“The Open TS parallel programming system”

Vladimir Roganov, Alexander Moskovsky, Sergei Abramov.

Program Systems Institute, Russian Academy of Science

Abstract

The paper describes the Open TS parallel programming system, concentrating on execution model and programming language semantics. The Open TS provides compiler and runtime support library for T++ language. The T++ is a “seamless” extension of C++ with few additional keywords, allowing smooth transition from sequential to parallel applications. T++ model is similar to coarse-grain dataflow model, with some extensions including multiple-assignment variables. The Runtime support library is built on top of Message-Passing Interface (MPI), multiple MPI implementations are supported. Open TS currently is available on computational clusters and SMPs running Linux. The paper describes also test re-implementation of two independently developed MPI applications with Open TS, arguing, that, while performance is not substantially hurt, source code is simplified.

keywords: Open TS, parallel computing, programming language, T++

Introduction

Many academic and industry projects exist in the field of high-level parallel programming, many of then are successful [1-13]. In this paper we describe an original Open TS approach that, we believe, has some advantages, with respect to combination of simplicity, smooth transition from the sequential programming and performance. Three ideas are basic for the Open TS:

- Use dataflow-derived programming model
- Extending a sequential programming language (C\C++ , for a first case)
- Parallelization at runtime, using hints from the compile-time.

The remainder of this paper is organized as the following. The programming model and language are described in the first section. Then, implementation is described, and, finally, the case study of comparison between Open TS and Message-Passing Interface (MPI) technologies is presented. The "Open TS" stands for "Open T-system" , so we will use "Open TS" and "T-system" as synonyms. The Open TS is a third version of the "T-system", which effort was initiated in 80-es.

Programming model and language.

The Open TS programming model is very similar to a coarse-grain dataflow model. In T-system, in order to enable the parallel computation for a program, programmer should designate

- Independent parts of the program (parallelism grains, or T-functions, see below)
- Variables, that are used to exchange data between parallelism grains

The rest of parallelization is done by the T-system: both compiler and runtime support library. It should be kept in mind, that grains should be large enough (coarse grains) to keep relatively small overhead, inflicted by the runtime system, so, creating a grain with, say, single floating-point multiplication would not be efficient. The same is (but not so necessarily) true for amount of data in variables.

To create programming languages for Open TS, existing programming languages are extended with extra keywords or pseudo-comments. That gives a chance to utilize existing sequential code while developing parallel applications. Currently, C++ and Fortran Open TS extensions were implemented, Refal [14] version is underway. Here we describe the C++ extension only, which in nicknamed as T++.

The T++ language adds the following keywords to the C++ language:

- **tfun** - a function attribute which should be placed just before the function declaration. Now, the function cannot represent a class method but must be an ordinary C function. A function with the “tfun” attribute is named “T-function”.
- **tval** - a variable type attribute which enables variables to contain a non-ready value. The variable can be cast to the “original” C++-type variable, which makes the thread of execution suspend until the value becomes ready. That it very similar to “dataflow variable” [11] or “mentat variable” [5] or “futures” [15]. That also differs T++ from “standard” data-flow models, where task is ready for execution only after all incoming data are ready – in opposite, threads in Open TS can be launched before any incoming data for a grain are ready.
- **tptr** - a T++ analogue of C++ pointers which can hold references to a non-ready value.
- **tout** - a function parameter attribute used to specify parameters whose values are produced by the function. This is a T++ analog of the “by-reference” parameter passing in C++.
- **tct** - an explicit T-context specification. This keyword is used for specification of additional attributes of T-entities.
- **tdrop** - a T++ -specific macro which makes a variable value ready. It may be very helpful in optimization when it’s necessary to make non-ready values ready before the producer function finishes.
- **twait** - a T++ specific - explicit wait for an argument expression to be ready

The “Hello world” program is very trivial - tfun attribute should be added to the “main” function:

```
tfun int main (int argc, char *argv[])
{
```

```
    printf("Hello world \n");
    return 0;
}
```

No parallel work can be done in that case. The code for recursive calculation the given Fibonacci number is more demonstrative.

```
tfun int fib(int n)
{
    if (n<2) return 1;
    return (fib(n-1)+fib(n-2));
}

tfun int main (int argc, char *argv[])
{
    int n = atoi(argv[1]);
    printf("Fibonacci %d is %d\n",n,(int)fib(n));
    return 0;
}
```

In that case, invocations of "fib" functions are treated as independent tasks, that can be computed in parallel, in independent threads, or on the remote computers/computational cluster nodes.
You can see, that minimal modifications differ the T++ from the C++ code: attributes of T-functions, and explicit cast of “fib” function result to “int”. That casting not only extracts value from T-value, which is returned by “fib”, but also makes “main” function to wait for the “fib” result.

