - by creating groups, and areas of communication over groups. Taking advantage of the descriptive of this area of communication, branches are guaranteed not to accept anything from outside subcollectional, and nothing is sent out. In parallel, they may continue to use any other non-subcollectional communicator at their disposal, such as MPI_COMM_WORLD, to exchange data within the entire application. Collective functions create a duplicate of the received communicator argument, and transmit data through a duplicate, without fear that their messages will be inadvertently confused with the messages of the «dot-dot» functions distributed through the original communicator. A programmer for the same purpose in different pieces of code can transmit data between branches through different communicators, one of which was created by copying the other.

Communicators are distributed automatically (by the functions of the «Create a New Comunicator» family), and for them there are no jokers («take it through any communicator») - two more of their essential advantages before message identifiers. Identifiers (integers) are distributed manually by the user and this is the source of two frequent errors due to confusion on the receiving side: Messages with different meanings are mistakenly assigned the same identifier manually. The reception function with the joker collects everything, including those messages that must be received and processed elsewhere in the branch. Intercommunicators and intracommunicators are also important parts in parallel programming in Golang. The descriptions of the communication regions respectively over two groups or over one. MPI_COMM_WORLD is an interagency communicator.

Intercommunicators are not a primary need for beginners, so there is no reference to them outside this paragraph. All the communicator functions mentioned in the document either do not distinguish between «intern» and «intern» at all or explicitly require an «intern». The latter are as follows:

– MPI_Bcast members;

– Topology management functions;

– MPI_Comm_xxx information functions, which are implemented through MPI_Group_xxx, such as MPI_Comm_size;

– MPI_Comm_remote_xxx is used instead [4].

Custom topologies. Inside the problem group are numbered linearly from 0 to (the size of group-1). However, another numbering system can be additionally imposed on them through the communicator. Such additional systems in MPI are two: the Cartesian n-dimensional lattice (cyclically and without), and the bioorientated graph. Functions are provided to create numbering (MPI_Topo_test, MPI_Cart_xxx, MPI_Graph_xxx) and to convert numbers from one system to another. This mechanism should not be perceived as providing an opportunity to adjust branch-to-branch connections to a hardware topology to increase speed; it merely automate the address recalculation that branches are supposed to perform, say, in matrix computation: Through the communicator is given a Cartesian coordinate system, where the coordinates of the branch coincide with the coordinates of the

submatrix it calculates. Multithreading. MPI itself implicitly uses multithreading very widely, and does not prevent the programmer from doing the same. However, different problems have MPI NECESSARILY different numbers, and different threads (threads) within the same problem do not differ [5]. The programmer himself must establish a discipline for threads so that one thread does not, say, cause MPI_Recv to be intercepted by the joker, which must receive and process another thread of the same task. Another source of errors can be the use of different threads of collective functions over the same communicator: use MPI_Comm_dup!File handling. In MPI-2, file redirection tools have been introduced, but not in MPI-1. All function calls are passed directly to the operating system (Unix/Parix/NFS/...) on that machine.

Conclusion:

As the conclusion could be said that nevertheless the Golang programming language is a very promising tool for developing massive systems that can speed up software code several times by properly converting sequential algorithms to competing ones, MPI developers are not recommended to use it due to its complexity for implementation. As a result, there is currently almost no MPI implemented by Golang methods and techniques that would fully integrate exchange and computation.

References:

[1] Gropp W. Using MPI: Portable Parallel Programming with the Message-Passing Interface. / W. Gropp, E. Lusk, S. Anthony – 328 c.

[2] A Golang Wrapper for MPI: https://hps.vi4io.org/_media/teaching/sommersemester_2013/paps-1213-beifuss_weging-golang_bindings_for_mpi-report.pdf

[3] MPI: A Message-Passing Interface Standard Version 3: https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf.

[4] Alexander Beifuss, Johann Weging, Dr. Julian Kunkel. A Golang Wrapper for MPI: http://docplayer.net/50544314-A-golang-wrapper-for-mpi.html.

[5] Alan A. A. Donovan. The Go Programming Language / Alan A. A. Donovan, Brian W. Kernighan.. – 380 c.

[6] I. Balbaert. The Way to Go: A Thorough Introduction to the Go Programming Language. Iuniverse.Com, 3 2012.

The article was delivered to editorial stuff on the 24.02.2021