Some specific of the T++ language should be underlined:

- It is a “seamless” C++ extension, that means, evident C++ macrodefinitions of T++ keywords enable T++ program compilation by a C++ compiler. If some good coding style in T++ is adhered to, that compilation will result in correct sequential program.
- Garbage collection is supported. Non-ready values, that are no longer necessary, are detected by run-time system and deallocated.
- Function evaluation can be postponed, not necessarily generating any computation after invocation, depending on execution strategy. By default, if no thread is waiting for function result function evaluation may be omitted.
- The variables support multiple assignments. That supported by the tricky protocol of assignment and readiness of the variable values, related to the thread lifecycle.

The latter feature deserves describing it in a more detail. First of all, variables and variable values should be distinguished from each other. Each T-variable has a link to T-value. Variables are type-safe: it is possible to assign non-ready variables of the same type only. A variable can have multiple values during it's lifetime. Many T-variables may, in principle, share the same T-value. Each value can be either "non-ready" or immutable C-value. Assignment of a T-variable to a C-variable makes execution thread to wait until T-variable's value is ready. Contrary, assignment of C-value to T-variable causes T-value of that variable to be ready.

The T-variable/T-value lifecycle is different on producer and consumer sides. On a producer side:

- A T-value is allocated, when a new T-variable is instantiated. It is assigned to the variable and to context of execution thread, which is a producer of that value. The value is “hot” upon creation – it can be accessed in the producer thread only.

- T-variable is assigned with C-value that makes T-value of that variable is assigned to C-value, and T-value becomes ready.
- When T-variable is used as parameter in the T-function invocation, appropriate T-value's "cold" copy is created and passed to the function. "Cold" values are accessible to all threads, but cannot be changed. "Cold" T-values can be non-ready – then they refer to another T-value, which is a source data, such "cold" values can become ready, when it's source become ready and cold.
- An assignment to the ready variable makes variable to break a link with current T-value and allocate a new value.
- After the thread finishes, all "hot" values allocated in the context of that thread are being "dropped" – available to consumer thread(s). During the drop, T-value is being copied to all linked T-values and "freezes". Linked T-values can be created during T-function invocations (function results), assignment or copying of T-variables. The "drop" can be done explicitly ("tdrop" function).

On a consumer side:

- A variable is initialized with some pre-existing T-value. It is "cold" value, since it's accessible to the consumer thread.
- The variable's C-value is accessed somehow, the thread's execution is suspended, unless C-data are available.
- If the variable is assigned, then the new "hot" value is allocated and associated to the variable.
- The thread finished, variable is destroyed and, finally, when last variable loses a link with a value, the garbage collector deallocates the value's resources (memory and others).

It should be noted, that all parallelism is implicit in T++: a programmer doesn't define, which thread to start and how this threads will cooperate with others, but Open TS. This makes T++ programs easily portable across the multitude of parallel architectures existing: mutli-core CPUs, SMP, computational clusters and grids. Theoretically, T++ code can be compiled even for targets like FPGA or FPGA+CPU.

Implementation

The compilation of T++ language is implemented with the help of two technologies:

1. Conversion T++ to C++ by Open C++ reflection [16] – that gives an option to use best platform-specific optimizing compilers
2. Front-end language to GNU compiler collection – that gives broader language C++ features support that converter utility.

The runtime support library consists of the three layers:

- S- shared memory
- M- mobile objects
- T - T-language entities.

C++ templates are used extensively, e.g. compiling convention of "tval int" if "ts::TVal<int>", which makes Open TS kernel similar to OMPC++[17]. The MPI is used for data exchange in cluster environment, with multiple MPI implementations support and dynamic (runtime) choosing of implementation. MPI implementations supported include LAM MPI, MPICH, MVAPICH and others. Totally, the runtime support library source code is approximately 5000 lines.

A runtime library has many tools to facilitate development. The simplest is based on WAD signal handler [18], which dumps a stack in case of segmentation fault or critical signal. More complex is a lightweight debugger (GNU Debugger), which can be started in case of critical signal, and support for TDB[19] parallel program debugger.

Fast context switch is a special feature of Open TS, which is very important for efficient T-applications. Since T-applications are known to create millions of simultaneous threads, fast switching is key important. Today, the T-context switch is 10 times faster than the fastest standard thread library switch.

Another important implementation feature is a work migration model. In cluster environment, each MPI process has it's own set of tasks and it’s own meta-scheduler instance (thread) running. Tasks are T-function invocations (with some exceptions: sometimes it may be more efficient to evaluate T-function call immediately. So, T-functions are potential parallelism grains, not inevitable). Tasks can be either running or pre-natal. In a general case, pre-natal tasks can be moved between nodes, while running tasks cannot. Macro-scheduler can make following decisions:

1) Send one or more pre-natal tasks o another node(s)
2) Continue one of running tasks
3) Execute a pre-natal task, thus making task running.

When executing, task can invoke T-functions, thus creating more tasks (possibly, of another type). Optimal load balancing is achieved with the help of heuristic algorithms or algorithms, specific to certain applications.

Currently, Open TS runs only on Linux platform on x86 or AMD 64 processors , however, development of the Microsoft Windows version is planned.

MPI vs Open TS case study.

In order to evaluate the programming technique as a whole, not only the runtime support library effectiveness and scalability is an issue, but programming language qualities as well. Despite programming language preferences is subjective matter at high extent, we believe, some sharp differences in code statistics, such as the number of lines of code (LOC), can be informative. The case study has been undertaken, involving two parallel applications, initially developed by the independent groups of parallel programming experts: MPI enabled version of Persistence of Vision Ray-tracing (POV Ray) program and Ames Lab Classical Molecular Dynamics (ALCMD) [20] code.

T-PovRay

The famous POV Ray (Persistence Of Vision) application is widely used to obtain realistic images using ray-tracing rendering technology. POV-Ray is freely distributed with source code, evolved from C to C++ implementation during last years. Since Ray-tracing consumes a lot of computation resources even for simple scenes, few parallel versions of POV-Ray have been developed and contributed by different authors. There are few approaches used to parallelize POV-Ray work on multicomputers: from trivial rsh-based scripts, invoking POV-Ray executable for parts of target scene on different unix hosts, to most effective PVM and MPI-based implementations, supporting dynamic load balancing and even original features like animation and interactive display functions.

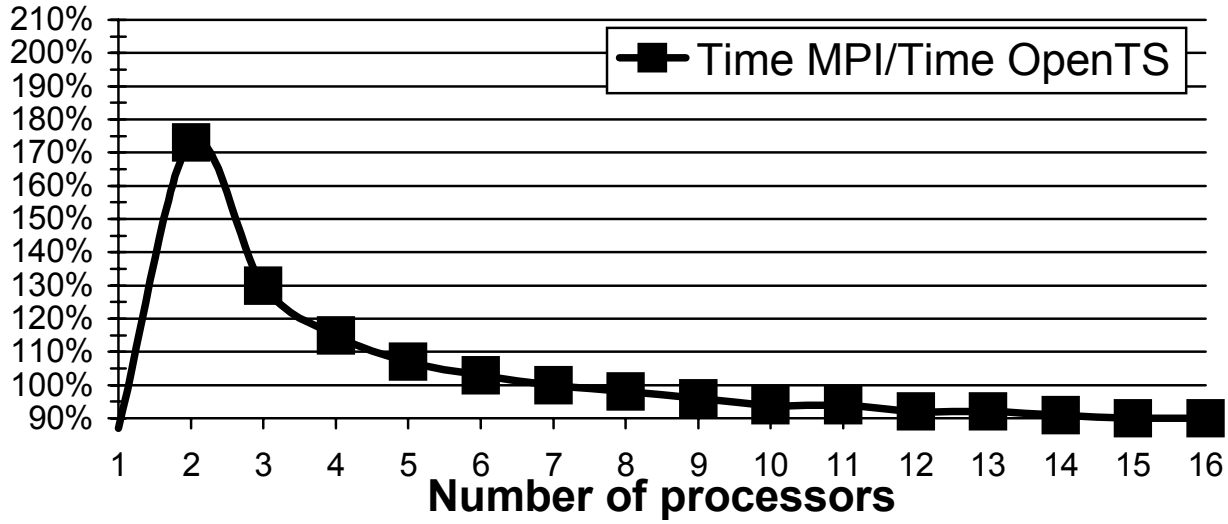
There are two most known MPI-based POV-Ray ports:

- MPI-POV-Ray based on POV-Ray 3.1g, http://www.verrall.demon.co.uk/mpipov/;
- ParaPov based on POV-Ray 3.50c
  http://news.povray.org/povray.binaries.programming/
  thread/%3C3E40469B.3080402@web.de%3E

Both MPI-ports are really just a 'kind of patch' for the corresponding vanilla POV-Ray source code. A significant disadvantage: 'phantom master' mode was not implemented at the moment of testing (July 2005), when master processor is also working as a renderer (slave)..

Total size of POV-Ray 3.1g MPI-related source files (mpipov.c and mpipov.h) is more than 1500 lines of code, with multiple changes scattered over many files. However, an intention to minimize changes in POV-Ray code resulted in coding style, that, we think, sometimes is challenging to the reviewer. POV-Ray 3.50c MPI patch express parallelization idea in more straightforward C++, about 3000 lines total.

To make comparison more correct, we made our code applicable to the same POV-Ray versions (both 3.1g and 3.50c). OpenTS port is straightforward: most of porting work consists in removing unnecessary task management MPI-code, replacing it by only two T-functions. Source file tpovray.tcc is shorter than 200 lines; with a few minor changes in file povray.c.



Performance comparison has been done with the "chess board scene" (scenes/advanced/chess2.pov taken from original POV-Ray distribution) with the scene width and height set to 1500. The chess board scene has 430 primitives in objects and 41 in bounding shapes. The graph, displaying the ratio between execution times MPI POV-Ray 3.50c and Open TS is presented above.

The computational cluster used had the following configuration:

- operating system — Linux, kernel release — 2.4.27
- 16 nodes; each cluster node: 2CPUs AMD Athlon MP 1800+ RAM 1GB, HDD 40GB

The performance advantage of Open TS version is due to sub-optimal load balancing of MPI version, one CPU is reserved for management work, that advantage gradually degrades when number of CPU increases.

ALCMD

Ames Lab Molecular Dynamics package is a parallel program for conducting simulation of dynamics of properties of atomic substances. It has two layers: a Fortran code, implementing solution of differential equations, and MP_Lite library, an original MPI implementation, which currently has some custom collective operations, facilitating molecular dynamics simulations.

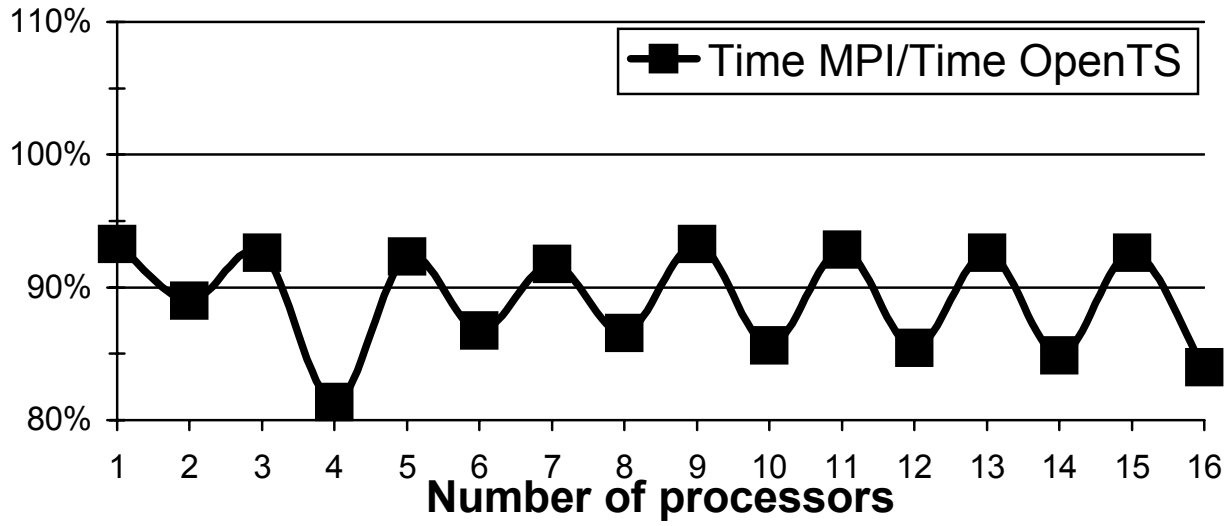
Original size of MP_Lite library is more than 20.000 lines of C code (including ability to work over shared memory and so on).

We selected from MP_Lite the subset of the functions that are used in ALCMD Fortran code. Reduced (sometimes by hands) subset of MP_Lite source code, which enables to run ALCMD over MPI is really about 3500 lines long. Following 31 functions are really required to run ALCMD (most MP_Lite functions are self-documented, see MP_Lite manual and source code for details):

MP_Init	MP_iBroadcast	MP_iSum	MP_SRecv
MP_Myproc	MP_Sync	MP_iSum_masked	MP_iSend
MP_Nprocs	MP_dMax	MP_Enter	MP_SSend
MP_Finalize	MP_dMin	MP_Leave	MP_SShift
MP_Remap	MP_dSum	MP_Zero_Time	MP_Get_Token
MP_Remap_Setup	MP_iMax	MP_Time_Report	MP_Pass_Token
MP_Broadcast	MP_iMax_masked	MP_iARecv	MP_Bedcheck
MP_dBroadcast	MP_iMin	MP_Wait	

Supporting all functions listed above our MP_Lite abstraction layer implementation (MP_Lite_{OpenTS}) contains less than 500 lines of code in T++ language — comparing with 3500 lines of original code on C over MPI, thanks to expressive power of C++ templates.

We used a simulation including 512 000 atoms with Lennard-Jones interaction potential as a test case. The comparison of execution times is presented on a graph below, using the cluster configuration described above.



Conclusion

We have described a parallel programming approach of Open T-system (Open TS). Open TS actively gains auditorium during last years, over 20 various parallel applications have been developed with Open TS technology, including

- Aerodynamics simulation package
- Tools for computational modeling in chemistry [21]
- Automatic text categorization package
- Dynamics of fluid simulation code [22]
- Remote sensing images processing

We believe, that Open TS programming model simplifies job of parallel programs development. Further steps in that area could include development of “templates” of parallel algorithms, capturing most widely used ways to parallelize computation.

Acknowledgements

This work is supported by basic research grant from Russian Academy of Science program "High-performance computing systems on new principles of computational process organization" and basic research program of Presidium of Russian Academy of Science "Development of basics for implementation of distributed scientific informational-computational environment on GRID technologies", as well as Russian Foundation of Basic Research grant 05-07-08005-ofi_a. Authors also thanks Microsoft corp. for support.

As well, we express our gratitude to Igor Zagorovsky, German Matveev, Alexandr Inyukhin, Alexandr Vodmomerov, Eugene Stepanov, Ilya Konev, Elena Shevchuk, Yuri Shevchuk, Alexei Adamovich, Philip Koryaka and others, who contributed to the implementation and development of Open TS and previous T-system versions.

Literature

1. William W. Carlson, Jesse M. Draper, David E. Culler, Kathy Yelick, Eugene Brooks, Karen Warren «Introduction to UPC and Language Specification» http://www.cs.berkeley.edu/~yelick/yelick/upctr.pdf
2. Kathleen Knobe, Carl D. Offner, "TStreams: How to Write a Parallel Program", Technical report, HP Labs Technical Report HPL-2004-193, http://www.hpl.hp.com/techreports/2004/HPL-2004-193.pdf
3. ,"The CxC 2.0 Parallel Programming Guide", 2002-2003 Engineered Intelligence Corporation.

4. Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Mass, Thomas Radke1, Edward Seidel, and John Shalf “The Cactus Framework and Toolkit: Design and Applications”, LNCS v 2565 / 2003, VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002., Selected Papers and Invited Talks pp 197-227, .
5. A. S. Grimshaw, "Easy to Use Object-Oriented Parallel Programming with Mentat," IEEE Computer, pp. 39-51,May, 1993.
6. Gregory V. Wilson (Editor), Paul Lu (Editor) “Parallel Programming Using C++” MIT Press, 1996
7. L. V. Kaleev, Sanjeev Krishnan “Charm++: Parallel Programming with Message-Driven Objects” in [5] pp 175-213/, also http://charm.cs.uiuc.edu/
8. Keith H. Randall “Cilk: Efficient Multithreaded Computing”,. Ph. D. Thesis, MIT Department of Electrical Engineering and Computer Science. June 1998. http://supertech.lcs.mit.edu/cilk/
9. Pointon R.F. Trinder P.W. Loidl H-W. “The Design and Implementation of Glasgow distributed Haskell:” IFL'00 - 12th International Workshop on the Implementation of Functional Languages, Aachen, Germany (September 2000) Springer Verlag LNCS 2011, pp 53-70
10. Alexey Lastovetsky, "mpC - a Multi-Paradigm Programming Language for Massively Parallel Computers", ACM SIGPLAN Notices, 31(2):13-20, February 1996
11. Gert Smolka , “The Development of Oz and Mozart”, LNCS, v 3389, 2005, Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Charleroi, Belgium, October 7-8, 2004, Revised Selected and Invited Papers, ed. Peter Van Roy , p 1.
12. E-language at http://www.erights.org/index.html

13. Robert H. Halstead, Jr.” MULTILISP: a language for concurrent symbolic computation”, ACM Transactions on Programming Languages and Systems (TOPLAS) , Volume 7,Issue 4 (October 1985) Pages: 501 - 538
14. "REFAL-5 programming guide and reference manual" by Valentin F. Turchin, New England Publishig Co., Holyoke 1989
15. http://en.wikipedia.org/wiki/Future_%28programming%29
16. Chiba S. A Metaobject Protocol for C++ , In Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), page 285-299, October 1995. http://www.csg.is.titech.ac.jp/~chiba/openc++.html
17. Yukihiko Sohda, Hirotaka Ogawa, Satoshi Matsuoka “OMPC++ - A Portable High-Performance Implementation of DSM using OpenC++ Reflection” Lecture Notes In Computer Science; Vol. 1616 pp 215-234 Proceedings of the Second International Conference on Meta-Level Architectures and Reflection , 1999
18. David M. Beazley, “An Embedded Error Recovery and Debugging Mechanism for Scripting Language Extensions”, USENIX 2001 , Pp. 147–160 of the Proceedings , http://www.usenix.org/publications/library/proceedings/usenix01/full_papers/beazley/beazley_html
19. A. I. Adamovich, M.R. Kovalenko, “TDB-open distributed program system for interactive debugging MPI-programs: architecture and basic principles”, Proc. Supercomputing Systems and Applications SSA'2004. international conference, 26-28 October 2004, Minsk, Belarus, pp. 147-151
20. ALCMD: The Ames Lab Classical Molecular Dynamics code http://www.cmp.ameslab.gov/cmp/CMP_Theory/cmd/cmd.html

21. (a) Potemkin V.A. Arslambekov R.M. Belik A.V. Guccione S. “A parallel version of MULTIGEN algorithm” in proceedings “Computer Application in Scientific Researches” IVTN-2003, p. 11. , www.ivtn.ru, (in Russian)

(b) Moskovsky A.A., Vanovsky V.V., Granovsky A.A., Nemukhin A.V., “Development of a two-level parallelism schema for distributed modelling of protein folding.”, International Conference “Distributed Computing and Grid Technologies in Science and Education” 29 June - 2 July 2004, Dubna, Russia

22. A. Kornev, “ On globally stable dynamic processes” //Russian Journal of Numerical Analysis and Mathematical Modelling, Volume 17, No. 5, p 472